CS 430 - SPRING 2023

INTRODUCTION TO ALGORITHMS

HOMEWORK #2

Name : Madhushalini Murali

A ID : A20513784

1) • It is given that the array is already sorted in descending order. Hence, the last element or the pivot element in the array is the least among all the other elements.

• At the first time, the partition step will take ~~Θ~~(n) time. ~~After~~ that, it will give a sub array of size $n-1$ and a subarray of size 0, which is the pivot element itself

• The subarray with size $n-1$ contains the remaining elements, for which the running time is again considered as recurrence. for this the running time is $\Theta(n)/T(n)$.

• Hence, running time to sort in increasing order using quick sort will be $T(n) = T(n-1) + n$

$$\Rightarrow \boxed{T(n) = T(n^2) \text{ or } \Theta(n^2)}$$

2a) Let us consider that 'A' has n elements.

If $n=1$, then the size of A is 1 element.

In this case, $\boxed{p=r.}$

Let us assume that $1 \leq k \leq n-1$, in this case the TAIL-RECURSIVE-QUICKSORT will correctly sort the array A containing k elements.

Let q be the pivot elements and by the induction hypothesis, TAIL-RECURSIVE-QUICKSORT correctly sorts the left subarray that is smaller in size.

After this, p gets updated to $q+1$ and the same steps are repeated and the array size is small.

Thus by induction hypothesis, QUICKSORT (A, 1, length[A]) correctly sorts the array A.

2b) · If the input array is already sorted, then the stack depth will be $\Theta(n)$. The right subarray will be having size 0. Hence there will be $(n-1)$ recursive calls before the while condition $p<r$ is violated.

2c) Modified Quicksort :

while $p < r$

do

q = PARTITION (A, p, r)

if $q < \lfloor (r-p)/2 \rfloor$ then

MODIFIED-TAIL-RECURSIVE-QUICKSORT $(A, p, q-1)$

$p = q+1$

else

MODIFIED-TAIL-RECURSIVE-QUICKSORT $(A, q+1, r)$

$r = q-1$

end if

end while

3a)

	i	j	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]
	0	13	13	19	9	5	12	8	7	4	11	2	6	21
	1	11	6	19	9	5	12	8	7	4	11	2	13	21
	2	10	6	13	9	5	12	8	7	4	11	2	19	21
	10	2	6	13	9	5	12	8	7	4	11	2	19	21

Parameters $p=1$, $r=12$.

3b) At the beginning of the loop, $i < j$.

$\Rightarrow$ It is true initially as longs as $|A| \geq 2$.

If this condition has to fail, then we would have left the loop in the prior iteration itself.

To show that the indices i and j cannot be accessed outside the subarray, we have to prove that at the beginning of every run of the loop, there is a $k > i$ so that $A[k] \geq x$ & $k' < j$ & $A[j'] \leq x$.

This is true because initially i and j are outside the bounds of the array & x must be between the two. Since $i < j$, consider $k = j$ & $k' = i$. The element k satisfies the desired relation to x, because the element at position i & j are the same which applies the same for k'.

3c) If the loop runs more than one time, there is $j < r$ because it decreases by one with every iteration.

At the last line of program (return j), value of i is 1 as $A[p] = x \geq x$. So if we terminate after single iteration of main loop, we should have $j = 1 < p$.

3d) After the iteration of the loop is finished, i become from i_1 to i_2 & j becomes from j_1 to j_2.

All the elements in $A[i_1+1 \ldots i_2-1]$ are less than x, because that does not terminate the loop from lines 8–10. Also after the exchange, $A[i_2] \le x \le A[j_2]$

By induction, all the elements in $A[p \ldots i_1]$ are less than or equal to x & all element in $\{A[j_1 \ldots r]\} >= x$.

Then $A[p \ldots i_2] = A[p \ldots i_1] \cup A[i_1+1 \ldots i_2-1] \cup \{A[i_2]\}$

and $A[j_2 \ldots r] = \cup \{A[j_2]\} \cup A[j_2+1 \ldots j_1-1] \cup A[j_1 \ldots r]$

have the desired inequality. At the termination, $i >= j$, $A[p \ldots j] \subseteq A[p \ldots i]$ & every element of $A[p \ldots j]$ is $<= x$ which is less than or equal to every element of $A[j+1 \ldots r] \subseteq A[j \ldots r]$

4) Maximum and minimum number of elements in a heap of height h:

- A complete binary tree of depth $h-1$ will have

$$\sum_{i=0}^{h-1} 2^{i} = 2^{h-1} \text{ elements.}$$

- The number of elements for a complete binary tree of depth $(h-1)$ exclusive and the number of elements for a complete binary tree of depth h inclusive decides the number of elements in a heap of depth h.

- Hence the maximum number of elements is $\underline{\underline{2^{h+1}-1}}$ and the minimum number of element is $\underline{\underline{2^{h}}}$.

5) Iterative MAX-HEAPIFY(A, i) (loop)

- The inputs are an array A and index i into the array.
- When the function is called, MAX-HEAPIFY will assume that the binary trees rooted at LEFT(i) and RIGHT(i) are MAX-HEAPS.
- MAX-HEAPIFY will let the value at A[i] to "float down" in the max-heap so the subtree rooted at index i will obey the max-heap property

Iterative MAX-HEAPIFY (A, i) : (declaring array A[i])

```
while (i < A.heap-size)
do
    left = LEFT (i)
    right = RIGHT (i)
    largest = i
    if left <= A.heap-size and A[left] > A[i] then
    then    largest = left
    end if
    if right <= A.heap-size and A[right] > A[i] then
            largest = right
    end if
    if largest ≠ i then
        exchange A[i] and A[largest]
    else    return A.
    end if
end while
return A
```

b) The smallest possible depth of a leaf in a decision tree for a comparison sort is $\boxed{n-1}$

If we construct a graph with vertices as the indices, then we join any 2 indices that are compared on the shortest path to form the edge. The graph has to be connected otherwise the algorithm will be ran twice. If we maintain the same relative ordering of the elements, then the algorithm will produce the same result.

Therefore for a graph of 'n' vertices, there will be (n-1) edges, because the addition of an edge can reduce the number of connected element by 1 & the graph with no edge will have n connected elements.

∴ The depth is $\underline{\underline{n-1}}$.