# MID-SEMESTER EXAMS

Name:- Tushar Rathi  Date:- 12th March.
Scholar ID:- 2012174
Subject:- Algorithms (CS-206)

Q1. Pseudo Code:-

```
i ← 0  → O(1)
while i < n do
    if n = A[i] then → O(1)
    return i
    else
        i ← i+1  → O(1)
return -1
```

$X_n = O(n)$

Hence, the above pseudo-code has a time complexity of O(n), But since Bill needs to search the whole n x n array, and this pseudo-code is only for a single line of array, he needs to iterate over every row and call the function. Since there are n rows, the complexity will be n times,

i.e. $O(n) \times n = O(n \times n) = O(n^2)$

Hence, the worst case complexity in terms of n will be $O(n^2)$.

No, it is not a linear time algorithm. For linear algorithm, time complexity should have been O(n). But the time complexity for problem is $O(n^2)$.
So, it is not linear.

Q2. ans)

Input :- A n-element array A

Output :- The Array A with its elements rearranged into increasing order.

① Pseudo code of selection sort:-

```
SELECTION -SORT [A];
for i = 1 to A.length - 1
    min = 1
for j = i+1 to A.length
    if A[j] < A[min]
        min = j
    temp = A[j]
    A[min] = temp
```

② LOOP INVARIATS:-

⇒ At the start of each iteration of outer loop, the sub-array A[1.....i-1] contains the smallest i-1 element in array, sorted in non-decreasing order.

⇒ At the start of each iteration of inner for loop, A[min] is smallest number in the subarray A[i.....j-1]

iii) Why does it need to run for only the first n-1 elements, rather than all n elements?

Ans → In final step the algorithm will be left with two elements to compare. It will store, the smaller one in A[n-1] and the larger one in A[n]. The final one will be largest element in array, since all previous iterations would have sorted, but the last two element. If we run it for n times, we will end up with a redundant step that sorts a single element sub-array. So, we run it n-1 times.

iv) Best and worst case complexities?

⇒ In the best-case, the array is already sorted, the body is such that it was never involved Thus no. of operation is one operation.

Worst case complexity is; $(n-1)\left((n+2)/_{2}+4\right)$

Q 3. solution

Given recurrence relation,

$$T(n) = 2T(n/4) + \sqrt{n}$$

$$\Rightarrow T(n) = 2T(n/4) + \Theta(\sqrt{n})$$

We know that, standard form is;

$$T(n) = aT(n/b) + \Theta(n^k (\log n)^i)$$

Here, $a = 2$, $b = 4$, $k = 1/2$, $i = 0$

Here $a = b^k$ and $I_p = 0$

Therefore, case 2 of master method is applicable.

$$T \in \Theta(n^{\log_b a} \log^{i+1} n)$$

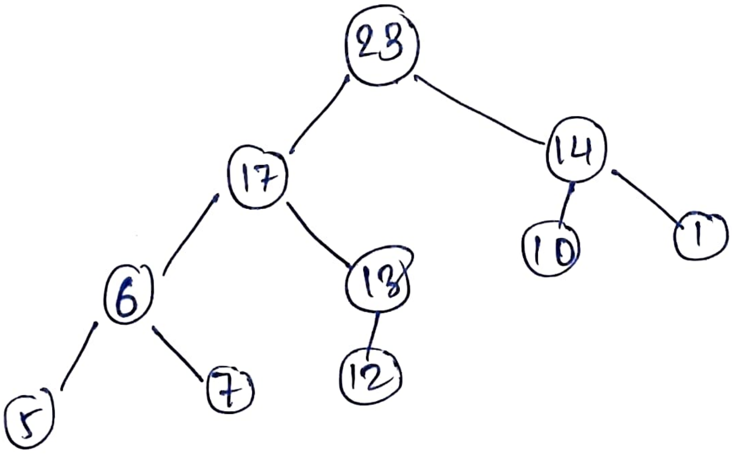
$$T \in \Theta(n^{1/2} \log n)$$

$$\therefore T \in \Theta(\sqrt{n} \log n)$$

---

Q 6. solution,

If the given array is converted into heap;



Since the node 6 has a childnode as 7 (i.e 7>

So, this will be not a max-heap.

2012174

Q.4. solution

Given recurrence $\Rightarrow T(n) = 3T(\sqrt{n}) + \log n$

Now, rename $\log n = m \Rightarrow n = 2^m$

$\therefore T(n) = 3T(\sqrt{n}) + \log n$

or, $T(2^m) = 3T(2^{m/2}) + m$

let $T(2^m) = S(m)$

or, $S(m) = 3S(m/2) + m$

Now, we guess $S(m) \leq cm^{\log 3} + dm$

$S(m) \leq 3(c(m/2)^{\log 3} + d(m/2) + m)$

or, $S(m) \leq cm^{\log 3} + \left(\frac{3}{2}d + 1\right)m \quad d(\leq -2)$

$\leq cm^{\log 3} + dm$

Then we guess, $S(m) \geq cm^{\log 3} + dm$,

$S(m) \geq 3(c(m/2)^{\log 3} + d(m/2)) + m$

or $S(m) \geq cm^{\log 3} + \left(\frac{3}{2}d + 1\right)m \quad (d \geq -2)$

or, $S(m) \geq cm^{\log 3} + dm$

Thus,

$S(m) = \theta(m^{\log 3})$

$T(m) = \theta((\log n)^{\log 3})$

Ans

Q5.ans) The Buck sort Algorithm is not an "in-place" algorithm.
As we more elements to buckets, which take up additional space, as big as the original array size or even more than that. Hence, due to the use of separate additional space, bucketsort is not an "in-place" algorithm.

---

Q8. Here, given array is A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]

6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2

Max element is the array is 6

Now, keeping a separate array for counting, the counting sort looks like.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2, | 2 | 2 | 2 | 1 | 0 | 2 | ← count |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | ← indices |

Now, we modify the count array with prefix sum,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 8 | 9 | 9 | 11 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | ← indices |

Rotating clockwise for 1 time,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 4 | 6 | 8 | 9 | 9 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | ← indices |

Outputing each object from the input sequence followed by increasing its count by 1,
We get,

0 0 1 1 2 2 3 3 4 6 6

Thus, we get the sorted array.

. 87. ans)

let $S_n$ be a set of values of nodes of height h.

Let $N_n$ be the number of nodes of height h

Since $N_{n-1}$ of these subtrees are full,

∴ Each subtree contains exactly $2^{n+1}-1$ nodes.

One of the height h subtrees may be not full, but it contains atleast 1 node at its lower level and has $2^k$ nodes. The remaining nodes have height strictly more than h.

To connect all subtrees rooted at node $S_n$, there must be exactly $N_{n-1}$ such nodes.

The total number of nodes is set atleast $\underline{(N_n-1)(2^{n+1}-1)+2^n+N_n-1}$,

while,

at most $\underline{N_n 2^{n+1}-1}$

Now,

$(N_n-1)(2^{n+1}-1)+2^n+(N_n-1) \le n \le N_n(2^{n+1}-1)+N_n-1$

$\Rightarrow -2^n \le n - N_n 2^{n+1} \le -1$

$\Rightarrow$ The fractional part of $n/2^{(n+1)}$ is $\geq 1/2$

$\Rightarrow \quad N_n \le \left[ \frac{n}{2}^{n+1} \right]$