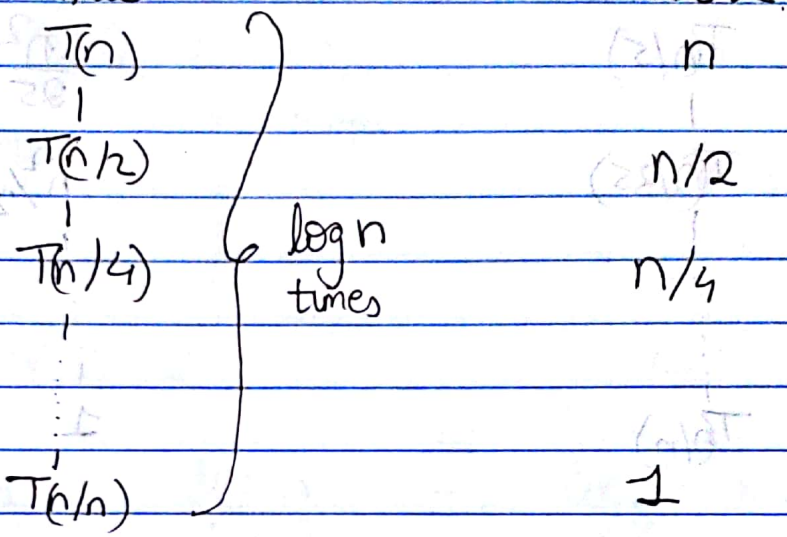
Written Assignment 2

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



$$\text{Work} = n + \frac{n}{2} + \frac{n}{4} + \dots 1$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots \frac{1}{n}\right)$$

$$\leq n(2)$$

$$= O(n)$$

b) $T(n) = T(n/5) + n^2$

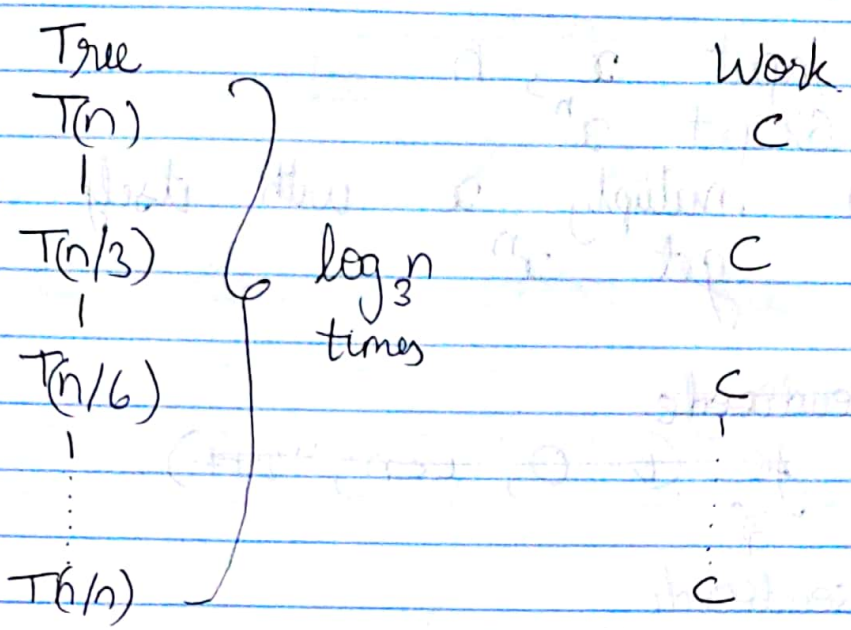
Tree	Work
$T(n)$	n^2
1	
$T(n/5)$	$\frac{n^2}{25}$
1	
$T(n/25)$	$n^2/25^2$
1	
⋮	⋮
1	
$T(n/n)$	1

$$\text{Work} = n^2\left(1 + \frac{1}{25} + \frac{1}{25^2} + \cdots \frac{1}{n^2}\right)$$

$$\leq n^2(2)$$

$$= O(n^2)$$

c) $T(n) = T(n/3) + c$



$$\log_3 n = \frac{\log_2 n}{\log_2 3} = \log_3 2 \times \log_2 n$$

$$\text{Work} = c \times \log_3 n$$
$$= c \times \log_3 2 \cdot \times \log_2 n$$
$$= O(\log n)$$

2] → Algorithm to compute x^n

Input x, n
Output x^n
1. Multiply x with itself n times to get x^n

~~Pseudocode~~
~~for (i = 0; i < n; i++)~~
~~{~~

→ Pseudocode:

```
mult = 1
for (i = 0; i < n; i++)
        mult = mult * x;
```

→ The Time Complexity of the above pseudocode is $O(n)$ as there is a for loop which runs n times.

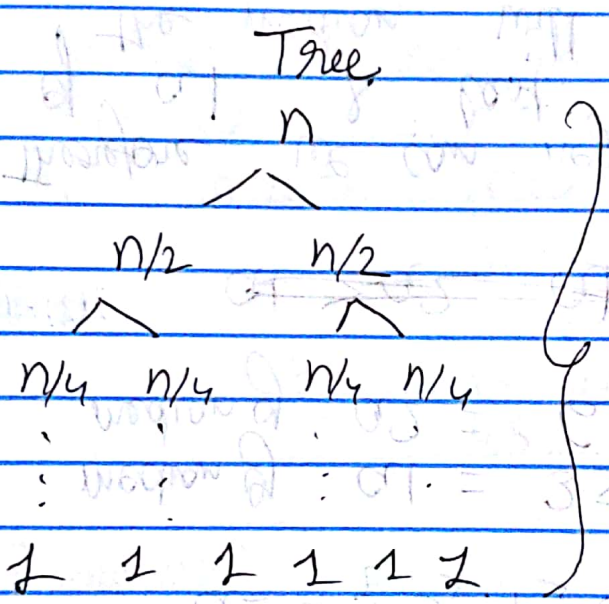
→ Yes, we can go faster than $O(n)$ by using divide and conquer.

→ Pseudocode

Input = x, n.
Output = x^n.

```
proc power(x, n):
    if (n == 0)
        return 1
    mult = power(x, n/2)
    if (n % 2 == 0)
        return mult * mult
    else
        return x * mult * mult
```

Time complexity of above algo is O(log n)



Work = c x log n

= O(log n)

3] We can use Divide & Conquer to find median of 2 sorted arrays with equal length k in $O(n)$ time.

Explanation :-

Consider 2 sorted arrays of length 4.

$a1 = \{1, 2, 5, 7\}$

$a2 = \{3, 4, 8, 9\}$

median of $a1 = 3.5 = m1$

median of $a2 = 6 = m2$

~~$a1 > a2$~~ ~~$a1 < a2$~~ $m1 < m2$

Therefore we can neglect the first element of $a1$ & last element of $a2$ as the median will lie between 2-8

$\therefore a1 = \{X, 2, 5, 7\}$

$a2 = \{3, 4, 8, X\}$

now median of $a1 = 5 = m1$

& median of $a2 = 4 = m2$

$m2 < m1$

~~$\therefore a1 = \{X, X, 5, 7\}$~~

∴ a1 = {*, 2, 5, *}
a2 = {*, 4, 8, *}.

now median of a1 = ~~3.5~~ = m1
& median of a2 = ~~6~~ = m2

Reference :- www.geeksforgeeks.org/median-of-two-sorted-arrays/

Now number of elements in each array = 2

∴ median = [max(ar1(0), ar2(0)) + min(ar1(1), ar2(1))]/2

= (2+5)/2
= 2.5

Pseudocode :-

Input : 2 arrays a1 & a2 of same length n
Output : median of the two arrays
proc findMedian(a1, a2, n):
Reference geeksforgeeks.org/median-of-two-sorted-arrays/
1. if (n == 2)
return [max(ar1[0] + ar2[0]) + min(ar1[1] + ar2[2])]/2

2. m1 = median(a1)
3. m2 = median(a2)
4. if m1 == m2
   return m1
5. if m1 < m2 // consider a1[m1....n] & a2[0....m2]
   if n is even.
   return find ~~return~~ Median (a1[0.... n/2+1], a2[n/2 - 1 .... n], n - n/2 + 1);

   if n is odd.
   return find Median (a1[0.... n/2], a2[n/2 .... 0], n = n/2)

6. if m1 > m2 // consider a1[0.... m1] & a2[m2.... n]
   if n is even
   return findMedian (a2[0.... n/2+1], a1[n/2 - 1 .... n], n - n/2 + 1);
   if n is odd
   return findMedian (a2[0.... n/2], a1[n/2 .... n], n - n/2)

Explaination:-

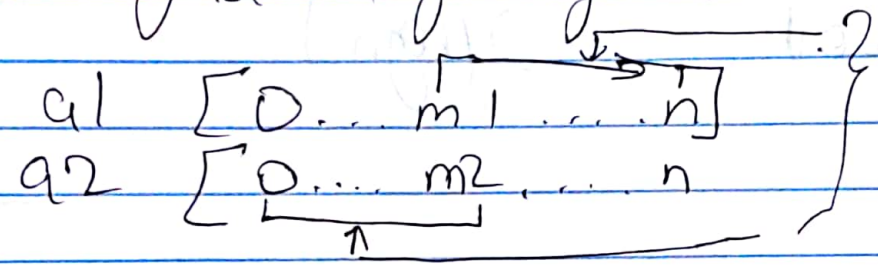
Find median of 2 arrays

let m1 be median of a1

& m2 be median of a2.

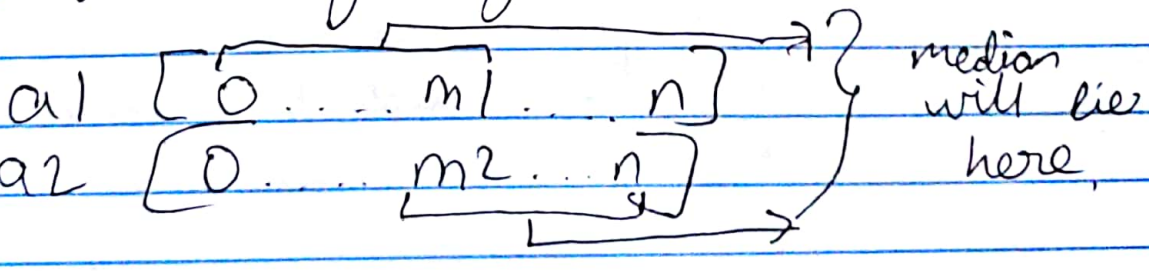
if m1 < m2

then median of both arrays will lie in the second half of a1 and first half of a2.

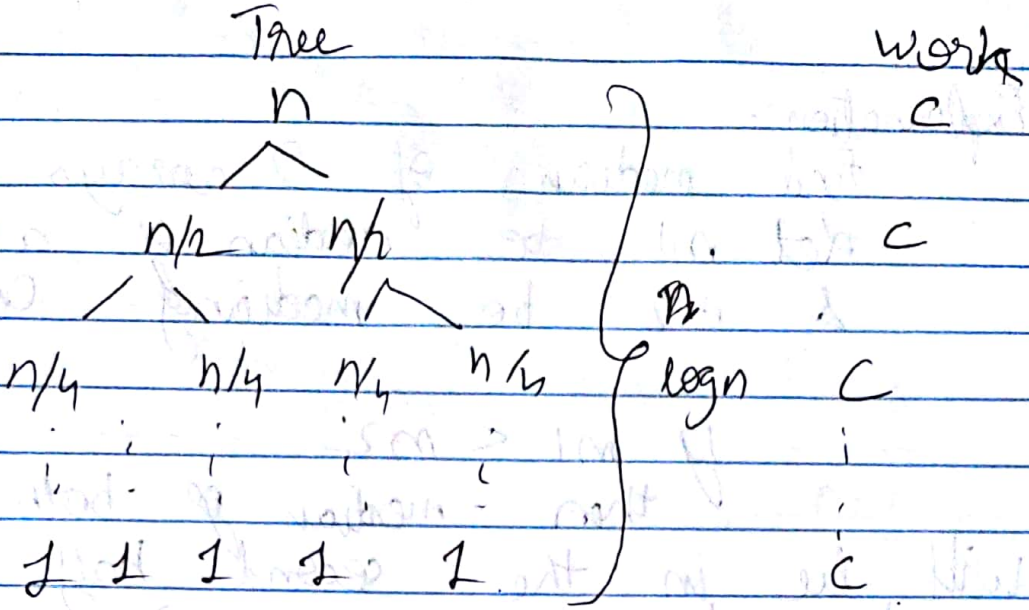


if m1 > m2

then median of both arrays will lie in the first half of a1 or second half of a2.



Proof :-



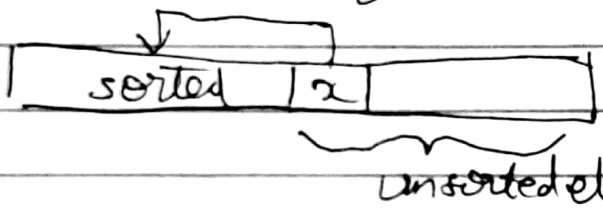
Work = $O(\log n)$

Q4 a]. A[] is nearly sorted

- Insertion sort or Bubble sort can be used for nearly sorted arrays. As they have O(n) Time Complexity for Best Case Scenario.

Description of Insertion sort.

Consider the array below.



→ The algorithm will pick up element x and put it in place in the sorted sub-array.
→ The algorithm will do it this again and again till all the elements are placed in order.

Reference CLRS pg 18 - - - - - - -

Insertion sort (arr, n) # arr = array
n = length of arr.

```
1. for j = 2 to n
2.     key = A[j]
3.     i = j - 1
4.     while i > 0 and A[i] > key
5.         A[i+1] = A[i]
6.         i = i - 1
7.     A[i+1] = key
```

- - - - - - - - - - - - - - - - - -

6) A[] consists of random numbers.
- Merge sort or Quick sort can be used for array having random numbers as they have a Time complexity of O(n log n)

Reference CLRS pg 171 ----------

Quicksort Algorithm:-

```
Quicksort (A, p, r)
1.  If p < r
2       q = Partition (A, p, r)
3.      Quicksort (A, p, q-1)
4       Quicksort (A, q+1, r)

Partition (A, p, r)
1.  x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.          if A[j] ≤ x
5.              i = i+1
6.              exchange Ai with A[j]
7   exchange A[i+1] with A[r]
8   return i+1
```

- - - - - - - - - -

Interesting Finding :-

On the website "www.toptal.com/developers/sorting-algorithms/", I noted that Merge sort will perform better than Quick sort in scenarios where array has few unique numbers. Also in worse case scenario Quick sort has Time complexity of $O(n^2)$ whereas Mergesort is $O(n \log n)$.