SOLO

Group: 13

PAGE NO.

DATE / /

Assignment 2
CS-202

Anubaal Dwivedi
B19071
CSE

Theory

Ques 1) We have red, white and blue elements.
Let us encode :-
red as 0, white as 1, blue as 2.
So, now we have to sort an array of 0s, 1s and 2s.

Algorithm :-

Input :- An array of size n.
Let A represents as [array] $[a_1, a_2, \ldots, a_n]$.

Initialise :- i = 0, low = 0, high = n-1.
[as it is a zero based indexing]

while (i <= high) :-
{ if A[i] is 0 :-
~~then, i = i+1, low = low+1;~~
then, swap(A[i], A[low]);
i = i+1, low = low+1;
else if A[i] is 1 :-
then, i = i+1;
else if A[i] is 2 :-
then, swap(A[i], A[high]);
high = high-1;

}

PAGE NO. _____
DATE 1 1

Now, as it traverses the array once only. So it is O(n).
Also, we can decode the sorted array of 0s, 1s, 2s with red, white and blue respectively.

Ques (2)

(a) Both are equally fast. The top element in the max-heap is maximum and in a sorted array, it is the last element. Both of which could be get in O(1) time.

(b) Max heap would be fast as it takes O(logn) time to delete but a sorted array could take O(n) time, because we have to shift all elements in worst case.

(c) Both the ~~to~~ max-heap and a sorted array take O(nlogn) time.

(d) The minimum element in a sorted array is the first. In a max-heap every leaf could be the minimum ~~elet~~ element. ∴ it takes O(n) time.

PAGE NO. _____

DATE __/__/__

Ques ③ :- If insert, maximum and extract-max would be possible in $O(1)$ time, we could use the following algorithm to sort data.

```
let A[1....n] be the array.
for i = 1 to n
    insert(A[i]);
for i = 1 to n
    A[i] = maximum();
    Extract-max();
```

So, this would sort data in $O(2n) = O(n)$ time.

But, we know that lower bound for sorting (comparison based) is $\Theta(n \log n)$, and our time complexity is less than lower bound.

So, it is a contradiction, our assumption was wrong, and Mr. B.C. Dull is mistaken.

PAGE NO.
DATE / /

Ques (4) :- Let us look at following algorithm for building a min-Heap of an input array A.

```
Build-Heap(A)
        heap_size = size(A);
        for i = floor(heap-size/2) to 1
            do heapify(A,i),
        end for
End
```

Now, we define the tighter bound by observing that running time of heapify depends on height of tree 'h'.
The height h increases as we move upwards.
Hence, heapify takes different time for each node, which is O(h).

Now, for finding time complexity of building a heap, we have $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ nodes with height h.

$$T(n) = \sum_{h=0}^{\log(n)} \left\lceil \frac{n}{2^{h+1}} \right\rceil * O(h)$$

$$\Rightarrow O\left(n * \sum_{h=0}^{\log n} \frac{h}{2^h}\right)$$

$$\leq O\left(n * \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \quad \text{---- (1)}$$

Also, using :-

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x} \quad \left[\text{Sum of infinite GP } x<1\right]$$

On differentiating & multiply by x,

$$\sum_{n=0}^{\infty} n x^n = \frac{x}{(1-x)^2} \quad \text{——— (2)}$$

Now, using (1) & (2):

$$T(n) = \Theta\left(n * \left[\frac{1/2}{(1-\frac{1}{2})^2}\right]\right)$$

$$\Rightarrow \Theta(n * 2)$$

$$= \Theta(2n)$$

So, $\boxed{T(n) = \Theta(n)}$

Hence, Proved

(Qus 5) We can do this by Radix Sort.
Run through the list of integers and convert each one to base n, then radix sort them. Each number will have at most $\log_n n^3 = 3$ digits so, there will be 3 passes of count sort. And we can do count sort in $O(n)$ time.

PAGE NO. _____

DATE / /

So, after performing count sort 3 times, we get time complexity as $O(3n) = O(n)$ which is linear.