Vardaan

2024602

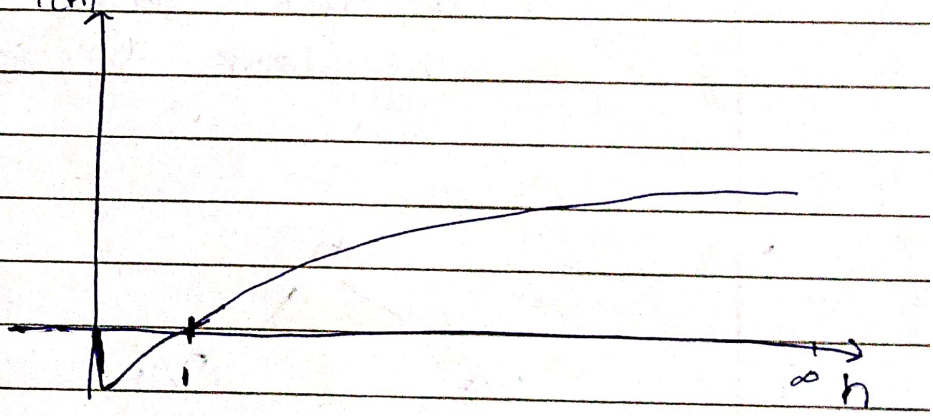
1.

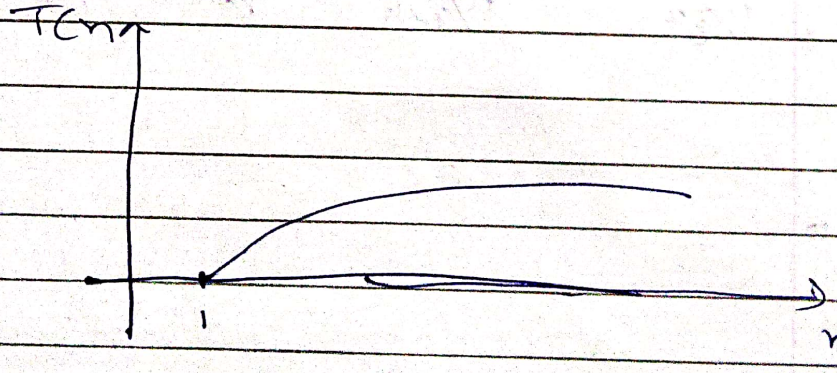
1.1 i) $T(n) = 2T\left(\frac{n}{8}\right) + \sqrt[3]{n}$

$n^{\log_8 2} = n^{1/3} = f(n)$

$\therefore T(n) = O(n^{1/3} \log n)$

2) $T(n)$





as $n \geq 1$ always

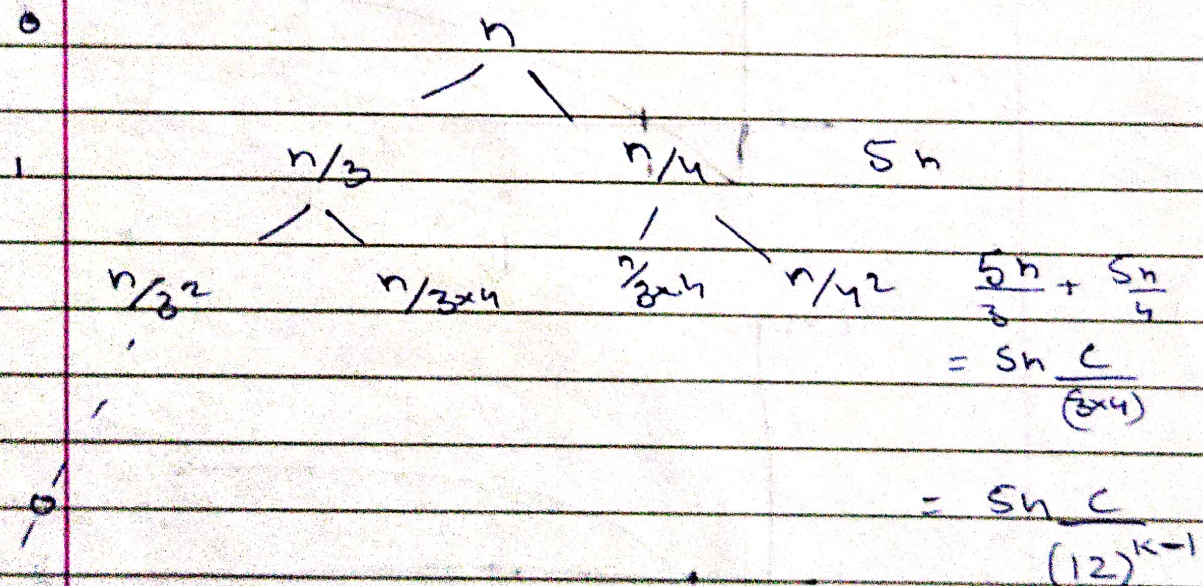
1.2 L0 $T(n) = T(n/3) + T(n/4) + 5n$

L1 $T\left(\frac{n}{3}\right) + T\left(\frac{n}{3\times4}\right) + \frac{5n}{3} + T\left(\frac{n}{3\times4}\right) + T\left(\frac{n}{4^2}\right) + \frac{5n}{4}$

L2 $\frac{5n}{3^2} + \ldots + \frac{5n}{3\times4} + \cdots + \frac{5n}{3\times4} + \frac{5n}{4^2}$

Cost of any level has 2^k terms where K is the level number

∴ the cost of K^{th} level will look like $\Rightarrow 5n \cdot \sum_{i=1}^{2^k} \frac{1}{\text{factor}}$



(3)

$$T(n) \leq n\left[c + \frac{c}{12} + \cdots \frac{c}{12^k} \cdots\right]$$

$$T(n) = O(n)$$

at k^{th} level of tree

∴ the cost is $= \frac{nc}{(12)^{k-1}}$

2. Let $T(n) \leq Cn$

$$T(n) = T(n/3) + T(n/4) + 5n$$

$$\leq Cn/3 + Cn/4 + 5n$$

$$\leq n(C/3 + C/4 + 5)$$

$$\leq Cn$$

$$T(n) \leq Cn$$

$$\therefore T(n) = O(n)$$

Scanned with OKEN Scanner

Date :
Page No. :

2.1 ① #include <stdio.h>
#include <string.h>

void Swap(char arr[][100], int ele1,
int ele2)
{ ~~temp[100] =~~ char temp[100];
strcpy(temp, arr[ele1]);
strcpy(arr[ele1], arr[ele2]);
strcpy(arr[ele2], temp); }

int main() {

char arrTitle[N][100] = { "T1", "T2" ...
... "TN" }

for (i=1, i<N, i++) {
pre = i-1;
while (pre >=0 && strlen(arrTitle
[pre+1] <= strlen(arrTitle[pre]))
{ swap(arrTitle, pre, pre+1);
pre = pre - 1;
}
}

Time complexity
Worst case

$1 + 2 + 3 \cdots (n-1) = \frac{n(n-1)}{2}$

$= O(n^2)$

eg [4,3,2,1] ⇒ [1,2,3,4]

Scanned with OKEN Scanner

Date :
Page No. :

considering books to be assigned a integer

2.1 (2) input : books [ ], target

left, right = 0, length(books) - 1

while (left ≤ right): {
mid = left + (right - left)/2

if (books[mid] == target) {
printf("BOOK found");
break ; }
elif (books[mid] < target) {
left = mid + 1 ; }

else { right = mid - 1; }

}

Printf ("Book not found")

Time complexity

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

$n^{(\log_2 1)} = 1 = f(n)$

$\therefore T(n) = O(\log n)$

OR

$\frac{n}{2^T} = 1 \Rightarrow T = \log_2 n$

Scanned with OKEN Scanner

Date :
Page No. :

2.0.1 (3)

a) Sorting once with quick sort or merge sort will take

$$O(n \log n)$$

let num of serches be m

∴ for binary serch time = $O(m \log n)$

∴ Total time complexity

$$= O(n \log n) + O(m \log n)$$

$$= O((n+m) \log n)$$

b) doing multible linear serches

for one serch Time complexity = $O(n)$

∴ for m serches Time complexity

$$= O(m n)$$

Scanned with OKEN Scanner

Date :
Page No. :

Ex: let $m = 100$
$n = 1000$

∴ with case 1

Time complexity = $O((100+1000)\log(1000))$

$= O(1100 \log(10)^3)$

$= O(3 \times 1100 \log_2 10)$

with case 2

Time complexity $O(100 \times 1000)$

$= O(10^5)$

$\because O((m+n)\log n) < O(mn)$

∴ case 1 is more optimised.

Date :
Page No. :

2.2 (1) void swap (arr[], int ii, int if) {

int temp = arr[ii]

int arr[ii] = arr[if]

int arr[if] = ~~arr[~~ temp. }

int main () {

Input: int arr[2n], number of element in pairs

let arr[2] be like $x_1 y_1 x_2 y_2 \cdots x_n y_n$

for (int i = 1, i < 2n, i++) {

if (i % 2 == 0) {

int j = i

while (arr[j-2] < arr[j] && j-2 >= 0)

{ swap (arr, j, j-2)

swap (arr, j+1, j-1)

j = j-2 }

} }

int prev = ~~9999~~ or int max

for (int i = 1, i < 2n, i++) {

if (i % 2 == 1) {

{ if (arr[i] <= prev) {

prev = arr[i];

printf ("~~arr~~ %d, %d", arr[i-1], arr[i])

}

}

Date :
Page No. :

2. (2) time complexity for sorting is $O(n^2)$

and for printing is $O(n)$

$\therefore$

$$T(n) = O(n^2) + O(n)$$
$$= O(n^2)$$

3. (1) The time complexity for the code is

$O(\log n)$ to find the target

and $O(1)$ to find the right and the leftmost elements of the target.

$$\therefore T(n) = O(\log n) + O(1)$$
$$= O(\log n)$$

& its Space complexity is $O(1)$ as now new space is consumed in the algorithm

(10)

Date :
Page No. :

In bruteforce approch.

The time complexity is

$T(n) = O(n)$ due to linear search.

& Space complexity is $O(1)$.

2. Yes my algorithm is the most efficient in finding the range & as it uses binary search to find the target leading to a time complexity of $O(\log n)$ which is much better than the $O(n)$ in the bruteforce method.

(11)

Date :
Page No. :

4. $T(n) = O(n)$

↳ Space complexity = $O(n)$

To find the person that will be left after moving k^{th} steps we can be calculate using a recursive function

in which

```
int person_teller(int n, int k){
    if(n == 1) {return 1;}
    else {return (person_teller(n-1,k) + k)%n+1;}
}
```

If we use $k=2$ then this will work as previous one only.