Counting Sort

Rachel Xu
03/31/2016

Comparison Sorts

- Ex: insertion sort, merge sort...
- All input items are black boxes
- The sorted order is based only on comparisons
- Worse-case number of comparisons: Ω (nlgn)

Counting Sort

- Linear-time sort
- Use actual values of input elements for indexing
- Input: an array of n integers in range(0, k)
- Need to create:

  An array of length k+1 to store the counts of each integers

  An array of length n to store the output

Steps of the Algorithm

Input Array: A[1 . . . n]

1. Use an array of length k to store the counts for all unique integers in A
2. Modify the count array such that the element at each index stores the sum of previous counts
3. Iterate over A and output each integer to its corresponding position in the output array

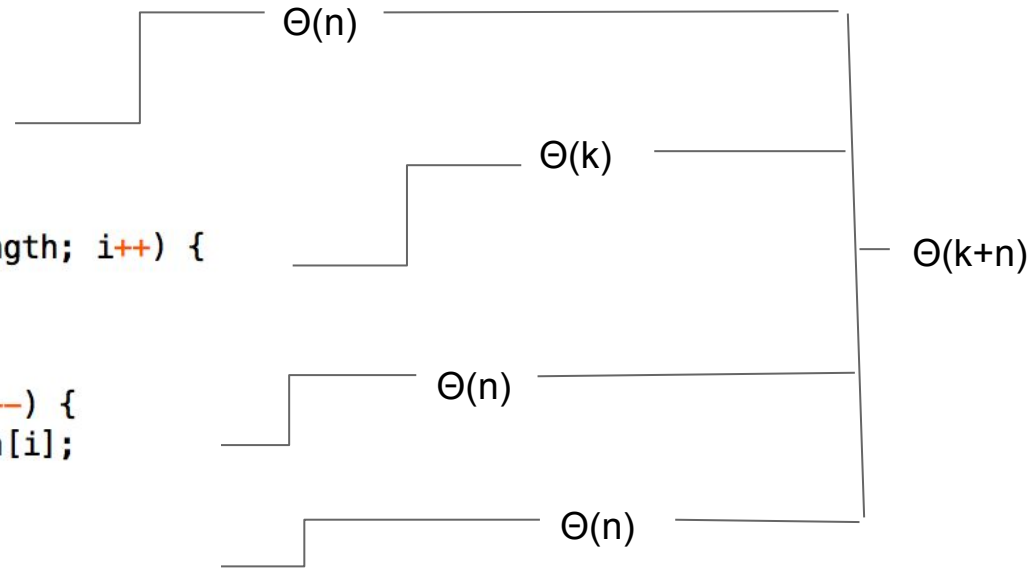
Java Code

```
public static void sort(int[] a, int k) {
    int N = a.length;
    int[] counts = new int[k];
    int[] output = new int[N];

    for (int i = 0; i < N; i++) {
        counts[a[i]]++;
    }

    for (int i = 1; i < counts.length; i++) {
        counts[i] += counts[i-1];
    }

    for (int i = N - 1; i >= 0; i--) {
        output[counts[a[i]]-1] = a[i];
        counts[a[i]]--;
    }
    for (int i = 0; i < N; i++) {
        a[i] = output[i];
    }
}
```



Time Complexity

1. Storing counts: $\Theta(n)$
2. Modifying count array: $\Theta(k)$
3. Output integers to output array: $\Theta(n)$

Overall: $\Theta(k + n)$

In practice: $k = O(n)$, running times becomes $\Theta(n)$

A Couple of Remarks

- Counting sort is stable: if two elements A[i] and A[j] have the same key value and i < j, A[i] will appear before A[j] in the output
  - Ex: input [1, 2, 3a, 4, 5, 3b] gives output [1, 2, 3a, 3b, 4, 5]
- If instead of going from N - 1 to 0 (as below), i goes from 0 to N - 1, the sort is not stable

```
for (int i = N - 1; i >= 0; i--) {
    output[counts[a[i]]-1] = a[i];
    counts[a[i]]--;
}
```

- Counting sort can be very efficient if k is much smaller than N

THANKS =D