

GATE PSUs

State Engg. Exams

MADE EASY
WORKBOOK 2024



**Detailed Explanations of
Try Yourself *Questions***

Computer Science & IT
Algorithms



1

Loops Analysis, Asymptotic Notations, Recursive Algorithm and Methods to Solve Recurrence Relations



Detailed Explanation of Try Yourself Questions

T1 : Solution

(a)

$$f(n) = \Omega(n), g(n) = O(n), h(n) = \Theta(n)$$

Then $[f(n) \cdot g(n)] + h(n)$

$$f(n) = \Omega(n)$$

i.e. $f(n)$ should be anything greater than or equal to 'n' lets take n .

$$g(n) = O(n)$$

i.e. $g(n)$ should be less than or equal to 'n' lets take n .

$$h(n) = \Theta(n)$$

i.e. $h(n)$ should be equal to n .

So $[f(n) \cdot g(n)] + h(n)$

$$[n \cdot n] + n$$

$$= \Theta n^2 + \Theta n = \Omega(n)$$

Here we only comment about lower bound. Upper bound depend an the $g(n)$ value i.e. $n^2, n^3, n^4 \dots$ etc.

T2 : Solution

The increasing order of given five fuctions are: $f_4 < f_2 < f_5 < f_1 < f_3$.

T3 : Solution**(b)**

```
find (int n)
{
    if (n < 2) then return;
    else
    {
        sum = 0;
        for (i = 1; i ≤ 4; i++) find(n/2);    → O(log n)
        for (i = 1; i ≤ n * n; i++)        → O(n2)
        sum = sum + 1;
    }
}
```

Since first for loop run $4 \log n$ times for which second for loop run n^2 times for every value of $(4 \log n)$.
So total time complexity = $O(4 \log n \times n^2) = O(n^2 \log n)$.

■■■■

2

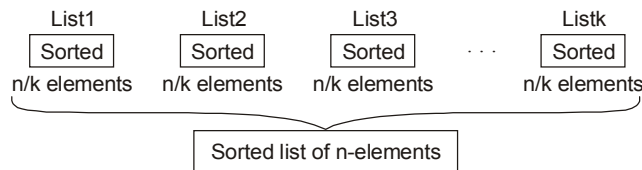
Divide and Conquer



Detailed Explanation of Try Yourself Questions

T1 : Solution

$O(n \log k)$



- (i) Remove the smallest element from each list and build min heap with k -elements $\Rightarrow O(k)$.
- (ii) Extract the minimum elements from this heap that will be the next smallest in the resulted list $\Rightarrow O(\log k)$.
- (iii) Remove the elements from original list where we have extracted next smallest element and insert into the heap $\Rightarrow O(\log k)$.

Repeat step2 and step3 until all elements are in the resulted list
 $= O(k) + [O(\log k) + O(\log k)] * O(n)$
 $= O(n \log k)$

T2 : Solution

$O(mk + m \log (m/k))$

Insertion sort takes $O(k^2)$ time per k -element list in worst case. Therefore sorting n/k lists of k -element each take $O(k^2 n/k) = O(nk)$ time in worst case.



3

Binary Trees, Binary Heaps and Greedy Algorithms



Detailed Explanation of Try Yourself Questions

T1 : Solution

$O(E + V)$

In adjacency list representation of directed graph to find the out degree of each vertex will take $O(E+V)$ time in worst case i.e. for an element we have to search n time.

T2 : Solution

$O(V + E)$

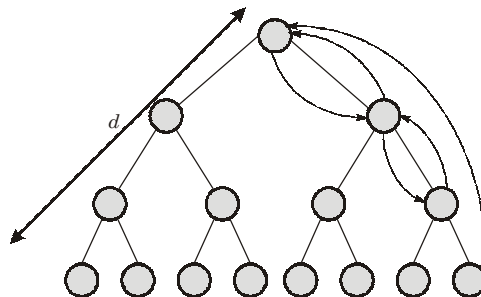
In adjacency matrix representation of directed graph to find universal sink will take $O(V + E)$ time i.e. for every entry in adjacency matrix we have to check n time.

T3 : Solution

(b)

Heap is implemented using array. If ' i ' is parent element then ' $2i$ ' is left child and ' $2i + 1$ ' is right child. So if an element is delete from last level of the heap then it will take $O(1)$ time. Since element can be deleted from any level of heap tree in worst case root element is deleted then at every level one element is exchange.

Example:



Minimum $O(d)$ time will take if a element is deleted in heap tree but not $O(1)$.

T5 : Solution **$O(n)$**

Sorting the array using **binary search tree** will take $O(n)$ time i.e. inorder sequence.

Sorting the array using **min heap tree** will take $O(n \log n)$ time i.e. $O(n)$ time to build and $\log n$ time to get every minimum element. So $O(n) + O(n \log n) = O(n \log n)$.

In the giving question **binary search tree is better than min heap tree by $O(n)$ time.**

T6 : Solution**(b)**

```
max-heapify (int A[ ], int n, int i)
```

```
{
```

```
    int P, m;
```

```
    P = i;
```

```
    while (2P ≤ n) for checking whether left child is present.
```

```
    {
```

```
        if (2P + 1 ≤ n && A[2P + 1] > A[2P]) for checking if right child is present or not and finding between left and right child which is greater.
```

```
        n = 2P + 1;
```

```
        else m = 2P;
```

```
        whichever is greater, swap that child with its parent
```

```
        if (A[P] < A[m])
```

```
        {
```

```
            Swap (A[P], A[m]);
```

```
            P = m;
```

```
        }
```

```
        else
```

```
        return;
```

```
    }
```

```
}
```

T7 : Solution**(7)**

Kruskal's algorithm: AE, AG, AB, CE, FI, FH, CD, CF

Prim's algorithm: CF, CE, EA, AG, AB, FI, FH, CD (since we have to find maximum difference so start with edge CF)

$$\max |(e_{p_i})_{\text{Prim's}} - (e_{p_i})_{\text{Kruskal's}}| = |8 - 1| = 7$$

T8 : Solution**[O(m)]****Kruskal's algorithm:**(i) Sorting $\Rightarrow O(m \log m)$ (ii) Union $\Rightarrow O(n \log n)$ (iii) Find $\Rightarrow O(m \log n)$ \Rightarrow Running time = $O(m \log m)$

Now edges are already sorted.

 \therefore Running time = $O(m)$ **T9 : Solution** **$O(|V|)$** Let e be an edge of G but not in T (i) Run DFS on $T \cup \{e\}$

(ii) Find cycle

(iii) Trace back edges and find edge e' thus has maximum weight.(iv) Remove e' from $T \cup \{e\}$ to get MSTIn $T \cup \{e\} \Rightarrow$ Number of edges = Number of vertices \therefore Running time of DFS = $O(|V| + |E|) = O(|V|)$ 

4

Sorting Algorithms, Graph Traversals and Dynamic Programming



Detailed Explanation of Try Yourself Questions

T1 : Solution

(c)

```
Insertion-sort (A)
{
  for j ← 2 to length (A)
  {
    key ← A[j]
    i =  $\boxed{j-1}$ 
    while (i > 0 && A[i] > key)
    {
      A[i + 1] ← A[i]
      i =  $\boxed{i-1}$ 
    }
    A[i + 1] ← key;
  }
}
```

T2 : Solution

$O(n)$

The length of array A is n which stores the integers. We need two additional arrays $B[0\dots k]$ and $C[0\dots k]$. Initialize B and C with 0. It requires $O(k)$. For each element of A increment $B[A[i]]$. It will take $O(n)$ time $B[j]$ contain the number of elements of A having value j .

Do $C[1] = B[1]$ and for each element i of array C do

$$C[i] = B[i] + [i - 1]$$

It will take $O(k)$ for getting answer compute $C[b] - C[a] + B[a]$.

The preprocessing time takes = $O(n)$

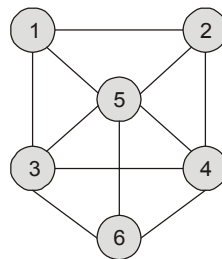
T3 : Solution

G is the connected graph with $n - 1$ edges \Rightarrow G don't have any cycle.
Therefore statement 1 and 3 implies 2.

T4 : Solution

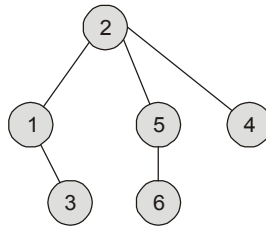
(d)

Lets take a undirected graph



(G) Graph

and 2 is source, after performing BFS on graph we obtain the following tree.



(T) Tree

Now missing edges are: 1 to 5, 4 to 5, 4 to 6, 3 to 6, 3 to 5, 3 to 4 for 1 to 5 = $d(u) - d(v)$
(Distance from 2 to 1) – (Distance from 2 to 5)

$$= 1 - 1 = 0$$

$$\text{for 4 to 5} = d(u) - d(v)$$

$$= 1 - 1 = 0$$

$$\text{for 4 to 6} = d(u) - d(v)$$

$$= 1 - 2$$

$$= -1 \text{ or } 1$$

$$\text{for 3 to 6} = d(u) - d(v)$$

$$\begin{aligned}
 &= 2 - 2 = 0 && \text{So 2 is not possible} \\
 \text{for 3 to 5} &= d(u) - d(v) && \text{So answer is (d)} \\
 &= 2 - 1 \\
 &= 1 \text{ or } -1 \\
 \text{for 3 to 4} &= d(u) - d(v) \\
 &= 2 - 1 \\
 &= -1 \text{ or } 1
 \end{aligned}$$

T5 : Solution

(64)

In question restriction on BST is height should be '6'. So we need 7 levels (given that root at height '0'). In creation of BST we have to use all element without repetition.

At **1 level** = We have 2 choice i.e., either take 1 or 7.

At **2 level** = We have 2 choice for root 1 and 7 each. If 1 is root then 2 choice will be 6 and 2. If 7 is root then 2 choice will be 1 and 6.

At **3 level** = If we take 1 at root, 6 at 2nd level than we have 2 choice i.e., 5 and 2 at 3rd level.

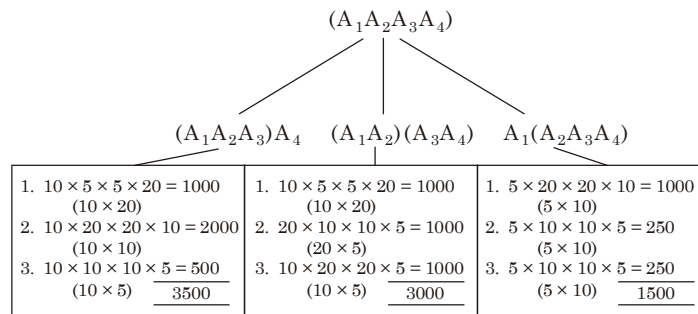
If we take 1 at root, 2 at 2nd level than we have 2 choice i.e., 6 and 3 at 3rd level. Similarly if we take 7 as root element.

So till 6th level, we have two choice at every level and for last level we left with only 1 element. So number of BST with height 6

$$= 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 1 = 2^6 = 64$$

T7 : Solution

(1500)



The minimum number of multiplication required using basic matrix multiplication method will be 1500.

