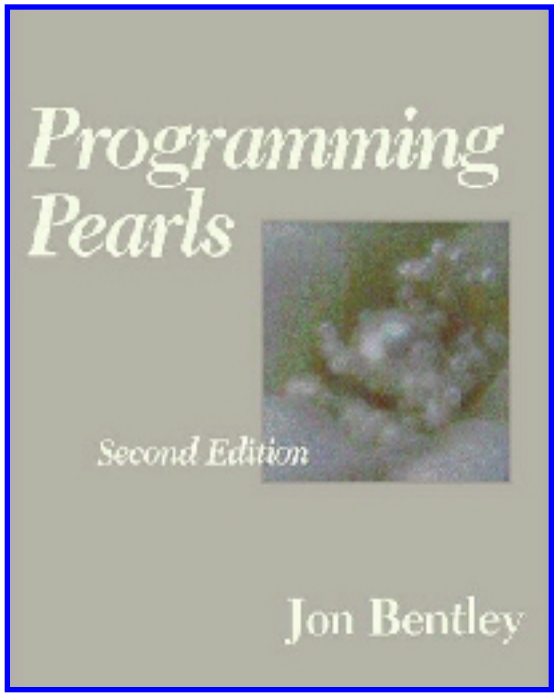
Code from Programming Pearls



- Column 1: Programs for sorting integers
  bitsort.c -- Sort with bit vectors.
  sortints.cpp -- Sort using C++ STL sets.
  qsortints.c -- Sort with C library qsort.
  bitsortgen.c -- Generate random integers for sorting.

- Column 2: Test and time algorithms
  rotate.c -- Three ways to rotate the elements of a vector.
  The next two program are used in a pipeline to compute all anagrams in a dictionary
  sign.c -- Sign each word by its letters in sorted order.
  squash.c -- Put each anagram class on a single line.

- Column 5: Scaffolding for testing and timing search functions
  search.c -- Linear and binary search.

- Column 7: Tiny experiment on C run times
  timemod0.c -- Edit main to time one operation.

- Column 8: Compute the maximum-sum subsequence in an array
  maxsum.c -- Time four algs: n^3, n^2, n log n, n.

- Column 9: Code tuning programs
  genbins.c -- Profile this, then try a special-purpose allocator.
  macfun.c -- Time the cost of macros and functions.
  The column also uses rotate.c (Column 2), search.c (Column 5) and maxsum.c (Column 8).

- Column 11: Test and time sorting algorithms
  sort.cpp -- Mostly C, but also C++ sort function.
  SortAnim.java -- Animate those sort functions in Java.

- Column 12: Generate a sorted list of random integers
  sortedrand.cpp -- Several algorithms for the task.

- Column 13: Set representations for the problem in Column 12
  sets.cpp -- Several data structures for sets.
  genbins.c (Column 9) implements the bin data structure in C.

- Column 14: Heaps

priqueue.cpp -- Implement and test priority queues.
The column also uses sort.c (Column 11) for heapsort.

- Column 15: Strings
  wordlist.cpp -- List words in the file, using STL set.
  wordfreq.cpp -- List words in the file, with counts, using STL map.
  wordfreq.c -- Same as above, with hash table in C.
  longdup.c -- Find long repeated strings in input.
  markov.c -- Generate random text from input.
  markovhash.c -- Like markov.c, but with hashing.
  markovlet.c -- Letter-level markov text, simple algorithm.

- Appendix 3: Cost Models
  spacemod.cpp -- Space used by various records.
  timemod.c -- Table of times used by various C constructs.

You may use this code for any purpose, as long as you leave the copyright notice and book citation attached.

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* bitsort.c -- bitmap sort from Column 1
 *   Sort distinct integers in the range [0..N-1]
 */

#include <stdio.h>

#define BITSPERWORD 32
#define SHIFT 5
#define MASK 0x1F
#define N 10000000
int a[1 + N/BITSPERWORD];

void set(int i) {        a[i>>SHIFT] |=  (1<<(i & MASK)); }
void clr(int i) {        a[i>>SHIFT] &= ~(1<<(i & MASK)); }
int  test(int i){ return a[i>>SHIFT] &   (1<<(i & MASK)); }

int main()
{       int i;
        for (i = 0; i < N; i++)
                clr(i);
/*      Replace above 2 lines with below 3 for word-parallel init
        int top = 1 + N/BITSPERWORD;
        for (i = 0; i < top; i++)
                a[i] = 0;
 */
        while (scanf("%d", &i) != EOF)
                set(i);
        for (i = 0; i < N; i++)
                if (test(i))
                        printf("%d\n", i);
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* sortints.cpp -- Sort input set of integers using STL set */

#include <iostream>
#include <set>
using namespace std;

int main()
{       set<int> S;
        int i;
        set<int>::iterator j;
        while (cin >> i)
                S.insert(i);
        for (j = S.begin(); j != S.end(); ++j)
                cout << *j << "\n";
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* qsortints.c -- Sort input set of integers using qsort */

#include <stdio.h>
#include <stdlib.h>

int intcomp(int *x, int *y)
{   return *x - *y;
}

int a[1000000];

int main()
{   int i, n=0;
    while (scanf("%d", &a[n]) != EOF)
        n++;
    qsort(a, n, sizeof(int), intcomp);
    for (i = 0; i < n; i++)
        printf("%d\n", a[i]);
    return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* bitsortgen.c -- gen $1 distinct integers from U[0,$2) */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXN 2000000
int x[MAXN];

int randint(int a, int b)
{       return a + (RAND_MAX * rand() + rand()) % (b + 1 - a);
}

int main(int argc, char *argv[])
{       int i, k, n, t, p;
        srand((unsigned) time(NULL));
        k = atoi(argv[1]);
        n = atoi(argv[2]);
        for (i = 0; i < n; i++)
                x[i] = i;
        for (i = 0; i < k; i++) {
                p = randint(i, n-1);
                t = x[p]; x[p] = x[i]; x[i] = t;
                printf("%d\n", x[i]);
        }
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* rotate.c -- time algorithms for rotating a vector
        Input lines:
                algnum numtests n rotdist
                algnum:
                  1: reversal algorithm
                  2: juggling algorithm
                  22:  juggling algorithm with mod rather than if
                  3: gcd algorithm
                  4: slide (don't rotate): baseline alg for timing
        To test the algorithms, recompile and change main to call testrot
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXN 10000000

int x[MAXN];
int rotdist, n;

/* Alg 1: Rotate by reversal */

void reverse(int i, int j)
{       int t;
        while (i < j) {
                t = x[i]; x[i] = x[j]; x[j] = t;
                i++;
                j--;
        }
}

void revrot(int rotdist, int n)
{       reverse(0, rotdist-1);
        reverse(rotdist, n-1);
        reverse(0, n-1);
}

/* Alg 2: Juggling (dolphin) rotation */

int gcd(int i, int j)
{       int t;
        while (i != 0) {
                if (j >= i)
                        j -= i;
                else {
                        t = i; i = j; j = t;
                }
        }
        return j;
}

void jugglerot(int rotdist, int n)
{       int cycles, i, j, k, t;
        cycles = gcd(rotdist, n);
        for (i = 0; i < cycles; i++) {
                /* move i-th values of blocks */
                t = x[i];
                j = i;
```

```
                for (;;) {
                        k = j + rotdist;
                        if (k >= n)
                                k -= n;
                        if (k == i)
                                break;
                        x[j] = x[k];
                        j = k;
                }
                x[j] = t;
        }
}

void jugglerot2(int rotdist, int n)
{       int cycles, i, j, k, t;
        cycles = gcd(rotdist, n);
        for (i = 0; i < cycles; i++) {
                /* move i-th values of blocks */
                t = x[i];
                j = i;
                for (;;) {
          /* Replace with mod below
                        k = j + rotdist;
                        if (k >= n)
                                k -= n;
           */
                         k = (j + rotdist) % n;
                        if (k == i)
                                break;
                        x[j] = x[k];
                        j = k;
                }
                x[j] = t;
        }
}

/* Alg 3: Recursive rotate (using gcd structure) */

void swap(int i, int j, int k) /* swap x[i..i+k-1] with x[j..j+k-1] */
{       int t;
        while (k-- > 0) {
                t = x[i]; x[i] = x[j]; x[j] = t;
                i++;
                j++;
        }

}

void gcdrot(int rotdist, int n)
{       int i, j, p;
        if (rotdist == 0 || rotdist == n)
                return;
        i = p = rotdist;
        j = n - p;
        while (i != j) {
                /* invariant:
                        x[0  ..p-i  ] is in final position
                        x[p-i..p-1  ] = a (to be swapped with b)
                        x[p  ..p+j-1] = b (to be swapped with a)
                        x[p+j..n-1  ] in final position
                */
```

```
                if (i > j) {
                        swap(p-i, p, j);
                        i -= j;
                } else {
                        swap(p-i, p+j-i, i);
                        j -= i;
                }
        }
        swap(p-i, p, i);
}

int isogcd(int i, int j)
{       if (i == 0) return j;
        if (j == 0) return i;
        while (i != j) {
                if (i > j)
                        i -= j;
                else
                        j -= i;
        }
        return i;
}

void testgcd()
{
        int i,j;
        while (scanf("%d %d", &i, &j) != EOF)
                printf("%d\n", isogcd(i,j) );
}

/* Test all algs */

void slide(int rotdist, int n) /* Benchmark: slide left rotdist (lose 0..rotdist-1) */
{       int i;

        for (i = rotdist; i < n; i++)
                x[i-rotdist] = x[i];
}

void initx()
{
        int i;
        for (i = 0; i < n; i++)
                x[i] = i;
}

void printx()
{       int i;
        for (i = 0; i < n; i++)
                printf(" %d", x[i]);
        printf("\n");
}

void roterror()
{
        fprintf(stderr, " rotate bug %d %d\n", n, rotdist);
        printx();
        exit (1);
}

void checkrot()
{       int i;
```

```
        for (i = 0; i < n-rotdist; i++)
                if (x[i] != i+rotdist)
                        roterror();
        for (i = 0; i < rotdist; i++)
                if (x[n-rotdist+i] != i)
                        roterror();
}

void testrot()
{       for (n = 1; n <= 20; n++) {
                printf(" testing n=%d\n", n);
                for (rotdist = 0; rotdist <= n; rotdist++) {
                        /* printf("  testing rotdist=%d\n", rotdist); */
                        initx(); revrot(rotdist, n);     checkrot();
                        initx(); jugglerot(rotdist, n);  checkrot();
                        initx(); jugglerot2(rotdist, n); checkrot();
                        initx(); gcdrot(rotdist, n);     checkrot();
                }
        }
}

/* Timing */

void timedriver()
{       int i, algnum, numtests, start, clicks;
        while (scanf("%d %d %d %d", &algnum, &numtests, &n, &rotdist) != EOF) {
                initx();
                start = clock();
                for (i = 0; i < numtests; i++) {
                        if (algnum == 1)
                                revrot(rotdist, n);
                        else if (algnum == 2)
                                jugglerot(rotdist, n);
                        else if (algnum == 22)
                                jugglerot2(rotdist, n);
                        else if (algnum == 3)
                                gcdrot(rotdist, n);
                        else if (algnum == 4)
                                slide(rotdist, n);
                }
                clicks = clock() - start;
                printf("%d\t%d\t%d\t%d\t%d\t%g\n",
                        algnum, numtests, n, rotdist, clicks,
                        1e9*clicks/((float) CLOCKS_PER_SEC*n*numtests));
        }
}

/* Main */

int main()
{       /* testrot(); */
        timedriver();
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* sign.c -- sign each line of a file for finding anagrams
    The input line "stop" gives the output line "opst stop"
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define WORDMAX 100

int charcomp(char *x, char *y)
{   return *x - *y;
}

int main()
{   char word[WORDMAX], sig[WORDMAX];
    while (scanf("%s", word) != EOF) {
        strcpy(sig, word);
        qsort(sig, strlen(sig), sizeof(char), charcomp);
        printf("%s %s\n", sig, word);
    }
    return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* squash.c -- print anagram classes on a single line
    The input lines "opst pots" and "opst stop" go to "pots stop"
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define WORDMAX 100

int main()
{   char word[WORDMAX], sig[WORDMAX], oldsig[WORDMAX];
    int linenum = 0;
    strcpy(oldsig, "");
    while (scanf("%s %s", sig, word) != EOF) {
        if (strcmp(oldsig, sig) != 0 && linenum > 0)
            printf("\n");
        strcpy(oldsig, sig);
        linenum++;
        printf("%s ", word);
    }
    printf("\n");
    return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* search.c -- test and time binary and sequential search
   Select one of three modes by editing main() below.
   1.) Probe one function
   2.) Test one function extensively
   3.) Time all functions
               Input lines:  algnum n numtests
               Output lines: algnum n numtests clicks nanosecs_per_elem
               See timedriver for algnum codes
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXN 1000000

typedef int DataType;

DataType x[MAXN];
int n;

/* Scaffolding */

int i = -999999;
#define assert(v) { if ((v) == 0) printf("  binarysearch bug %d %d\n", i, n); }

/* Alg 1: From Programming Pearls, Column 4: raw transliteration */

int binarysearch1(DataType t)
{       int l, u, m;
        l = 0;
        u = n-1;
        for (;;) {
                if (l > u)
                        return -1;
                m = (l + u) / 2;
                if (x[m] < t)
                        l = m+1;
                else if (x[m] == t)
                        return m;
                else /* x[m] > t */
                        u = m-1;
        }
}

/* Alg 2: Make binarysearch1 more c-ish */

int binarysearch2(DataType t)
{       int l, u, m;
        l = 0;
        u = n-1;
        while (l <= u) {
                m = (l + u) / 2;
                if (x[m] < t)
                        l = m+1;
                else if (x[m] == t)
                        return m;
                else /* x[m] > t */
                        u = m-1;
```

```
        }
        return -1;
}

/* Alg 3: From PP, Col 8 */

int binarysearch3(DataType t)
{       int l, u, m;
        l = -1;
        u = n;
        while (l+1 != u) {
                m = (l + u) / 2;
                if (x[m] < t)
                        l = m;
                else
                        u = m;
        }
        if (u >= n || x[u] != t)
                return -1;
        return u;
}

/* Alg 4: From PP, Col 9 */

int binarysearch4(DataType t)
{       int l, p;
        if (n != 1000)
                return binarysearch3(t);
        l = -1;
        if (x[511]   < t) l = 1000 - 512;
        if (x[l+256] < t) l += 256;
        if (x[l+128] < t) l += 128;
        if (x[l+64 ] < t) l += 64;
        if (x[l+32 ] < t) l += 32;
        if (x[l+16 ] < t) l += 16;
        if (x[l+8  ] < t) l += 8;
        if (x[l+4  ] < t) l += 4;
        if (x[l+2  ] < t) l += 2;
        if (x[l+1  ] < t) l += 1;
        p = l+1;
        if (p >= n || x[p] != t)
                return -1;
        return p;
}

/* Alg 9: Buggy, from Programming Pearls, Column 5 */

int sorted()
{   int i;
    for (i = 0; i < n-1; i++)
        if (x[i] > x[i+1])
            return 0;
    return 1;
}

int binarysearch9(DataType t)
{       int l, u, m;
/* int oldsize, size = n+1; */
        l = 0;
        u = n-1;
        while (l <= u) {
/* oldsize = size;
```

```
size = u - l +1;
assert(size < oldsize); */
                m = (l + u) / 2;
/* printf("  %d %d %d\n", l, m, u); */
                if (x[m] < t)
                        l = m;
                else if (x[m] > t)
                        u = m;
                else {
                        /* assert(x[m] == t); */
                        return m;
                }
        }
        /* assert(x[l] > t && x[u] < t); */
        return -1;
}

/* Alg 21: Simple sequential search */

int seqsearch1(DataType t)
{       int i;
        for (i = 0; i < n; i++)
                if (x[i] == t)
                        return i;
        return -1;
}

/* Alg 22: Faster sequential search: Sentinel */

int seqsearch2(DataType t)
{       int i;
        DataType hold = x[n];
        x[n] = t;
        for (i = 0; ; i++)
                if (x[i] == t)
                        break;
        x[n] = hold;
        if (i == n)
                return -1;
        else
                return i;
}

/* Alg 23: Faster sequential search: loop unrolling */

int seqsearch3(DataType t)
{       int i;
        DataType hold = x[n];
        x[n] = t;
        for (i = 0; ; i+=8) {
                if (x[i] == t)   {          break; }
                if (x[i+1] == t) { i += 1; break; }
                if (x[i+2] == t) { i += 2; break; }
                if (x[i+3] == t) { i += 3; break; }
                if (x[i+4] == t) { i += 4; break; }
                if (x[i+5] == t) { i += 5; break; }
                if (x[i+6] == t) { i += 6; break; }
                if (x[i+7] == t) { i += 7; break; }
        }
        x[n] = hold;
        if (i == n)
                return -1;
```

```
        else
                return i;
}


/* Scaffolding to probe one algorithm */

void probe1()
{       int i;
        DataType t;
        while (scanf("%d %d", &n, &t) != EOF) {
                for (i = 0; i < n; i++)
                        x[i] = 10*i;
                printf(" %d\n", binarysearch9(t));
        }
}


/* Torture test one algorithm */

#define s seqsearch3
void test(int maxn)
{       int i;
        for (n = 0; n <= maxn; n++) {
                printf("n=%d\n", n);
                /* distinct elements (plus one at top) */
                for (i = 0; i <= n; i++)
                        x[i] = 10*i;
                for (i = 0; i < n; i++) {
                        assert(s(10*i)     ==  i);
                        assert(s(10*i - 5) == -1);
                }
                assert(s(10*n - 5) == -1);
                assert(s(10*n)     == -1);
                /* equal elements */
                for (i = 0; i < n; i++)
                        x[i] = 10;
                if (n == 0) {
                        assert(s(10) == -1);
                } else {
                        assert(0 <= s(10) && s(10) < n);
                }
                assert(s(5) == -1);
                assert(s(15) == -1);
        }
}

/* Timing */

int p[MAXN];

void scramble(int n)
{       int i, j;
        DataType t;
        for (i = n-1; i > 0; i--) {
                j = (RAND_MAX*rand() + rand()) % (i + 1);
                t = p[i]; p[i] = p[j]; p[j] = t;
        }
}

void timedriver()
{       int i, algnum, numtests, test, start, clicks;
```

```
        while (scanf("%d %d %d", &algnum, &n, &numtests) != EOF) {
                for (i = 0; i < n; i++)
                        x[i] = i;
                for (i = 0; i < n; i++)
                        p[i] = i;
                scramble(n);
                start = clock();
                for (test = 0; test < numtests; test++) {
                        for (i = 0; i < n; i++) {
                                switch (algnum) {
                                case 1:  assert(binarysearch1(p[i]) == p[i]); break;
                                case 2:  assert(binarysearch2(p[i]) == p[i]); break;
                                case 3:  assert(binarysearch3(p[i]) == p[i]); break;
                                case 4:  assert(binarysearch4(p[i]) == p[i]); break;
                                case 9:  assert(binarysearch9(p[i]) == p[i]); break;
                                case 21: assert(seqsearch1(p[i]) == p[i]); break;
                                case 22: assert(seqsearch2(p[i]) == p[i]); break;
                                case 23: assert(seqsearch3(p[i]) == p[i]); break;
                                }
                        }
                }
                clicks = clock() - start;
                printf("%d\t%d\t%d\t%d\t%g\n",
                        algnum, n, numtests, clicks,
                        1e9*clicks/((float) CLOCKS_PER_SEC*n*numtests));
        }
}

/* Main */

int main()
{       /* probe1(); */
        /* test(25); */
        timedriver();
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* timemod0.c -- Simple experiments on C run time costs */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{       int i, n, ia, ib, ic;
        float fa, fb, fc;
        n = 1000000000; /* run time in secs gives nanosecs/loop */
        ia = ib = ic = 9;
        fa = fb = 9.0;
        for (i = 0; i < n; i++) {
                /* null loop            19.1 */
                /* ia = ib + ic;        17.7 */
                /* ia = ib - ic;        17.6 */
                /* ia = ib * ic;        17.7 */
                /* ia = ib % ic;        98.3 */
                /* ia = ib / ic;        98.3 */
                /* ia = rand();         41.5 */
                /* fa = sqrt(fb);       184 */
                /* free(malloc(8));     2400 */
        }
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* maxsum.c -- time algs for maximum-sum subsequence
 * Input:  algnum, n
 * Output: algnum, n, answer, ticks, secs
 *              See main for algnum codes
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXN 10000000
int n;
float x[MAXN];

void sprinkle() /* Fill x[n] with reals uniform on [-1,1] */
{   int i;
    for (i = 0; i < n; i++)
        x[i] = 1 - 2*( (float) rand()/RAND_MAX);
}

float alg1()
{   int i, j, k;
    float sum, maxsofar = 0;
    for (i = 0; i < n; i++)
        for (j = i; j < n; j++) {
            sum = 0;
            for (k = i; k <= j; k++)
                sum += x[k];
            if (sum > maxsofar)
                maxsofar = sum;
        }
    return maxsofar;
}

float alg2()
{   int i, j;
    float sum, maxsofar = 0;
    for (i = 0; i < n; i++) {
        sum = 0;
        for (j = i; j < n; j++) {
            sum += x[j];
            if (sum > maxsofar)
                maxsofar = sum;
        }
    }
    return maxsofar;
}

float cumvec[MAXN+1];

float alg2b()
{   int i, j;
    float *cumarr, sum, maxsofar = 0;
    cumarr = cumvec+1; /* to access cumarr[-1] */
    cumarr[-1] = 0;
    for (i = 0; i < n; i++)
        cumarr[i] = cumarr[i-1] + x[i];
    for (i = 0; i < n; i++) {
        for (j = i; j < n; j++) {
```

```
            sum = cumarr[j] - cumarr[i-1];
            if (sum > maxsofar)
                maxsofar = sum;
        }
    }
    return maxsofar;
}

/* MS VC++ has a max macro, and therefore a perf bug */

#ifdef max
#undef max
#endif


#define maxmac(a, b) ((a) > (b) ? (a) : (b) )

float maxfun(float a, float b)
{   return a > b ? a : b;
}

#define max(a, b) maxfun(a, b)

float recmax(int l, int u)
{   int i, m;
    float lmax, rmax, sum;
    if (l > u)  /* zero elements */
                return 0;
    if (l == u)  /* one element */
                return max(0, x[l]);
    m = (l+u) / 2;
        /* find max crossing to left */
    lmax = sum = 0;
    for (i = m; i >= l; i--) {
                sum += x[i];
                if (sum > lmax)
                        lmax = sum;
    }
        /* find max crossing to right */
    rmax = sum = 0;
    for (i = m+1; i <= u; i++) {
                sum += x[i];
                if (sum > rmax)
                        rmax = sum;
    }
    return max(lmax + rmax,
                max(recmax(l, m), recmax(m+1, u)));
}

float alg3()
{   return recmax(0, n-1);
}

float alg4()
{   int i;
    float maxsofar = 0, maxendinghere = 0;
    for (i = 0; i < n; i++) {
        maxendinghere += x[i];
        if (maxendinghere < 0)
            maxendinghere = 0;
        if (maxsofar < maxendinghere)
            maxsofar = maxendinghere;
```

```
    }
    return maxsofar;
}

float alg4b()
{   int i;
    float maxsofar = 0, maxendinghere = 0;
    for (i = 0; i < n; i++) {
        maxendinghere += x[i];
        maxendinghere = maxmac(maxendinghere, 0);
        maxsofar = maxmac(maxsofar, maxendinghere);
    }
    return maxsofar;
}

float alg4c()
{   int i;
    float maxsofar = 0, maxendinghere = 0;
    for (i = 0; i < n; i++) {
        maxendinghere += x[i];
        maxendinghere = maxfun(maxendinghere, 0);
        maxsofar = maxfun(maxsofar, maxendinghere);
    }
    return maxsofar;
}

int main()
{   int algnum, start, clicks;
    float thisans;

    while (scanf("%d %d", &algnum, &n) != EOF) {
        sprinkle();
        start = clock();
        thisans = -1;
        switch (algnum) {
                        case 1:  thisans = alg1();  break;
                        case 2:  thisans = alg2();  break;
                        case 22: thisans = alg2b(); break;
                        case 3:  thisans = alg3();  break;
                        case 4:  thisans = alg4();  break;
                        case 42: thisans = alg4b(); break;
                        case 43: thisans = alg4c(); break;
                        default: break;
                }
        clicks = clock()-start;
        printf("%d\t%d\t%f\t%d\t%f\n", algnum, n, thisans,
            clicks, clicks / (float) CLOCKS_PER_SEC);
        if  (alg4() != thisans)
            printf(" maxsum error: mismatch with alg4: %f\n", alg4());
    }
    return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* genbins.c -- generate random numbers with bins */

/* If NODESIZE is 8, this program uses the special-case malloc.
   Change NODESIZE to 0 to use the system malloc.
 */

#include <stdio.h>
#include <stdlib.h>

#define NODESIZE 8
#define NODEGROUP 1000
int nodesleft = 0;
char *freenode;

void *pmalloc(int size)
{       void *p;
        if (size != NODESIZE)
                return malloc(size);
        if (nodesleft == 0) {
                freenode = malloc(NODEGROUP*NODESIZE);
                nodesleft = NODEGROUP;
        }
        nodesleft--;
        p = (void *) freenode;
        freenode += NODESIZE;
        return p;
}

struct node {
        int val;
        struct node *next;
};

struct node **bin, *sentinel;
int bins, bincnt, maxval;

void initbins(int maxelms, int pmaxval)
{       int i;
        bins = maxelms;
        maxval = pmaxval;
        bin = pmalloc(bins*sizeof(struct node *));
        sentinel = pmalloc(sizeof(struct node));
        sentinel->val = maxval;
        for (i = 0; i < bins; i++)
                bin[i] = sentinel;
        bincnt = 0;
}

struct node *rinsert(struct node *p, int t)
{       if (p->val < t) {
                p->next = rinsert(p->next, t);
        } else if (p->val > t) {
                struct node *q = pmalloc(sizeof(struct node));
                q->val = t;
                q->next = p;
                p = q;
                bincnt++;
        }
        return p;
```

```
}

void insert(int t)
{       int i;
        i = t / (1 + maxval/bins);
        i = t / (1 + maxval/bins);
        bin[i] = rinsert(bin[i], t);
}

void report()
{       int i, j = 0;
        struct node *p;
        for (i = 0; i < bins; i++)
                for (p = bin[i]; p != sentinel; p = p->next)
                        /* printf("%d\n", p->val) */;
                        /* Uncomment for testing, comment for profiling */
}

int bigrand()
{       return RAND_MAX*rand() + rand();
}

int main(int argc, char *argv[])
{       int m = atoi(argv[1]);
        int n = atoi(argv[2]);
        initbins(m, n);
        while (bincnt < m) {
                insert(bigrand() % n);
        }
        report();
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* macfun.c -- time macro and function implementations of max
 * Input: a sequence of (alg num, n) pairs.
 * Output: for each test, (alg num, n, ans, ticks, secs)
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXN 1000000
float x[MAXN];


/* arrmax1 -- max is a macro */

#define max1(a, b) ((a) > (b) ? (a) : (b))

float arrmax1(int n)
{       if (n == 1)
                return x[0];
        else
                return max1(x[n-1], arrmax1(n-1));
}


/* arrmax2 -- max is a function */

float max2(float a, float b)
{       return a > b ? a : b;
}

float arrmax2(int n)
{       if (n == 1)
                return x[0];
        else
                return max2(x[n-1], arrmax2(n-1));
}


/* arrmax3 -- MS VC++ stdlib defines max as a macro */

#ifndef max
#define max(a, b) max2(a, b)
#endif

float arrmax3(int n)
{       if (n == 1)
                return x[0];
        else
                return max(x[n-1], arrmax3(n-1));
}


int main()
{   int algnum, i, n, start, clicks;
    float thisans;

        for (i = 0; i < MAXN; i++)
                x[i] = MAXN-i;
        while (scanf("%d %d", &algnum, &n) != EOF) {
                start = clock();
```

```
                thisans = -1;
                switch (algnum) {
                        case 1: thisans = arrmax1(n); break;
                        case 2: thisans = arrmax2(n); break;
                        case 3: thisans = arrmax3(n); break;
                        default: break;
                }
                clicks = clock()-start;
                printf("%d\t%d\t%g\t%d\t%g\n", algnum, n, thisans,
                        clicks, clicks / (float) CLOCKS_PER_SEC);
    }
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* sort.cpp -- test and time sorting algorithms
        Input lines:  algnum n mod
        Output lines: algnum n mod clicks nanosecs_per_elem
   This is predominantly a C program; the only use of C++
   sort function immediately below the include line.
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// To change from C++ back to C, remove the following two lines
// and the call to sort in main
#include <algorithm>
using namespace std;

/* Data and supporting functions */

#define MAXN 10000000

typedef int DType;

DType realx[MAXN];
int *x = realx; /* allow x to shift for heaps */
int n;

void swap(int i, int j)
{       DType t = x[i];
        x[i] = x[j];
        x[j] = t;
}

int randint(int l, int u)
{       return l + (RAND_MAX*rand() + rand()) % (u-l+1);
}

/* LIBRARY QSORT */

int intcomp(int *x, int *y)
{       return *x - *y;
}

/* INSERTION SORTS */

/* Simplest insertion sort */
void isort1()
{       int i, j;
        for (i = 1; i < n; i++)
                for (j = i; j > 0 && x[j-1] > x[j]; j--)
                        swap(j-1, j);
}

/* Write swap function inline */
void isort2()
{       int i, j;
        DType t;
        for (i = 1; i < n; i++)
                for (j = i; j > 0 && x[j-1] > x[j]; j--) {
                        t = x[j];
                        x[j] = x[j-1];
```

```
                        x[j-1] = t;
                }
}

/* Move assignments to and from t out of loop */
void isort3()
{       int i, j;
        DType t;
        for (i = 1; i < n; i++) {
                t = x[i];
                for (j = i; j > 0 && x[j-1] > t; j--)
                        x[j] = x[j-1];
                x[j] = t;
        }
}

/* QUICKSORTS */

/* Simplest version, Lomuto partitioning */
void qsort1(int l, int u)
{       int i, m;
        if (l >= u)
                return;
        m = l;
        for (i = l+1; i <= u; i++)
                if (x[i] < x[l])
                        swap(++m, i);
        swap(l, m);
        qsort1(l, m-1);
        qsort1(m+1, u);
}

/* Sedgewick's version of Lomuto, with sentinel */
void qsort2(int l, int u)
{       int i, m;
        if (l >= u)
                return;
        m = i = u+1;
        do {
                do i--; while (x[i] < x[l]);
                swap(--m, i);
        } while (i > l);
        qsort2(l, m-1);
        qsort2(m+1, u);
}

/* Two-way partitioning */
void qsort3(int l, int u)
{       int i, j;
        DType t;
        if (l >= u)
                return;
        t = x[l];
        i = l;
        j = u+1;
        for (;;) {
                do i++; while (i <= u && x[i] < t);
                do j--; while (x[j] > t);
                if (i > j)
                        break;
                swap(i, j);
```

```
        }
        swap(l, j);
        qsort3(l, j-1);
        qsort3(j+1, u);
}

/* qsort3 + randomization + isort small subarrays + swap inline */
int cutoff = 50;
void qsort4(int l, int u)
{       int i, j;
        DType t, temp;
        if (u - l < cutoff)
                return;
        swap(l, randint(l, u));
        t = x[l];
        i = l;
        j = u+1;
        for (;;) {
                do i++; while (i <= u && x[i] < t);
                do j--; while (x[j] > t);
                if (i > j)
                        break;
                temp = x[i]; x[i] = x[j]; x[j] = temp;
        }
        swap(l, j);
        qsort4(l, j-1);
        qsort4(j+1, u);
}

/* selection */

void select1(int l, int u, int k)
{       int i, j;
        DType t, temp;
        if (l >= u)
                return;
        swap(l, randint(l, u));
        t = x[l];
        i = l;
        j = u+1;
        for (;;) {
                do i++; while (i <= u && x[i] < t);
                do j--; while (x[j] > t);
                if (i > j)
                        break;
                temp = x[i]; x[i] = x[j]; x[j] = temp;
        }
        swap(l, j);
        if (j < k)
                select1(j+1, u, k);
        else if (j > k)
                select1(l, j-1, k);
}

/* HEAP SORTS */

void siftup(int u)
{       int i, p;
        i = u;
        for (;;) {
                if (i == 1)
```

```
                        break;
                p = i / 2;
                if (x[p] >= x[i])
                        break;
                swap(p, i);
                i = p;
        }
}

void siftdown1(int l, int u)
{       int i, c;
        i = l;
        for (;;) {
                c = 2*i;
                if (c > u)
                        break;
                if (c+1 <= u && x[c+1] > x[c])
                        c++;
                if (x[i] > x[c])
                        break;
                swap(i, c);
                i = c;
        }
}

void siftdown1b(int l, int u) /* More C-ish version of 1 */
{       int i, c;
        for (i = l; (c = 2*i) <= u; i = c) {
                if (c+1 <= u && x[c+1] > x[c])
                        c++;
                if (x[i] > x[c])
                        break;
                swap(i, c);
        }
}

void hsort1()
{       int i;
        x--;
        for (i = 2; i <= n; i++)
                siftup(i);
        for (i = n; i >= 2; i--) {
                swap(1, i);
                siftdown1(1, i-1);
        }
        x++;
}

void hsort2()
{       int i;
        x--;
        for (i = n/2; i >= 1; i--)
                siftdown1(i, n);
        for (i = n; i >= 2; i--) {
                swap(1, i);
                siftdown1(1, i-1);
        }
        x++;
}

void siftdown3(int l, int u) /* push to bottom, then back up */
```

```
{       int i, c, p;
        i = l;
        for (;;) {
                c = 2*i;
                if (c > u)
                        break;
                if (c+1 <= u && x[c+1] > x[c])
                        c++;
                swap(i, c);
                i = c;
        }
        for (;;) {
                p = i/2;
                if (p < l)
                        break;
                if (x[p] > x[i])
                        break;
                swap(p, i);
                i = p;
        }
}

void hsort3()
{       int i;
        x--;
        for (i = n/2; i >= 1; i--)
                siftdown3(i, n);
        for (i = n; i >= 2; i--) {
                swap(1, i);
                siftdown3(1, i-1);
        }
        x++;
}

void siftdown4(int l, int u) /* replace swap with assignments */
{       int i, c, p;
        DType t;
        t = x[l];
        i = l;
        for (;;) {
                c = 2*i;
                if (c > u)
                        break;
                if (c+1 <= u && x[c+1] > x[c])
                        c++;
                x[i] = x[c];
                i = c;
        }
        x[i] = t;
        for (;;) {
                p = i/2;
                if (p < l)
                        break;
                if (x[p] > x[i])
                        break;
                swap(p, i);
                i = p;
        }
}

void hsort4()
```

```
{       int i;
        x--;
        for (i = n/2; i >= 1; i--)
                siftdown4(i, n);
        for (i = n; i >= 2; i--) {
                swap(1, i);
                siftdown4(1, i-1);
        }
        x++;
}


/* Other Sorts -- Exercises in Column 11 */

void selsort()  /* Selection sort */
{       int i, j;
        for (i = 0; i < n-1; i++)
                for (j = i; j < n; j++)
                        if (x[j] < x[i])
                                swap(i, j);
}

void shellsort()
{       int i, j, h;
        for (h = 1; h < n; h = 3*h + 1)
                ;
        for (;;) {
                h /= 3;
                if (h < 1) break;
                for (i = h; i < n; i++) {
                        for (j = i; j >= h; j -= h) {
                                if (x[j-h] < x[j]) break;
                                swap(j-h, j);
                        }
                }
        }
}



/* SCAFFOLDING */

/* Timing */

void timedriver()
{       int i, k, algnum, mod, start, clicks;
        while (scanf("%d %d %d", &algnum, &n, &mod) != EOF) {
                if (mod <= 0)
                        mod = 10*n;
                for (i = 0; i < n; i++)
                        x[i] = randint(0, mod-1);
                k = n/2;
                start = clock();
                switch (algnum) {
                case 11: qsort(x, n, sizeof(int), (int (__cdecl *)(const void *,const void
*)) intcomp); break;
                case 12: sort(x, x+n); break;
                case 21: isort1(); break;
                case 22: isort2(); break;
                case 23: isort3(); break;
                case 31: qsort1(0, n-1); break;
                case 32: qsort2(0, n-1); break;
```

```
                case 33: qsort3(0, n-1); break;
                case 34: qsort4(0, n-1); isort3(); break;
                case 41: select1(0, n-1, k); break;
                case 51: hsort1(); break;
                case 52: hsort2(); break;
                case 53: hsort3(); break;
                case 54: hsort4(); break;
                case 61: selsort(); break;
                case 62: shellsort(); break;
                }
                clicks = clock() - start;
                if (algnum == 41) { /* Test selection */
                        for (i = 0; i < k; i++)
                                if (x[i] > x[k])
                                        printf("  SELECT BUG i=%d\n", i);
                        for (i = k+1; i < n; i++)
                                if (x[i] < x[k])
                                        printf("  SELECT BUG i=%d\n", i);
                } else { /* Test sort */
                        for (i = 0; i < n-1; i++)
                                if (x[i] > x[i+1])
                                        printf("  SORT BUG i=%d\n", i);
                }
                printf("%d\t%d\t%d\t%d\t%g\n",
                        algnum, n, mod, clicks,
                        1e9*clicks/((float) CLOCKS_PER_SEC*n));
        }
}

/* Main */

int main()
{       timedriver();
        return 0;
}
```

```
// Copyright (C) 1999 Lucent Technologies
// From 'Programming Pearls' by Jon Bentley

// SortAnim.java -- Animate sorting algorithms

import java.applet.*;
import java.awt.*;
import java.util.Date;

public class SortAnim extends Applet {
        // Screen Elements
        private TextField n_text;
        private Choice dist_choices;
        private Choice alg_choices;
        private Button run_button;
        private Label msg_label;
        private Color draw_color = Color.black;
        private Color back_color = Color.white;

// SORTING DATA AND ALGS

        static private final int MAXN = 10000;
        static private int n=100;
        static private float a[] = new float[MAXN];

        // Sorting: Generate Inputs
        static private final int GEN_RAND = 0;
        static private final int GEN_ASCEND = 1;
        static private final int GEN_DESCEND = 2;
        static private int gen_num = GEN_RAND;

        private void genarray()
        {       for (int i = 0; i < n; i++) {
                        switch(gen_num) {
                        case GEN_RAND: a[i] = (float) Math.random(); break;
                        case GEN_ASCEND: a[i] = ((float) i)/n; break;
                        case GEN_DESCEND: a[i] = (float) (1.0 - ((float) i)/n); break;
                        }
                }
        }

        // Sorting: Supporting Algs
        private void baseswap(int i, int j)
        {       float t = a[i];
                a[i] = a[j];
                a[j] = t;
        }

        // Sorting: Animation Support
        static private final int MINX = 20, MAXX = 580;
        static private final int MINY = 50, MAXY = 380;
        static private float factorx, factory;
        static private boolean wantanim = true;

        private void initdisplay()
        {       Graphics g = this.getGraphics();
                Rectangle r = this.bounds();
                g.setColor(back_color);
                g.fillRect(r.x, r.y, r.width, r.height);
                factorx = ((float) MAXX-MINX) / n;
                factory = ((float) MAXY-MINY);
        }
```

```
private void draw(int i, Color c)
{       Graphics g = this.getGraphics();  // BETTER WAY?
        int d = 4;
        int px = (int) (MINX + factorx*i);
        int py = MAXY - (int)(factory*a[i]);
        g.setColor(c);
        g.drawOval(px, py, d, d);
}

private void swap(int i, int j)
{       if (wantanim) {
                draw(i, back_color);
                draw(j, back_color);
        }
        baseswap(i, j);
        if (wantanim) {
                draw(i, draw_color);
                draw(j, draw_color);
        }
}

// Sorting Algs
private void isort()
{       for (int i = 1; i < n; i++)
                for (int j = i; j > 0 && a[j-1] > a[j]; j--)
                        swap(j-1, j);
}

private void ssort()
{       for (int i = 0; i < n-1; i++)
                for (int j = i; j < n; j++)
                        if (a[j] < a[i])
                                swap(i, j);
}

private void shellsort()
{       int i, j, h;
        for (h = 1; h < n; h = 3*h + 1)
                ;
        for (;;) {
                h /= 3;
                if (h < 1) break;
                for (i = h; i < n; i++) {
                        for (j = i; j >= h; j -= h) {
                                if (a[j-h] < a[j]) break;
                                swap(j-h, j);
                        }
                }
        }
}

private void siftdown(int l, int u)
{       int i, c;
        i = l;
        for (;;) {
                c = 2*i;
                if (c > u)
                        break;
                if (c+1 <= u && a[c+1] > a[c])
                        c++;
                if (a[i] >= a[c])
```

```
                        break;
                swap(i, c);
                i = c;
        }
}

private void heapsort() // BEWARE!!! Sorts x[1..n-1]
{       int i;
        for (i = n/2; i > 0; i--)
                siftdown(i, n-1);
        for (i = n-1; i >= 2; i--) {
                swap(1, i);
                siftdown(1, i-1);
        }
}

private void qsort(int l, int u)
{       if (l >= u)
                return;
        int m = l;
        for (int i = l+1; i <= u; i++)
                if (a[i] < a[l])
                        swap(++m, i);
        swap(l, m);
        qsort(l, m-1);
        qsort(m+1, u);
}

void qsort2(int l, int u)
{       if (l >= u)
                return;
        int i = l;
        int j = u+1;
        for (;;) {
                do i++; while (i <= u && a[i] < a[l]);
                do j--; while (a[j] > a[l]);
                if (i > j)
                        break;
                swap(i, j);
        }
        swap(l, j);
        qsort2(l, j-1);
        qsort2(j+1, u);
}

// Drive Sort
static private final int ALG_ISORT = 0;
static private final int ALG_SELSORT = 1;
static private final int ALG_SHELLSORT = 2;
static private final int ALG_HSORT = 3;
static private final int ALG_QSORT = 4;
static private final int ALG_QSORT2 = 5;
static private int alg_num = ALG_ISORT;

private void dosort()
{       switch(alg_num) {
        case ALG_ISORT:     isort(); break;
        case ALG_SELSORT:   ssort(); break;
        case ALG_SHELLSORT: shellsort(); break;
        case ALG_HSORT:     heapsort(); break;
        case ALG_QSORT:     qsort(0, n-1); break;
```

```
                case ALG_QSORT2:    qsort2(0, n-1); break;
                }
        }

        private void runanim()
        {       n = Integer.parseInt(n_text.getText());
                if (n < 1 || n > MAXN) {
                        n = 50;
                        n_text.setText("" + n);
                }
                initdisplay();
                msg_label.setText("Running");
                genarray();
                for (int i = 0; i < n; i++)
                        draw(i, draw_color);
                Date timer = new Date();
                long start = timer.getTime();
                dosort();
                timer = new Date();
                long msecs = timer.getTime() - start;
                msg_label.setText("Msecs: " + msecs);
                if (! wantanim) // Draw results over input
                        for (int i = 0; i < n; i++)
                                draw(i, draw_color);
        }

// GUI FUNCTIONS
        public void init() {
                this.setBackground(back_color);

                // TextField for n (problem size)
                n_text = new TextField(5);
                this.add(new Label("n:"));
                this.add(n_text);
                n_text.setText("" + n);

                // Choice of Starting distributions
                dist_choices = new Choice();
                dist_choices.addItem("Random");
                dist_choices.addItem("Ascending");
                dist_choices.addItem("Descending");
                this.add(new Label("Input:"));
                this.add(dist_choices);

                // Choice of Sort Algorithms
                alg_choices = new Choice();
                alg_choices.addItem("Insertion Sort");
                alg_choices.addItem("Selection Sort");
                alg_choices.addItem("Shell Sort");
                alg_choices.addItem("Heap Sort");
                alg_choices.addItem("Quicksort");
                alg_choices.addItem("2-way Quicksort");
                this.add(new Label("Algorithm:"));
                this.add(alg_choices);

                // Run Button
                run_button = new Button("Run");
                this.add(run_button);

                // Message Label
                msg_label = new Label("                    ");
                this.add(msg_label);
```

```
        }

        public boolean action(Event event, Object arg) {
                if (event.target == dist_choices) {
                        if (arg.equals("Random")) gen_num = GEN_RAND;
                        else if (arg.equals("Ascending")) gen_num = GEN_ASCEND;
                        else if (arg.equals("Descending")) gen_num = GEN_DESCEND;
                        return true;
                } else if (event.target == alg_choices) {
                        if (arg.equals("Insertion Sort")) alg_num = ALG_ISORT;
                        else if (arg.equals("Selection Sort")) alg_num = ALG_SELSORT;
                        else if (arg.equals("Shell Sort")) alg_num = ALG_SHELLSORT;
                        else if (arg.equals("Heap Sort")) alg_num = ALG_HSORT;
                        else if (arg.equals("Quicksort")) alg_num = ALG_QSORT;
                        else if (arg.equals("2-way Quicksort")) alg_num = ALG_QSORT2;
                        return true;
                } else if (event.target == run_button) {
                        runanim();
                        return true;
                } else
                        return super.action(event, arg);
        }
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* sortedrand.cpp -- output m sorted random ints in U[0,n) */

#include <iostream>
#include <set>
#include <algorithm>
using namespace std;

int bigrand()
{       return RAND_MAX*rand() + rand();
}

int randint(int l, int u)
{       return l + bigrand() % (u-l+1);
}

void genknuth(int m, int n)
{       for (int i = 0; i < n; i++)
                /* select m of remaining n-i */
                if ((bigrand() % (n-i)) < m) {
                        cout << i << "\n";
                        m--;
                }
}

void gensets(int m, int n)
{       set<int> S;
        set<int>::iterator i;
        while (S.size() < m) {
                int t = bigrand() % n;
                S.insert(t);
        }
        for (i = S.begin(); i != S.end(); ++i)
                cout << *i << "\n";
}

void genshuf(int m, int n)
{       int i, j;
        int *x = new int[n];
        for (i = 0; i < n; i++)
                x[i] = i;
        for (i = 0; i < m; i++) {
                j = randint(i, n-1);
                int t = x[i]; x[i] = x[j]; x[j] = t;
        }
        sort(x, x+m);
        for (i = 0; i < m; i++)
                cout << x[i] << "\n";
}

void genfloyd(int m, int n)
{       set<int> S;
        set<int>::iterator i;
        for (int j = n-m; j < n; j++) {
                int t = bigrand() % (j+1);
                if (S.find(t) == S.end())
                        S.insert(t); // t not in S
                else
                        S.insert(j); // t in S
        }
```

```
        for (i = S.begin(); i != S.end(); ++i)
                cout << *i << "\n";
}

int main(int argc, char *argv[])
{       int m = atoi(argv[1]);
        int n = atoi(argv[2]);
        genknuth(m, n);
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* sets.cpp -- exercise set implementations on random numbers */

#include <iostream>
#include <set>
using namespace std;


class IntSetSTL {
private:
        set<int> S;
public:
        IntSetSTL(int maxelements, int maxval) { }
        int size() { return S.size(); }
        void insert(int t) { S.insert(t);}
        void report(int *v)
        {       int j = 0;
                set<int>::iterator i;
                for (i = S.begin(); i != S.end(); ++i)
                        v[j++] = *i;
        }
};


class IntSetBitVec {
private:
        enum { BITSPERWORD = 32, SHIFT = 5, MASK = 0x1F };
        int     n, hi, *x;
        void set(int i)  {        x[i>>SHIFT] |=  (1<<(i & MASK)); }
        void clr(int i)  {        x[i>>SHIFT] &= ~(1<<(i & MASK)); }
        int  test(int i) { return x[i>>SHIFT] &   (1<<(i & MASK)); }
public:
        IntSetBitVec(int maxelements, int maxval)
        {       hi = maxval;
                x = new int[1 + hi/BITSPERWORD];
                for (int i = 0; i < hi; i++)
                        clr(i);
                n = 0;
        }
        int size() { return n; }
        void insert(int t)
        {       if (test(t))
                        return;
                set(t);
                n++;
        }
        void report(int *v)
        {       int j=0;
                for (int i = 0; i < hi; i++)
                        if (test(i))
                                v[j++] = i;
        }
};


class IntSetArr {
private:
        int     n, *x;
public:
        IntSetArr(int maxelements, int maxval)
```

```
        {       x = new int[1 + maxelements];
                n=0;
                x[0] = maxval; /* sentinel at x[n] */
        }
        int size() { return n; }
        void insert(int t)
        {       int i, j;
                for (i = 0; x[i] < t; i++)
                        ;
                if (x[i] == t)
                        return;
                for (j = n; j >= i; j--)
                        x[j+1] = x[j];
                x[i] = t;
                n++;
        }
        void report(int *v)
        {       for (int i = 0; i < n; i++)
                        v[i] = x[i];
        }
};


class IntSetList {
private:
        int     n;
        struct node {
                int val;
                node *next;
                node(int i, node *p) { val = i; next = p; }
        };
        node *head, *sentinel;
        node *rinsert(node *p, int t)
        {       if (p->val < t) {
                        p->next = rinsert(p->next, t);
                } else if (p->val > t) {
                        p = new node(t, p);
                        n++;
                }
                return p;
        }
public:
        IntSetList(int maxelements, int maxval)
        {       sentinel = head = new node(maxval, 0);
                n = 0;
        }
        int size() { return n; }
        void insert(int t) { head = rinsert(head, t); }
        void insert2(int t)
        {       node *p;
                if (head->val == t)
                        return;
                if (head->val > t) {
                        head = new node(t, head);
                        n++;
                        return;
                }
                for (p = head; p->next->val < t; p = p->next)
                        ;
                if (p->next->val == t)
                        return;
```

```
                p->next = new node(t, p->next);
                n++;
        }
        void insert3(int t)
        {       node **p;
                for (p = &head; (*p)->val < t; p = &((*p)->next))
                        ;
                if ((*p)->val == t)
                        return;
                *p = new node(t, *p);
                n++;
        }
        void report(int *v)
        {       int j = 0;
                for (node *p = head; p != sentinel; p = p->next)
                        v[j++] = p->val;
        }
};

// Change from new per node to one new at init
// Factor of 2.5 on VC 5.0, 6% on SGI CC
class IntSetList2 {
private:
        int     n;
        struct node {
                int val;
                node *next;
        };
        node *head, *sentinel, *freenode;
public:
        IntSetList2(int maxelements, int maxval)
        {       sentinel = head = new node;
                sentinel->val = maxval;
                freenode = new node[maxelements];
                n = 0;
        }
        int size() { return n; }
        void insert(int t)
        {       node **p;
                for (p = &head; (*p)->val < t; p = &((*p)->next))
                        ;
                if ((*p)->val == t)
                        return;
                freenode->val = t;
                freenode->next = *p;
                *p = freenode++;
                n++;
        }
        void report(int *v)
        {       int j = 0;
                for (node *p = head; p != sentinel; p = p->next)
                        v[j++] = p->val;
        }
};


class IntSetBST {
private:
        int     n, *v, vn;
        struct node {
                int val;
```

```
                node *left, *right;
                node(int v) { val = v; left = right = 0; }
        };
        node *root;
        node *rinsert(node *p, int t)
        {       if (p == 0) {
                        p = new node(t);
                        n++;
                } else if (t < p->val) {
                        p->left = rinsert(p->left, t);
                } else if (t > p->val) {
                        p->right = rinsert(p->right, t);
                } // do nothing if p->val == t
                return p;
        }
        void traverse(node *p)
        {       if (p == 0)
                        return;
                traverse(p->left);
                v[vn++] = p->val;
                traverse(p->right);
        }
public:
        IntSetBST(int maxelements, int maxval) { root = 0; n = 0; }
        int size() { return n; }
        void insert(int t) { root = rinsert(root, t); }
        void report(int *x) { v = x; vn = 0; traverse(root); }
};


class IntSetBST2 {
private:
        int     n, *v, vn;
        struct node {
                int val;
                node *left, *right;
        };
        node *root, *freenode, *sentinel;
        node *rinsert(node *p, int t)
        {       if (p == sentinel) {
                        p = freenode++;
                        p->val = t;
                        p->left = p->right = sentinel;
                        n++;
                } else if (t < p->val) {
                        p->left = rinsert(p->left, t);
                } else if (t > p->val) {
                        p->right = rinsert(p->right, t);
                } // do nothing if p->val == t
                return p;
        }
        void traverse(node *p)
        {       if (p == sentinel)
                        return;
                traverse(p->left);
                v[vn++] = p->val;
                traverse(p->right);
        }
public:
        IntSetBST2(int maxelements, int maxval)
        {       root = sentinel = new node;  // 0 if using insert1
```

```
                n = 0;
                freenode = new node[maxelements];
        }
        int size() { return n; }
        void insert1(int t) { root = rinsert(root, t); }
        void insert(int t)
        {       sentinel->val = t;
                node **p = &root;
                while ((*p)->val != t)
                        if (t < (*p)->val)
                                p = &((*p)->left);
                        else
                                p = &((*p)->right);
                if (*p == sentinel) {
                        *p = freenode++;
                        (*p)->val = t;
                        (*p)->left = (*p)->right = sentinel;
                        n++;
                }
        }
        void report(int *x) { v = x; vn = 0; traverse(root); }
};


class IntSetBins {
private:
        int     n, bins, maxval;
        struct node {
                int val;
                node *next;
                node(int v, node *p) { val = v; next = p; }
        };
        node **bin, *sentinel;
        node *rinsert(node *p, int t)
        {       if (p->val < t) {
                        p->next = rinsert(p->next, t);
                } else if (p->val > t) {
                        p = new node(t, p);
                        n++;
                }
                return p;
        }
public:
        IntSetBins(int maxelements, int pmaxval)
        {       bins = maxelements;
                maxval = pmaxval;
                bin = new node*[bins];
                sentinel = new node(maxval, 0);
                for (int i = 0; i < bins; i++)
                        bin[i] = sentinel;
                n = 0;
        }
        int size() { return n; }
        void insert(int t)
        {       int i = t / (1 + maxval/bins);  // CHECK !
                bin[i] = rinsert(bin[i], t);
        }
        void report(int *v)
        {       int j = 0;
                for (int i = 0; i < bins; i++)
                        for (node *p = bin[i]; p != sentinel; p = p->next)
```

```
                                v[j++] = p->val;
        }
};


class IntSetBins2 {
private:
        int     n, bins, maxval;
        struct node {
                int val;
                node *next;
        };
        node **bin, *sentinel, *freenode;
        node *rinsert(node *p, int t)
        {       if (p->val < t) {
                        p->next = rinsert(p->next, t);
                } else if (p->val > t) {
                        freenode->val = t;
                        freenode->next = p;
                        p = freenode++;
                        n++;
                }
                return p;
        }
public:
        IntSetBins2(int maxelements, int pmaxval)
        {       bins = maxelements;
                maxval = pmaxval;
                freenode = new node[maxelements];
                bin = new node*[bins];
                sentinel = new node;
                sentinel->val = maxval;
                for (int i = 0; i < bins; i++)
                        bin[i] = sentinel;
                n = 0;
        }
        int size() { return n; }
        void insert1(int t)
        {       int i = t / (1 + maxval/bins);
                bin[i] = rinsert(bin[i], t);
        }
        void insert(int t)
        {       node **p;
                int i = t / (1 + maxval/bins);
                for (p = &bin[i]; (*p)->val < t; p = &((*p)->next))
                        ;
                if ((*p)->val == t)
                        return;
                freenode->val = t;
                freenode->next = *p;
                *p = freenode++;
                n++;
        }
        void report(int *v)
        {       int j = 0;
                for (int i = 0; i < bins; i++)
                        for (node *p = bin[i]; p != sentinel; p = p->next)
                                v[j++] = p->val;
        }
};
```

```
// Drivers for the set data structures

int bigrand()
{       return RAND_MAX*rand() + rand();
}

int randint(int l, int u)
{       return l + bigrand() % (u-l+1);
}

void gensets(int m, int maxval)
{       int *v = new int[m];
        IntSetList S(m, maxval);
        while (S.size() < m)
                S.insert(bigrand() % maxval);
        S.report(v);
//      for (int i = 0; i < m; i++)
        for (int i = 0; i < 2; i++)
                cout << v[i] << "\n";
}

void genfloyd(int m, int maxval)
{       int *v = new int[m];
        IntSetSTL S(m, maxval);
        for (int j = maxval-m; j < maxval; j++) {
                int t = bigrand() % (j+1);
                int oldsize = S.size();
                S.insert(t);
                if (S.size() == oldsize) // t already in S
                        S.insert(j);
        }
        S.report(v);
        for (int i = 0; i < m; i++)
                cout << v[i] << "\n";
}

void memaccesstest(int m, int n)
{       IntSetList S(m, n);     // change among Arr, List and List2
        for (int i = 0; i < m; i++)
                S.insert(i);
}

void overheadonly(int m, int n)
{       int i, *v = new int[m];
        for (i = 0; i < m; i++)
                v[i] = bigrand() % n;
        for (i = 0; i < m; i++)
                cout << v[i] << "\n";
}

int main(int argc, char *argv[])
{       int m = atoi(argv[1]);
        int maxval = atoi(argv[2]);
        gensets(m, maxval);
        // overheadonly(m, n);
        // memaccesstest(m, n);
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* priqueue.cpp -- priority queues (using heaps) */

#include <iostream>
using namespace std;

// define and implement priority queues

template<class T>
class priqueue {
private:
        int     n, maxsize;
        T       *x;
        void swap(int i, int j)
        {       T t = x[i]; x[i] = x[j]; x[j] = t; }
public:
        priqueue(int m)
        {       maxsize = m;
                x = new T[maxsize+1];
                n = 0;
        }
        void insert(T t)
        {       int i, p;
                x[++n] = t;
                for (i = n; i > 1 && x[p=i/2] > x[i]; i = p)
                        swap(p, i);
        }
        T extractmin()
        {       int i, c;
                T t = x[1];
                x[1] = x[n--];
                for (i = 1; (c=2*i) <= n; i = c) {
                        if (c+1<=n && x[c+1]<x[c])
                                c++;
                        if (x[i] <= x[c])
                                break;
                        swap(c, i);
                }
                return t;
        }
};

// sort with priority queues (heap sort is strictly better)

template<class T>
void pqsort(T v[], int n)
{       priqueue<T> pq(n);
        int i;
        for (i = 0; i < n; i++)
                pq.insert(v[i]);
        for (i = 0; i < n; i++)
                v[i] = pq.extractmin();
}

// main

int main()
{       const int       n = 10;
        int     i, v[n];
        if (0) { // Generate and sort
```

```
                for (i = 0; i < n; i++)
                        v[i] = n-i;
                pqsort(v, n);
                for (i = 0; i < n; i++)
                        cout << v[i] << "\n";
        } else { // Insert integers; extract with 0
                priqueue<int> pq(100);
                while (cin >> i)
                        if (i == 0)
                                cout << pq.extractmin() << "\n";
                        else
                                pq.insert(i);
        }
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* wordlist.cpp -- Sorted list of words (between white space) in file */

#include <iostream>
#include <set>
#include <string>
using namespace std;

int main()
{       set<string> S;
        string t;
        set<string>::iterator j;
        while (cin >> t)
                S.insert(t);
        for (j = S.begin(); j != S.end(); ++j)
                cout << *j << "\n";
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* wordfreq.cpp -- List all words in input file, with counts */

#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{       map<string, int> M;
        map<string, int>::iterator j;
        string t;
        while (cin >> t)
                M[t]++;
        for (j = M.begin(); j != M.end(); ++j)
                cout << j->first << " " << j->second << "\n";
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* wordfreq.c -- list of words in file, with counts */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node *nodeptr;
typedef struct node {
        char *word;
        int count;
        nodeptr next;
} node;

#define NHASH 29989
#define MULT 31
nodeptr bin[NHASH];

unsigned int hash(char *p)
{       unsigned int h = 0;
        for ( ; *p; p++)
                h = MULT * h + *p;
        return h % NHASH;
}

#define NODEGROUP 1000
int nodesleft = 0;
nodeptr freenode;

nodeptr nmalloc()
{       if (nodesleft == 0) {
                freenode = malloc(NODEGROUP*sizeof(node));
                nodesleft = NODEGROUP;
        }
        nodesleft--;
        return freenode++;
}

#define CHARGROUP 10000
int charsleft = 0;
char *freechar;

char *smalloc(int n)
{       if (charsleft < n) {
                freechar = malloc(n+CHARGROUP);
                charsleft = n+CHARGROUP;
        }
        charsleft -= n;
        freechar += n;
        return freechar - n;
}

void incword(char *s)
{       nodeptr p;
        int h = hash(s);
        for (p = bin[h]; p != NULL; p = p->next)
                if (strcmp(s, p->word) == 0) {
                        (p->count)++;
                        return;
                }
```

```
        p = nmalloc();
        p->count = 1;
        p->word = smalloc(strlen(s)+1);
        strcpy(p->word, s);
        p->next = bin[h];
        bin[h] = p;
}

int main()
{       int i;
        nodeptr p;
        char buf[100];
        for (i = 0; i < NHASH; i++)
                bin[i] = NULL;
        while (scanf("%s", buf) != EOF)
                incword(buf);
        for (i = 0; i < NHASH; i++)
                for (p = bin[i]; p != NULL; p = p->next)
                        printf("%s %d\n", p->word, p->count);
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* spacemod.cpp -- simple model for C++ space */

#include <iostream>
using namespace std;

#define MEASURE(T, text) {                                          \
        cout << text << "\t";                                       \
        cout << sizeof(T) << "\t";                                  \
        int lastp = 0;                                                      \
        for (int i = 0; i < 11; i++) {                  \
                T *p = new T;                                               \
                int thisp = (int) p;                                \
                if (lastp != 0)                                             \
                        cout << " " << thisp - lastp;   \
                lastp = thisp;                                              \
        }                                                                                       \
        cout << "\n";                                               \
}

// Must use macros; templates give funny answers

template <class T>
void measure(char *text)
{       cout << " measure: " << text << "\t";
        cout << sizeof(T) << "\n";
}

struct structc { char c; };
struct structic { int i; char c; };
struct structip { int i; structip *p; };
struct structdc { double d; char c; };
struct structcd { char c; double d; };
struct structcdc { char c1; double d; char c2; };
struct structiii { int i1; int i2; int i3; };
struct structiic { int i1; int i2; char c; };
struct structc12 { char c[12]; };
struct structc13 { char c[13]; };
struct structc28 { char c[28]; };
struct structc29 { char c[29]; };
struct structc44 { char c[44]; };
struct structc45 { char c[45]; };
struct structc60 { char c[60]; };
struct structc61 { char c[61]; };

int main()
{       cout << "Raw sizeof";
        cout << "\nsizeof(char)="     << sizeof(char);
        cout << "  sizeof(short)="    << sizeof(short);
        cout << "  sizeof(int)="      << sizeof(int);
        cout << "\nsizeof(float)="    << sizeof(float);
        cout << "  sizeof(struct *)=" << sizeof(structc *);
        cout << "  sizeof(long)="     << sizeof(long);
        cout << "\nsizeof(double)="   << sizeof(double);

        cout << "\n\nMEASURE macro\n";
        MEASURE(int, "int");
        MEASURE(structc, "structc");
        MEASURE(structic, "structic");
        MEASURE(structip, "structip");
```

```
        MEASURE(structdc, "structdc");
        MEASURE(structcd, "structcd");
        MEASURE(structcdc, "structcdc");
        MEASURE(structiii, "structiii");
        MEASURE(structiic, "structiic");
        MEASURE(structc12, "structc12");
        MEASURE(structc13, "structc13");
        MEASURE(structc28, "structc28");
        MEASURE(structc29, "structc29");
        MEASURE(structc44, "structc44");
        MEASURE(structc45, "structc45");
        MEASURE(structc60, "structc60");
        MEASURE(structc61, "structc61");


        cout << "\nmeasure template (strange results)\n";
        // Uncomment below lines to see answers change
        measure<int>("int");
//      measure<structc>("structc");
//      measure<structic>("structic");
        return 0;
}
```

```
/* Copyright (C) 1999 Lucent Technologies */
/* From 'Programming Pearls' by Jon Bentley */

/* timemod.c -- Produce table of C run time costs */

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

#define MAXN 100000
int x[MAXN];
int startn = 5000;
int n;

/* FUNCTIONS TO BE TIMED */

int intcmp(int *i, int *j)
{   return *i - *j; }

#define swapmac(i, j) { t = x[i]; x[i] = x[j]; x[j] = t; }

void swapfunc(int i, int j)
 {  int t = x[i];
        x[i] = x[j];
        x[j] = t;
 }

#define maxmac(a, b) ((a) > (b) ? (a) : (b))

int maxfunc(int a, int b)
{       return a > b ? a : b; }


/* WORKHORSE */

#define T(s) printf("%s (n=%d)\n", s, n);
#define TRIALS 5
#define M(op)                                                   \
        printf(" %-22s", #op);                          \
        k = 0;                                                                  \
        timesum = 0;                                            \
        for (ex = 0; ex < TRIALS; ex++) {       \
                start = clock();                                \
                for (i = 1; i <= n; i++) {              \
                        fi = (float) i;                         \
                        for (j = 1; j <= n; j++) {      \
                                op;                                             \
                        }                                                       \
                }                                                       \
                t = clock()-start;                              \
                printf("%6d", t);                               \
                timesum += t;                                   \
        }                                                               \
        nans = 1e9 * timesum / ((double)        \
                n*n * TRIALS * CLOCKS_PER_SEC); \
        printf("%8.0f\n", nans);


int main()
{   int i, j, k;
        float fi, fj, fk;
        int t, ex, timesum, start, globalstart;
```

```
        double nans;
        globalstart = clock();
        for (i = 0; i < n; i++)
                x[i] = rand();
        n = startn;
        printf("C Time Cost Model, n=%d\n", n);
        T("Integer Arithmetic");
        M({});
        M(k++);
        M(k = i + j);
        M(k = i - j);
        M(k = i * j);
        M(k = i / j);
        M(k = i % j);
        M(k = i & j);
        M(k = i | j);
        T("Floating Point Arithmetic");
        M(fj=j;);
        M(fj=j; fk = fi + fj;);
        M(fj=j; fk = fi - fj;);
        M(fj=j; fk = fi * fj;);
        M(fj=j; fk = fi / fj;);
        T("Array Operations");
        M(k = i + j);
        M(k = x[i] + j);
        M(k = i + x[j]);
        M(k = x[i] + x[j]);
        T("Comparisons");
        M(if (i < j) k++);
        M(if (x[i] < x[j]) k++);
        T("Array Comparisons and Swaps");
        M(k = (x[i]<x[k]) ? -1:1);
        M(k = intcmp(x+i, x+j));
        M(swapmac(i, j));
        M(swapfunc(i, j));
        T("Max Function, Macro and Inline");
        M(k = (i > j) ? i : j);
        M(k = maxmac(i, j));
        M(k = maxfunc(i, j));
        n = startn / 5;
        T("Math Functions");
        M(fk = j+fi;);
        M(k = rand(););
        M(fk = sqrt(j+fi));
        M(fk = sin(j+fi));
        M(fk = sinh(j+fi));
        M(fk = asin(j+fi));
        M(fk = cos(j+fi));
        M(fk = tan(j+fi));
        n = startn / 10;
        T("Memory Allocation");
        M(free(malloc(16)););
        M(free(malloc(100)););
        M(free(malloc(2000)););

        /* Possible additions: input, output, malloc */
        printf("  Secs: %4.2f\n",
                ((double) clock()-globalstart)
                / CLOCKS_PER_SEC);
        return 0;
}
```