Mahesh Koppala
1001764522

DAA Exam1

(1) $T(n) = 2T(n/4) + 1$

This is in the form of $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

So, we can use master's theorem

Compare $n^{\log_b a}$ with $f(n)$

$a=2,\ b=4$.

$n^{\log_b a} \Rightarrow n^{\log_4 2} \Rightarrow n^{0.5} = \sqrt{n}$

$\sqrt{n} > f(n) = 1$.

So, this comes under case 1.

So, time complexity is $\theta(\sqrt{n})$

(2) Given, $f(n) = n^3 + n^2 + n + 1$

(a) $f(n) = O(n^4)$

$\Rightarrow g(n) = n^4$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3+n^2+n+1}{n^4}$$

$$= \lim_{n\to\infty} \frac{1}{n} + \frac{1}{n^2} + \frac{1}{n^3} + \frac{1}{n^4}$$

$$= 0$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0, & f(n) \text{ has smaller order of growth than } g(n) \\ c, & f(n) \text{ has same order of growth as } g(n) \\ \infty, & f(n) \text{ has larger order of growth than } g(n) \end{cases}$$

$\therefore f(n) = n^3+n^2+n+1$ has smaller order of growth than $g(n) = n^4$.

So $f(n) = O(n^4)$

(b) $f(n) = \Theta(n^3)$

$g(n) = n^3$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3+n^2+n+1}{n^3}$$

$$= \lim_{n\to\infty} 1+\frac{1}{n}+\frac{1}{n^2}+\frac{1}{n^3} = 1$$

$f(n)$ has same order of growth as $g(n)$.

$\therefore f(n) = \Theta(n^3)$

(c) $f(n) = \Omega(n^2)$

$g(n) = n^2$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3+n^2+n+1}{n^2}$$

$$= \lim_{n\to\infty} n+1+\frac{1}{n}+\frac{1}{n^2}$$

$$= \infty$$

$\therefore f(n) = \Omega(n^2)$

(3) $T(n) = 2T(\lfloor n/2 \rfloor) + n$

We need to show that the above equation is in $T(n) = O(n \lg n)$

So we must prove that $T(n) \leq cn \log n$ for some constant c.

As our inductive hypothesis, we assume $T(n) \leq cn \log n$ for all positive numbers less than n.

Therefore $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$, and

$$T(n) \leq 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n$$

$$\leq cn \log(n/2) + n$$

$$= cn \log n - cn \log 2 + n$$

$$= cn \log n - cn + n$$

$$\leq cn \log n \quad (\text{for } c \geq 1)$$

Now $T(2) = 2T(1) + 2 = 4$

$T(3) = 2T(1) + 3 = 5$.

We have to choose a c that satisfies those constraints on $T(2)$ and $T(3)$

We can choose $c = 2$, because $4 \leq 2 \cdot 2 \log 2$ and $5 \leq 2 \cdot 3 \log 3$.

$\therefore T(n) \leq 2n \log n$ for all $n \geq 2$. So $T(n) = O(n \log n)$

(4) $T(n) = T(n-1) + n \quad , T(0) = 0$

$T(1) = T(1-1) + 1$
$= T(0) + 1$
$= 0 + 1$
$T(1) = 1$

$T(n) = T(n-1) + n$
$= (T((n-1)-1) + (n-1)) + n$
$= T(n-2) + (n-1) + n$
$= T(n-2-1) + (n-2) + (n-1) + n$
$= T(n-3) + (n-2) + (n-1) + n$
.....
$= T(1) + 2 + 3 + \dots + n$
$= 1 + 2 + 3 + \dots + n$
$= n(n+1)/2$
$T(n) = n(n+1)/2$

Now, we have to prove the above guess pattern using Induction.

As per Induction for $T(n) = n(n+1)/2$ we can write

(1) Let's prove that $T(1) = 1$
$T(n) = n(n+1)/2$
$T(1) = 1(1+1)/2$
$= 1$
$T(1) = 1$ is proved

(2) Assume $T(n-1)$ is true means

$T(n-1) = (n-1)(n-1+1)/2$ is true

(3) Now we will prove that $T(n)$ is also true

$$T(n) = T(n-1) + n$$
$$= (n-1)(n-1+1)/2 + n$$
$$= (n-1)(n)/2 + n$$
$$= n(n-1)/2 + n$$
$$= n^2/2 - n/2 + n$$
$$= n^2/2 + n/2$$

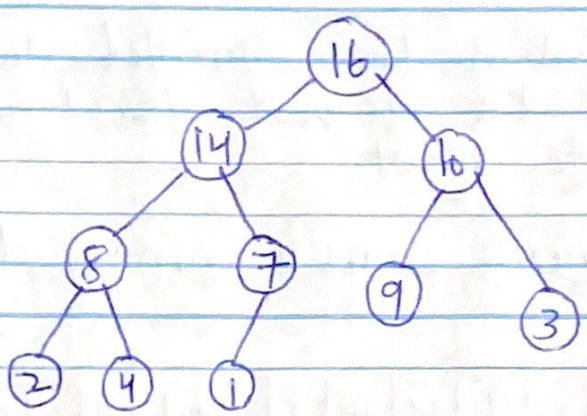
$$T(n) = n(n+1)/2$$

Hence, it is proved that $T(n)$ is true. So, as per the Induction, $T(n) = n(n+1)/2$ is true and it is our solution.

And complexity is $O(n^2)$ as

$$T(n) = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} = O(n^2)$$

(5)



(a) This is Max Heap.

Max Heap property: Key of a node $\geq$ Keys of children.

from the given tree, Key of any node is greater than the Key of it's children.

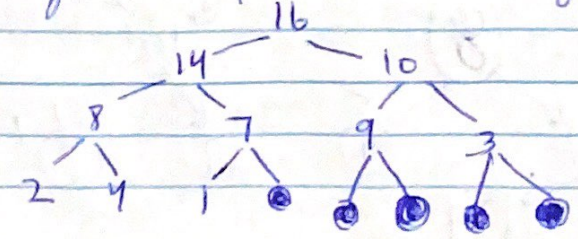
for Eg:- $16 > 14$ and $16 > 10$
$14 > 8$ and $14 > 7$

→ This is not Min Heap.

Min Heap property: Key of a node $<$ Keys of children

for Eg:- $16 < 14$ ✗
$16 < 10$ ✗

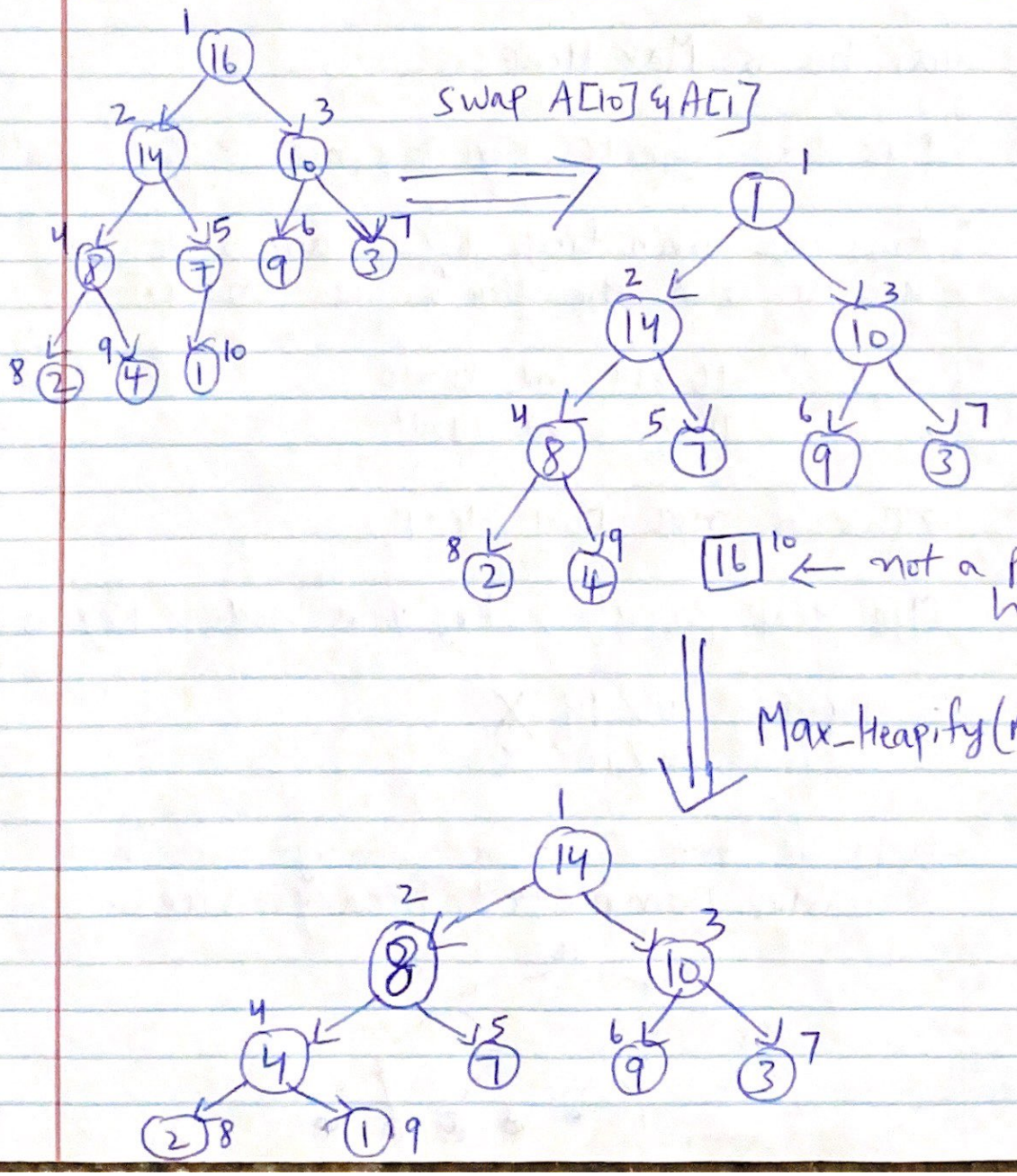
→ It is not Just a binary tree but "nearly complete" binary trees.



We can think of the unfilled slots as null pointers. the lowest level is filled left to right.

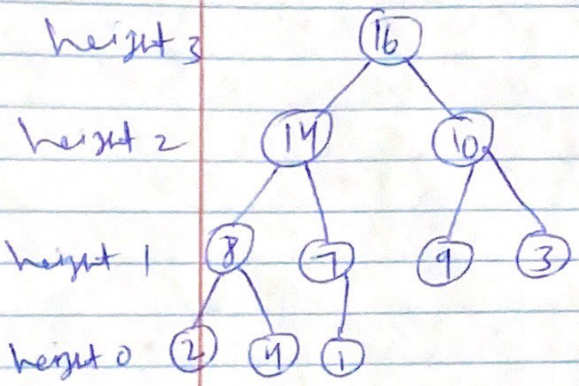
So, it's a nearly complete binary tree.

(b) $A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$

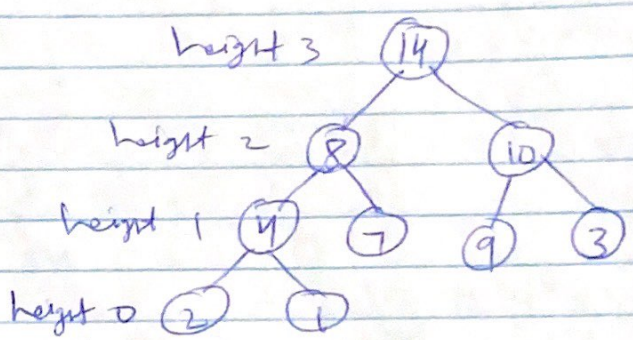


(c) Height of Tree = Height of Root.
Height of the tree is 3.

before removing 16



After removing 16.



Height of Heap = $O(\log n)$

$\Rightarrow O(\log 10) \Rightarrow$ before removing 16

$\Rightarrow O \log(9) \Rightarrow$ After removing 16.

(b) A = (2, 8, 7, 1, 3, 5, 6, 4)

J→0	1	2	3	4	5	6	7
2	8	7	1	3	5	6	4 ← pivot element.

Let i = -1, J = 0

Quick sort Logic :-

if A[J] > pivot then increment J by 1
if A[J] < pivot then increment i by 1 and swap A[i], A[J]. and increment J by 1

Ex:- i=-1, J
2, 8, 7, 1, 3, 5, 6, 4

2 < 4

i, J
2, 8, 7, 1, 3, 5, 6, 4

8 > 4, so increment J by 1.

i J
2, 8, 7, 1, 3, 5, 6, 4

7 > 4, so increment J by 1.

```
    i   J
2, 8, 7, 1, 3, 5, 6, [4]
```

1<4, so increment i by 1, swap 8 and 1 and then increment J by 1.

```
    i      J
2, 1, 7, 8, 3, 5, 6, [4]
```

3<4, so increment i by 1, swap 7 and 3 and then increment J by 1

```
       i      J
2, 1, 3, 8, 7, 5, 6, [4]
```

5>4, so increment J

```
       i         J
2, 1, 3, 8, 7, 5, 6, [4]
```

6>4 and J has reached end.

Now, swap A[i+1] and the pivot.

[2, 1, 3, 4, 7, 5, 6, 8] is the array elements after one partition.

Elements to the left of 4 are less than 4.
Elements to the right of 4 are greater than 4.

(7) (a) Yes, Bubble Sort is a stable Sorting algorithm because the relative order of any two equal elements is preserved in the sorted array.

Eg- 7, 2, 3, 7, 1, 6, 4.

In the input list, there is one pair of equal elements 7 at position 0 and 3. In order to show that bubble sort is stable, the order of positions of the two equal elements 7 must be same in the sorted array which means the element 7 at position 0 must come after the element 7 at position 3.

The sorting of elements using bubble sort is as follows:

position	0	1	2	3	4	5	6
Initial Array	7	2	3	7	1	6	4
pass 1	2	3	7	1	6	4	7
pass 2	2	3	1	6	4	7	7
pass 3	2	1	3	4	6	7	7
pass 4	1	2	3	4	6	7	7

If we consider the pass1, by end of the pass1, the element 7 at position 0 will come after the element 7 at position 3.

Hence, bubble sort algorithm is a stable algorithm.

(b) Heap sort is not a stable sorting algorithm because operations in the heap can change the relative order of the equivalent keys.

Eg:- Consider an array 21 20a 20b 12 11 8 7
the above array is already in max-heap format

Here 20a=20b, Just to differentiate the order we represent them as 20a and 20b

while performing heap sort, first 21 is removed and placed in the last index, then 20a is removed and placed in last but one index and 20b in the last but two index. so after heap sort, the array looks like:

7 8 11 12 20b 20a 21

It doesn't preserve the order of elements and hence can't be stable.

(c) Yes, Bubble sort is an in-place sorting algorithm.

Bubble sort algorithm doesn't need extra space except the input space excluding the constant space used for iterators. It usually also excludes the space used for stack in recursive algorithms.