Algorithms Homework- week 10

Massimo Hong
Student id: 2020280082

November 22, 2021

2-1

2-1.a

Input size $= k$, worst-case time $= \Theta(k^2)$.
If we need to sort the $\frac{n}{k}$ sublists, each of sike k, we will have:
$\frac{n}{k}\Theta(k^2) = \Theta(nk)$

2-1.b

We can merge 2 sublists (all sublists have size k) at a time until we get the fully sorted list of n.
Because we are merging two sublists at a time and we have a total of $\frac{n}{k}$ sublists, we have to perform: $log(\frac{n}{k})$ steps.
During each step we have to compare all the n elements. So the worst-case time will be $\Theta(nlog(\frac{n}{k}))$.
We can also see it as a tree structure. We have $\frac{n}{k}$ sublists, so the height of the tree will be $log(\frac{n}{k})$. In each level the merging costs $\Theta(n)$ (because we need to compare all n elements). So in total we have: $\Theta(nlog(\frac{n}{k}))$.

2-1.c

Worst-case time of the modified algorithm is: $\Theta(nk + nlog(\frac{n}{k}))$.
Worst-case time of standard merge sort is: $\Theta(nlogn)$.
We need to find k such that: $\Theta(nk + nlogn - nlogk)) = \Theta(nlogn)$
In order for this equality to hold, k cannot grow faster than log(n) asymptotically, otherwise nk will run at a worse time compared to $\Theta(nlogn)$. We need to have: $k \leq \Theta(logn)$.
Let's assume $k = \Theta(logn)$:

$$\Theta(nk + nlog(\frac{n}{k}) = \Theta(nlogn + nlogn - nlog(logn))$$

$$= \Theta(2nlogn - nlog(logn))$$

$log(logn)$ is very small compared to log(n), and thus we can ignore it.

$$= \Theta(nlogn)$$

2-1.d

In practice, k is the biggest value of sublist length for which insertion sort is faster than merge sort.
Worst-case for insersion sort: $C_1 k^2$
Worst-case for merge sort: $C_2 klogk$

$$C_1 k^2 < C_2 klogk$$

$$k < \frac{C_2}{C_1} logk$$

9.3.1

Groups of 7

The number of elements greater than x will be at least:

$$4(\lceil \frac{1}{2} \lceil \frac{n}{7} \rceil \rceil - 2) \geq \frac{2n}{7} - 8$$

Similarly, at least $\frac{2n}{7} - 8$. So the SELECT function will call itself recursively on step 5 on sub-problems of size (at most): $\frac{5n}{7} + 8$.

So we get the recurrence relation:

$$T(n) \leq \begin{cases} O(1), & if n \leq n_0 \\ T(\lceil \frac{n}{7} \rceil) + T(\frac{5n}{7} + 8) + O(n), & if n \geq n_0 \end{cases}$$

We can determine an upper bound for: $T(n) \leq cn$ and $O(n) \leq an$

$$T(n) \leq c(\lceil \frac{n}{7} \rceil) + c(\frac{5n}{7} + 8) + an \leq \frac{cn}{7} + c + \frac{5cn}{7} + 8c + an$$

$$\begin{aligned} &= \frac{6cn}{7} + 9c + an \\ &= cn - \frac{cn}{7} + 9c + an \leq cn \qquad (1) \\ &= O(n) \end{aligned}$$

Naturally, in order for 1 to hold, we need $-\frac{cn}{7} + 9c + an \leq 0$.
Getting:

$$c \geq \frac{7an}{n - 63}$$

If we assume that $n \geq n_0 = 126$, we have $\frac{n}{n-63} \leq 2$.
So choosing $c \geq 14a$ will satisfy the inequality.

Groups of 3

The number of elements greater than x will be at least:

$$2(\lceil\frac{1}{2}\lceil\frac{n}{7}\rceil\rceil - 2) \geq \frac{n}{3} - 4$$

Similarly, at least $\frac{n}{3} - 4$. So the SELECT function will call itself recursively on step 5 on sub-problems of size (at most): $\frac{2n}{3} + 4$.
We get the following recurrence relation:

$$T(n) \leq T(\lceil\frac{n}{3}\rceil) + T(\frac{2n}{3} + 4) + O(n)$$

We assume that $T(n) \geq cnlogn$ and bound O(n) with an.

$$T(n) >= T(\lceil\frac{n}{3}\rceil) + T(\frac{2n}{3} + 4) + an$$

$$= c(\frac{n}{3})log(\frac{n}{3}) + cnlogn + c(\frac{2n}{3})log(\frac{n}{3}) + an$$

$$= c(\frac{n}{3})log(\frac{n}{3}) + cnlogn + c(\frac{2n}{3})log(\frac{n}{3}) + an$$

$$T(n) \geq 2cnlogn + an$$

We have demonstrated that $T(n) = \Omega(nlogn)$.

9.3.7

1. Find the median of the input array (linear time).
2. We create a new array which contains the absolute value of the distance of each element to the median (linear time).
3. Find the k-th smallest element in the new array (k-th order statistics in linear time)
4. Select elements with distance $\leq$ k-th order statistic