Solutions of Question - 01

(A) $f_1(n) = n^3 + 7n^2 = O(n^3)$

(B) $f_2(n) = (\log n)^{2023} = O\left((\log n)^{2023}\right)$

(C) $f_3(n) = \log(n!)$

By Stirling's approximation, $\log(n!) \approx n\log n - n$.

Hence, in Big-Oh notation, this simplifies to $O(n\log n)$

Alternative Approach:-

$f_3(n) = \log(n!) = \log\left(n\times(n-1)\times(n-2)\times(n-3)\times \cdots\cdots \times 3\times 2\times 1\right)$

$= \log(n\times n\times n\times n\times \cdots\cdots \times n\times n\times n)$ [As Upper Bound of each term $= O(n)$]

$= \log(n^n) = n\log(n)$

$= O(n\log n)$

(D) $f_4(n) = n^2 \log_n n^n$

$= n^2 \times n \times \log_n n$ [Power rule of logarithm]

$= n^3 \times 1$

$= O(n^3)$

(E)

$f_5(n) = n * \sqrt[5]{n^2}$

$= n \times n^{\frac{2}{5}}$

$= n^{1+\frac{2}{5}}$

$= n^{\frac{7}{5}}$

$= O(n^{1.4})$

Day :

Time : Date : / /

Solutions of Questions-02

(A)

Outer loop: for(i=1; i<=n; i++)

- executes starting from 1 to n.

Inner loop: for(j=1; $\boxed{j<i}$; j*=2)

- loop controlling condition depends on i (outer loop variable). To solve this, let's forget about the outer loop for a moment and assume i is just a very big integer. The loop will keep running as long as it is j<i and after each iteration j gets multiplied by 2.

Hence, $j = 1,\ 1\times2,\ 1\times2^2,\ 2^3,\ 2^4, \ldots\ldots$

After k-th iteration, let's uppose

$$2^k >= i$$

$$\Rightarrow 2^k = i \Rightarrow k = \log_2(i)$$

As the range of i is from 1 to n. Hence, the op(); gonna executes for —

$$\sum_{i=1}^{n} \log_2(i) = \log_2(1) + \log_2(2) + \log_2(3) + \ldots\ldots + \log_2(n)$$

$$= \log_2(1\times2\times3\times\ldots\ldots\times n) \quad \text{[Product rule of logarithm]}$$

$$= \log(n!)$$

$$\approx n\log n - n \quad \text{[Stirling's Approximation]}$$

Hence, time complexity of this code snippet :- $O(n\log n)$.

Day :
Time : Date : / /

(B)

Outer loop: for(i=1; i<=n; i++)

- executes starting from $i=1$ to $i=n$.

Inner loop: for(j=1; j<=n; j+=i)

- Here, the loop increment is dependent on the outer loop variable i.

To solve this, let's forget about the outer loop for a moment and assume i is just an integer through which we're incrementing j. Hence, j will go like -

$j = 1,\ 1+i,\ 1+2i,\ 1+3i,\ i+4i, \ldots\ldots, 1+ki$

Now, let's assume,

$1+ki > n \Rightarrow ki = n \Rightarrow k = \frac{n}{i}$

So, for each value of i, the inner loop runs for $O(\frac{n}{i})$ times. Hence, op(); will get executes in total -

$$\sum_{i=1}^{n} \frac{n}{i} = \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \ldots\ldots + \frac{n}{n}$$

$$= n \times \left[\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right]$$

$$= n \times \sum_{i=1}^{n} \frac{1}{i}$$

$$= n \times \ln(n)$$ [As a key approximation for the Harmonic series. is, $H_n \approx \ln(n)$]

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

Hence, the time complexity of this code snippet is $O(n \log n)$

Solutions of Question-01

(A)

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

$$= 3\times\left[3T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$$

$$= 3^2 T\left(\frac{n}{2^2}\right) + \frac{3n}{2} + n$$

$$= 3^3 T\left(\frac{n}{2^3}\right) + \frac{3^2 n}{2^2} + \frac{3^1 n}{2^1} + n$$

$$\vdots$$

$$= 3^k T\left(\frac{n}{2^k}\right) + \frac{3^0 n}{2^0} + \frac{3^1 n}{2^1} + \frac{3^2 n}{2^2} + \cdots + \frac{3^{k-1} n}{2^{k-1}}$$

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

$$= 3^{\log_2 n} + n\times\left[\frac{3^0}{2^0} + \frac{3^1}{2^1} + \frac{3^2}{2^2} + \cdots + \frac{3^{k-1}}{2^{k-1}}\right]$$

$$= 3^{\log_2 n} + n\times\left[\frac{1\times\left(\frac{3^k}{2^k} - 1\right)}{\frac{3}{2} - 1}\right]$$

$$= n^{\log_2 3} + n\times\left[2\times\left(\frac{3^{\log_2 n}}{2^{\log_2 n}} - 1\right)\right]$$

$$= n^{\log_2 3} + n\times\left[2\times\left(\frac{n^{\log_2 3}}{n^{\log_2 2}} - 1\right)\right]$$

$$= n^{\log_2 3} + n\left[2\times\left(\frac{n^{\log_2 3}}{n} - 1\right)\right]$$

$$= n^{\log_2 3} + 2n^{\log_2 3} - 1$$

$$= O\left(n^{\log_2 3}\right)$$

(Answer)

Day :

Time : Date : / /

(B)

$$\frac{n}{4^k} = 1 \Rightarrow 4^k = n$$
$$\Rightarrow k = \log_4 n$$

$$T(n) = 8T\left(\frac{n}{4}\right) + O(n\sqrt{n})$$

$$= 8^2 T\left(\frac{n}{4^2}\right) + \frac{8n}{4}\sqrt{\frac{n}{4}} + n\sqrt{n}$$

$$= 8^3 T\left(\frac{n}{4^3}\right) + \frac{8^2 n}{4^2}\sqrt{\frac{n}{4^2}} + \frac{8n}{4} \times \sqrt{\frac{n}{4}} + n\sqrt{n}$$

$$\vdots$$

$$= 8^k T\left(\frac{n}{4^k}\right) + \left(\frac{8^0 n}{4^0} \times \sqrt{\frac{n}{4^0}}\right) + \left(\frac{8^1 n}{4^1} \times \sqrt{\frac{n}{4^1}}\right) + \left(\frac{8^2 n}{4^2} \times \sqrt{\frac{n}{4^2}}\right) + \cdots + \left(\frac{8^{k-1} n}{4^{k-1}} \times \sqrt{\frac{n}{4^{k-1}}}\right)$$

$$= 8^{\log_4 n} + n\sqrt{n}\left[\frac{8^0}{4^0\sqrt{4^0}} + \frac{8^1}{4^1\sqrt{4^1}} + \frac{8^2}{4^2\sqrt{4^2}} + \cdots + \frac{8^{k-1}}{4^{k-1} \cdot \sqrt{4^{k-1}}}\right]$$

$$= n^{1.5} + n\sqrt{n} \times \left[(K) \times 1\right]$$

$$\left[\begin{array}{l}\text{As common ratio, } r = 1 \\ \text{Hence, } S_n = n \times a\end{array}\right]$$

$$= n^{1.5} + n\sqrt{n} \times \log_4 n$$

$$= O(n\sqrt{n} \times \log n)$$

(Ans)

Solutions of Question-04

(i)

```
Algorithm findSplitIndex(arr){
    n ← arr.size
    low ← 0
    high ← n-1
    splitIndex ← n

    while(low <= high){
        mid ← (low+high)//2
        if (arr[mid] >= arr[n-1]){
            splitIndex ← mid
            high ← mid-1
        }
        else{
            low ← mid+1
        }
    }
    return splitIndex
}
```

(ii)

Here, i'm using a modified version of Binary Search where the search space gets divided by half at each iteration.

Hence, time complexity: $O(\log n)$

Scenario: 1

Solutions of Question-05

ⓐ

The most efficient algorithm for maintaining a frequently appended and sorted array is Insertion Sort.

Explanation:- Insertion Sort is the best choice here because of its efficiency when dealing with an already mostly sorted array. Here is also the same case, we're appending a new value at the end of a sorted array. Hence, we just need to insert the last value in its right place to keep the array sorted again. And for this in worst case, we've to disturb all of the previous n-1 elements. Hence, worst-case time complexity is simply $O(n)$.

ⓑ

Before appending: [1, 4, 7, 8, 11]

After appending: [1, 4, 7, 8, 11, 3]

Simulation:-

[1, 4, 7, 8, 11, 3] $\boxed{\text{Saved Value} = 3}$

[1, 4, 7, 8, 11, 11] $\boxed{\text{Shifting untill find the correct inserting position}}$

[1, 4, 7, 8, 8, 11]

[1, 4, 7, 7, 8, 11]

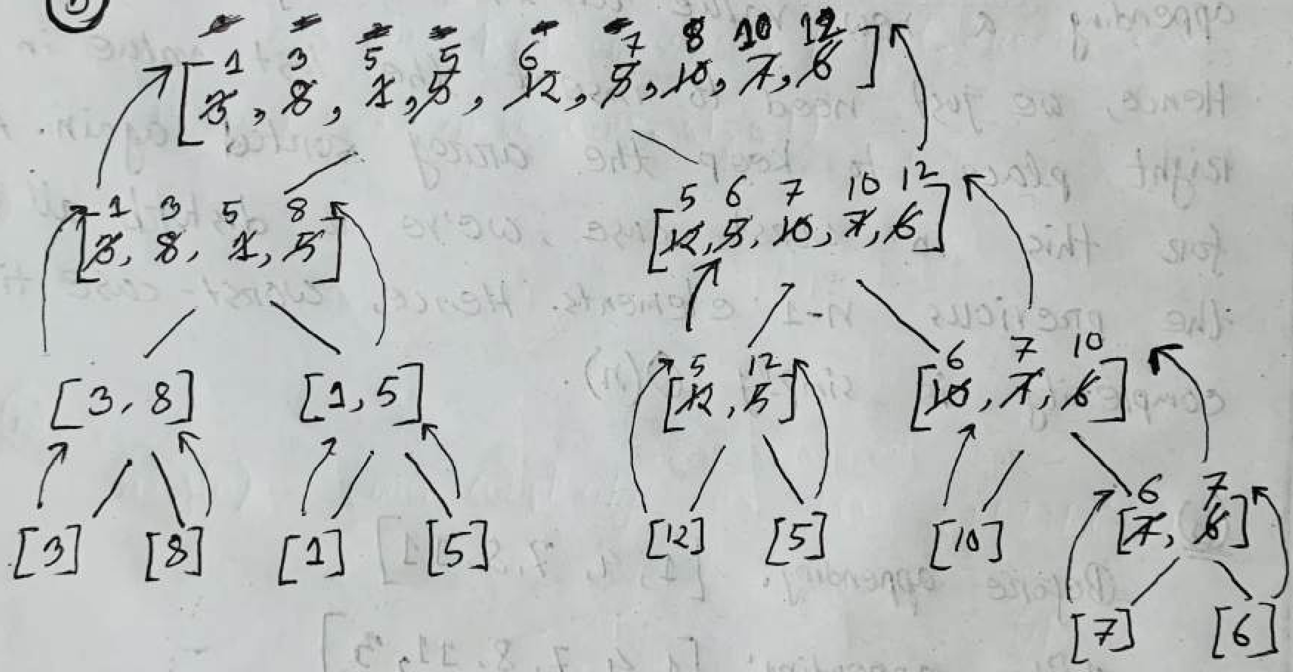
[1, 4, 4, 7, 8, 11] => [1, 3, 4, 7, 8, 11].

Scenario 2

(a) Merge Sort.

<u>Explanation:</u> We can't perform any task that has higher than O(nlogn) time complexity. In the worst-case only merge sort has O(nlogn) time complexity. So, merge sort is the best choice in this scenario.

(b)



Time : Date : / /

(a)

Solutions of Questions-06

```
Algorithm maxSubarray(arr, P, r){
    if (P==r){
        return (arr[P], P, r)
    }
    q = (P+r)//2
    L= maxSubarray(arr, P, q)
    R= maxSubarray(arr, q+1, r)
    c= crossingSubarray (arr, P, q, r)
    if (C[0] >= L[0] and C[0]>= R[0]){
        return c
    elif (L[0]>= R[0] and L[0]>= C[0]){
        return L
    }
    else {
        return R
    }
}

Algorithm crossingSubarray(A, P, q, r){
    sI= P
    eI= q
    left-cross_sum <- -∞
    crrentSum <- 0
    for(i<-q to P){
        currentSum += A[i]
        if(currentSum > left-cross_sum){
            left_cross-sum <- currentSum
            sI=i
        }
    }
```

```
right-cross-sum ← -∞
curSum ← 0
for(i ← q+1 to r){
    curSum += A[i]
    if (curSum > right-cross-sum){
        right-cross-sum = curSum
        eI = i
    }
}
max-cross-sum = left-cross-sum + right-cross-sum
return (max-cross-sum, sI, eI)
}
```

(b)

$n\log(n)$ = time complexity as $T(n) = 2T(\frac{n}{2}) + O(n)$

(Ans:)

Where's the simulation of (a)?

- Ha..Ha.. Sob ami kore dile tumi ki korba, ota basai nije practice kore nio.