HWw10-1

Samuel Pegg

November 2020

1 Page 22 2.1-3

```
i=NIL
for j=1 to n
    if A[j]==v
        i=j
        break
return i
```

List invariant: the list $A[1..j-1]$ does not contain v.

- Initialisation. When $j = 1$, the list $A[1..0]$ i.e. the empty list does not contain v.
- Maintenance. Assume $A[1..j-1]$ doesn't contain v. The next step of the algorithm will check if $A[j] = v$. If $A[j] = v$, then the algorithm returns $i = j$. If $A[j] \neq v$ then $A[1..j]$ does not contain v and hence loop invariance is preserved.
- Termination. If any of the $A[j] = v$ then the loop breaks and the algorithm returns the correct index i. If none of the $A[j] = v$ then i is not changed from NIL, and hence the algorithm returns $i =$NIL as desired.

2 Page 39 2-1

a

Insertion sort on a list of length k runs in $\Theta(k^2)$ worst-case time. This is repeated $\frac{n}{k}$ times, so

$$\Theta\left(k^2 \times \frac{n}{k}\right) = \Theta(nk)$$

b

We merge the $\frac{n}{k}$ sublists of length k in pairs until we get the full list of length n. This takes $\log\left(\frac{n}{k}\right)$ steps. Each step has $\Theta(n)$ complexity since we compare n elements at each step, hence the final complexity is

$$\Theta\left(n \log\left(\frac{n}{k}\right)\right)$$

as required.

c

We want $k \leq \log(n)$. This is since the upper bound, $k = \log(n)$, means that the expression $\Theta\left(nk + n\log\left(\frac{n}{k}\right)\right)$ becomes

$$\Theta\left(n\log(n) + n\log\left(\frac{n}{\log(n)}\right)\right) = \Theta\left(2n\log(n) - n\log\left(\log(n)\right)\right) = \Theta\left(n\log(n)\right)$$

d

k should be chosen such that insertion sort on lists of length k is faster than merge sort on lists of length k. If there are multiple k that satisfy this condition, we should choose the largest option.

3 Page 223 9.3-1

At least half of the $\lceil \frac{n}{7} \rceil$ groups contribute at least 4 elements greater than x. Excluding the group containing x and the last of the groups, which might not have 7 elements, there are at least

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

elements greater than x. Hence in the worst case, SELECT gets called on at most

$$n - \left(\frac{2n}{7} - 8\right) = \frac{5n}{7} + 8$$

elements. Hence

$$T(n) = T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{5n}{7} + 8\right) + O(n)$$

for sufficiently large n. Assume $T(n) \leq cn$ for all $n < k$ for some constant c. Then for $n \geq k$ we have

$$\begin{aligned} T(n) &\leq c\left(\left\lceil \frac{n}{7} \right\rceil\right) + c\left(\frac{5n}{7} + 8\right) + an \\ &\leq c\left(\frac{6n}{7} + 9\right) + an \\ &= cn + \left(-\frac{cn}{7} + 9c + an\right) \end{aligned}$$

so for $T(n) \leq cn$ for all $n > 0$ we require

$$\left(-\frac{cn}{7} + 9c + an\right) \leq 0 \iff c\frac{63 - n}{7} \leq -an \iff c \geq 7a\frac{n}{n - 63}$$

if $n \geq 64$. The largest $\frac{n}{n-63}$ can be is $\frac{64}{1}$, so we get the requirement

$$c \geq 448a$$

Thus using groups of 7 also works in linear time providing we choose a $c \geq 448a$. However, for groups of 3, following the analysis above, we have

$$2\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{3} \right\rceil \right\rceil - 2\right) \geq \frac{n}{3} - 2$$

elements greater than x, and hence SELECT gets called on at most

$$\frac{2n}{3} + 4$$

elements. Following the above analysis, this yields

$$T(n) \leq c\left\lceil \frac{n}{3} \right\rceil + c\left(\frac{2n}{3} + 4\right) + an \leq cn + 5c + an = (c + a)n + 5c$$

It is impossible for

$$(c + a)n + 5c \leq cn$$

since it would force

$$an \leq -5c$$

i.e. it would force $a < 0$. However, by definition $a > 0$, so this cannot be true and hence using groups of 3 will stop the algorithm running in linear time. Extending this, we guess $T(n) \geq cn\log(n)$ for all $n > 0$. Then

$$\begin{aligned} T(n) &\geq c\left\lceil \frac{n}{3} \right\rceil \log\left(\left\lceil \frac{n}{3} \right\rceil\right) + c\left(\frac{2n}{3} + 4\right)\log\left(\frac{2n}{3} + 4\right) + an \\ &\geq c\frac{n}{3}\log\left(\frac{n}{3}\right) + c\frac{2n}{3}\log\left(\frac{2n}{3} + 4\right) + an \\ &\geq c\frac{n}{3}\log\left(\frac{n}{3}\right) + c\frac{2n}{3}\log\left(\frac{2n}{3}\right) + an \\ &\geq cn\left(\frac{1}{3}\log(n) + \frac{2}{3}\log(n)\right) + cn\left(\frac{2}{3}\log(2) - \log(3)\right) + an \\ &\geq cn\log(n) \qquad \text{as long as } cn\left(\frac{2}{3}\log(2) - \log(3)\right) + an \geq 0 \end{aligned}$$

So $T(n)$ grows more quickly than linear providing we choose $a \geq c\left(\log(3) - \frac{2}{3}\log(2)\right)$.

4 Page 223 9.3-7

Use SELECT$\left(A, \left\lceil \frac{n}{2} \right\rceil\right)$ to find the median m in $O(n)$ time. Subtract m from each element of A (in $O(n)$ time) and call the resulting list B. Use SELECT(B, k) to find x, the k^{th} smallest element in B in $O(n)$ time. Finally, iterate through the array. At each step of the iteration, if $B[i] \leq x$ then add it to the output. Clearly, this is also $O(n)$ time, hence this algorithm is in $O(n)$ time as required.