

Algorithms Homework 2

Liam Dillingham

October 8, 2018

1 Question 8.2-4

Describe an algorithm that, given n integers in the range 0 to k , preprocesses its input and then answers any query about how many of the n integers fall into a range $[a..b]$ in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time

Suppose we have a set of integers, S , where the $\min(S) = 0$, and $\max(S) = k$. We want an algorithm that can tell how many integers, m fall in the range $[a, b]$ such that $0 \leq a \leq b \leq k$ in time $\mathcal{O}(1)$.

Let us observe COUNTING-SORT. First, we initialize an array of size k such that every element is equal to 0. Then we populate the array with the frequency in which each element appears. Then, for any element $j \in [0, k]$, we have another loop to sum the number of elements less than or equal to j with the frequency of j .

Note that since an array is already ordered by its own index. Since this is true, then we can count the number of elements m that fall within the range $[a, b]$ by simply iterating across the "counting" array: $m += (C[b] \text{ downto } C[a])$. Note that the difference between two constants is also a constant.

The counting step (building the frequency table) takes $\Theta(n)$ because we must loop over the entire input array, and the frequency step takes $\Theta(k)$ because we must loop over the size of the counting array. Thus the preprocessing step will take $\Theta(n + k)$, and the query step will take $\Theta(b - a)$, or $\mathcal{O}(1)$.

2 Question 8.3-4

Show how to sort n integers in the range 0 to $n^3 - 1$ in $\mathcal{O}(n)$ time.

Note: According to lemma 8.3: "Given n d -digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $\Theta(d(n + k))$ time if the stable sort it uses takes $\Theta(n + k)$ time"

if we distribute the d , we get $\Theta(dn + dk)$.

3 Question 8.4-2

Explain why the worst-case running time for bucket sort is $\Theta(n^2)$. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time $\mathcal{O}(n \lg n)$?
