Name - Sanib Biswas.
Regd NO - 2341002219
Serial NO - 05
Section - 23412K1
Branch - CSE

AD ASSIGNMENT - 1
(Date - 11/11/24)

S.L. NO - 1 Q-1, Q-2, Q-3, Q-4, Q-5 (Chapter - 2)

1. (a) Double the input size

(i) n^2

Old: n^2

New: $(2n)^2 = 4n^2$

Ratio : $\frac{4n^2}{n^2} = 4$

($\therefore$ 4 times slower)

(ii) n^3

Old: n^3

New: $(2n)^3 = 8n^3$

Ratio: $\frac{8n^3}{n^3} = 8$

($\therefore$ 8 times slower)

(iii) $100n^2$

Old: $100n^2$

New: $100(2n)^2 = 400n^2$

Ratio: $\frac{400n^2}{100n^2} = 4$

($\therefore$ 4 times slower)

(iv) $n\log n$

Old: $n\log n$

New: $2n\log(2n) = 2n(\log n + \log 2)$

$= 2n\log n + 2n\log 2$

Ratio: $\frac{2n\log n + 2n\log 2}{n\log n}$

$= 2 + \frac{2\log 2}{\log n}$

As n grows, this approaches 2.

($\therefore$ Approaches 2 times slower)

(v) 2^n Old: 2^n New: $2^{2n} = (2^n)^2$

Ratio : $\frac{(2^n)^2}{2^n} = 2^n$

($\therefore$ 2^n times slower)

(b) increase the input size by one

(i) n^2

Old: n^2 New: $(n+1)^2$ Ratio: $\frac{n^2+2n+1}{n^2} = 1 + \frac{2n+1}{n^2}$

≈ 1 (for large n)

($\therefore$ Approx. 1 time slower)

(ii) n^3

Old: n^3 New: $(n+1)^3$ Ratio: $\frac{n^3+3n^2+3n+1}{n^3} = 1 + \frac{3n^2+3n+1}{n^3}$

≈ 1 (for large n)

($\therefore$ Approx. 1 time slower)

(iii) $100n^2$

Old: $100n^2$ New: $100(n+1)^2$ Ratio: $\frac{100(n^2+2n+1)}{100n^2} = 1 + \frac{2n+1}{n^2}$

≈ 1 (for large n)

$\therefore$ (Approx. 1 time slower)

(iv) $n \log n$

Old: $n \log n$ New: $(n+1)\log(n+1)$

Ratio: $\frac{(n+1)\log(n+1)}{n \log n} \approx 1$ (for large n)

$\therefore$ (Approx. 1 time slower)

(v) 2^n

Old: 2^n New: $2^{n+1} = 2^n \cdot 2$

Ratio = $\frac{2^n \cdot 2}{2^n} = 2$

$\therefore$ (2 times slower)

2. Given, operations performed per second $= 10^{10}$
Max^m time limit $= 1\,hr = 3600s$
Total operations $= 3600 \times 10^{10} = 3.6 \times 10^{13}$.

(a) $\boxed{n^2}$

$n^2 \le 3.6 \times 10^{13}$ $\quad n \le \sqrt{3.6 \times 10^{13}}$

$n \approx 6 \times 10^6$

So, largest n for n^2 is approx 6×10^6.

(b) $\boxed{n^3}$

$n^3 \le 3.6 \times 10^{13}$

$n \le \sqrt[3]{3.6 \times 10^{13}} \approx 3.3 \times 10^4$

So, largest n for n^3 is approx. 3.3×10^4.

(c) $\boxed{100n^2}$

$100n^2 \le 3.6 \times 10^{13}$

$n^2 \le 3.6 \times 10^{11}$

$n \approx 6 \times 10^5$

So, largest n for $100n^2$ is approx. 6×10^5.

(d) $\boxed{n \log n}$

$n \log n \le 3.6 \times 10^{13}$

$n \approx 10^{12}$

$\therefore 10^{12} \log_2 10^{12} \approx 3.98 \times 10^{13}$

So, by trial method largest n for $n \log n$ is approx. 10^{12}.

(e) $\boxed{2^n}$

$2^n \le 3.6 \times 10^{13}$

$\Rightarrow n \log_2 2 \le \log_2 (3.6 \times 10^{13}) \Rightarrow n \le 45$

So, largest n for 2^n is approx. 45.

(f) $\boxed{2^{2^n}}$

$2^{2^n} \le 3.6 \times 10^{13}$

$\Rightarrow 2n \log_2 2 \le \log_2 (3.6 \times 10^{13}) \approx 45$

$\Rightarrow n \le 22.5$

So, largest n for 2^{2^n} is approx 22.

3. $f_1(n) = n^{2.5}$, $O(n^{2.5}) = O(n^{5/2})$
$f_2(n) = \sqrt{2n}$, $O(\sqrt{2n}) = O(n^{1/2})$
$f_3(n) = n+10$, $O(n)$
$f_4(n) = 10^n$, $O(10^n)$
$f_5(n) = 100^n$, $O(10^{2n})$
$f_6(n) = n^2 \log n$, $O(n^2 \log n)$

Ascending order:
$f_2(n) < f_3(n) < f_1(n) < f_6(n) < f_4(n) < f_5(n)$

4. $-g_1(n) = 2^{\sqrt{\log n}}$, $O(2^{\sqrt{\log n}})$
$-g_2(n) = 2^n$, $O(2^n)$
$-g_3(n) = n(\log n)^3$, $O(n(\log n)^3)$
$-g_4(n) = n^{4/3}$, $O(n^{4/3})$
$-g_5(n) = n^{\log n}$, $O(n^{\log n})$
$-g_6(n) = 2^{2^n}$, $O(2^{2^n})$
$-g_7(n) = 2^{n^2}$, $O(2^{2n})$

Ascending order: $g_1(n) < g_3(n) < g_4(n) < g_5(n) < g_2(n) < g_7(n) < g_6(n)$.

5. Given, $f(n) = O(g(n))$
$\Rightarrow f(n) \leq c \cdot g(n)$, $n \geq n_0$

(a) $\log_2 f(n)$ is $O(\log_2 g(n))$

Proof :- Given , $f(n) \leq c \cdot g(n)$, $n > n_0$
$\Rightarrow \log_2 f(n) \leq \log_2 (c \cdot g(n)) = \log_2 c + \log_2 g(n)$
$\Rightarrow \log_2 f(n) \leq \log_2 g(n) +$ constant
$\Rightarrow \log_2 f(n) = O(\log_2 g(n))$.
$\therefore$ Statement is true.

(b) $2^{f(n)}$ is $O(2^{g(n)})$

Counter example :- Let $f(n) = n$, $g(n) = 2n$
$\therefore f(n) \leq g(n)$
$\therefore 2^{f(n)} = 2^n$ and $2^{g(n)} = 2^{2n} = (2^n)^2$, so $2^{f(n)}$ is $O(\sqrt{2^{g(n)}})$.

∴ Statement is false.

(c) $f(n)^2$ is $O(g(n)^2)$.

Proof :- Given, $f(n) \le c \cdot g(n)$

$\Rightarrow f(n)^2 \le (c \cdot g(n))^2 = c^2 \cdot g(n)^2$

$\Rightarrow f(n)^2 = O(g(n)^2)$.

∴ Statement is true.

S.L. NO-2 Let f and g be two fns that take non-negative values. Prove $g = \Omega(f)$ if $f = O(g)$.

Given, $f(n) \le c \cdot g(n)$ for $n \ge n_0$ and c.

$\Rightarrow \frac{1}{c} f(n) \le g(n)$

$\Rightarrow g(n) \ge \frac{1}{c} f(n)$

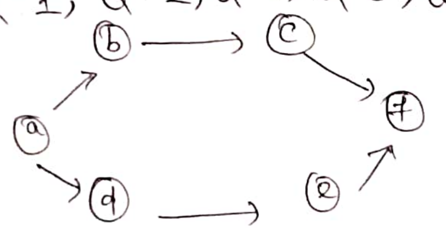
$\Rightarrow g(n) = \Omega f(n)$, $n \ge n_0$ and constant $= \frac{1}{c}$.

$\Rightarrow g = \Omega(f)$.

(proved).

S.L. NO-3 Q-1, Q-2, Q-3, Q-5, Q-6 (chapter-3)

1.



Soln:- 6 Possible Topological ordering :

1. a, b, c, d, e, f
2. a, b, d, c, e, f
3. a, b, d, e, c, f
4. a, d, e, b, c, f
5. a, d, b, e, c, f
6. a, d, b, c, e, f

2. Give an algo. to detect whether a given undirected graph contains a cycle. output = 1 if yes. Time for running = O(m+n).

Soln:- Let assume G is connected or algo. works separately with connected components. (after computing them in O(m+n) time).

Cycle DetectionBFS (G, s)

1. If G is connected : find all connected component in O(m+n) time.
2. Initialize BFS: start BFS from arbitrary node s.

3. Construct BFS tree T : for each edge $e = (u, v)$ in G:
if e is part of traversal, add it to T.

4. If G = T, it contains no cycle.

5. If $e = (v, w) \in G$ and not in T
return 1.

Time complexity: $O(m+n)$, since BFS traversal and building the tree T both take linear time.

3. TO of a DAG finds node with no incoming edges and deletes it. Given a graph may or may not be a DAG. Extend TO algo., it outputs one of two things:
(a) a TO
(b) a cycle in G ⇒ G is not a DAG.
TC should be $O(m+n)$.
[m → edge
n → node]

Soln,-
1. Find node v with no incoming edges and order it first.
2. Delete v
3. Recursively compute a TO of G - v
4. If in some iteration, it transpires every node has least one incoming edge.
5. G contains a cycle.

5. Show by induction that in any bin. tree no. of nodes with two children is exactly one less than no. of leaves.

Soln,-
Proof
By induction method,
Let n = no. of nodes in T
n_0 = no. of leaves in T
n_2 = no. of leaves with 2 children

Basic step
Let T has only one single node. This node is only leaf and no node with two children.

Inductive step
Let T be an arbitrary binary tree with more than one node and v be a leaf.

Since T has more than one node, v is not root and has a parent u.

Let $T' = T - \{u\}$.

Case-I: If u has no other child in T, it is a leaf in T'.

$\therefore n_0(T') = n_0(T)$ and $n_2(T') = n_2(T')$

Case-II: If u has another child in T, it is not a leaf in T'.

$n_0(T) = n_0(T) - 1$ and $n_2(T') = n_2(T') - 1$.

Proved

6. We have connected graph $G = (V, E)$ and $u \in V$. We compute DFS rooted at u and obtain tree T that includes all nodes of G. Then we compute a BFS at u and obtain same T. Prove $G = T$.

Soln:- Suppose G has an edge $e = \{a, b\} \notin T$.

Since T is a DFS tree, one of the two ends must be an ancestor of other.

Again, Since T is a BFS tree, dist. of two nodes from u in T can differ by at most 1.

Let say, a is ancestor of b.

$\Rightarrow$ Dist. from u to b in T is at most one greater than the dist. from u to a, then a must in fact be the direct parent of b in T.

$\Rightarrow \{a, b\} \in T$.

$\Rightarrow G = T$.

Proved

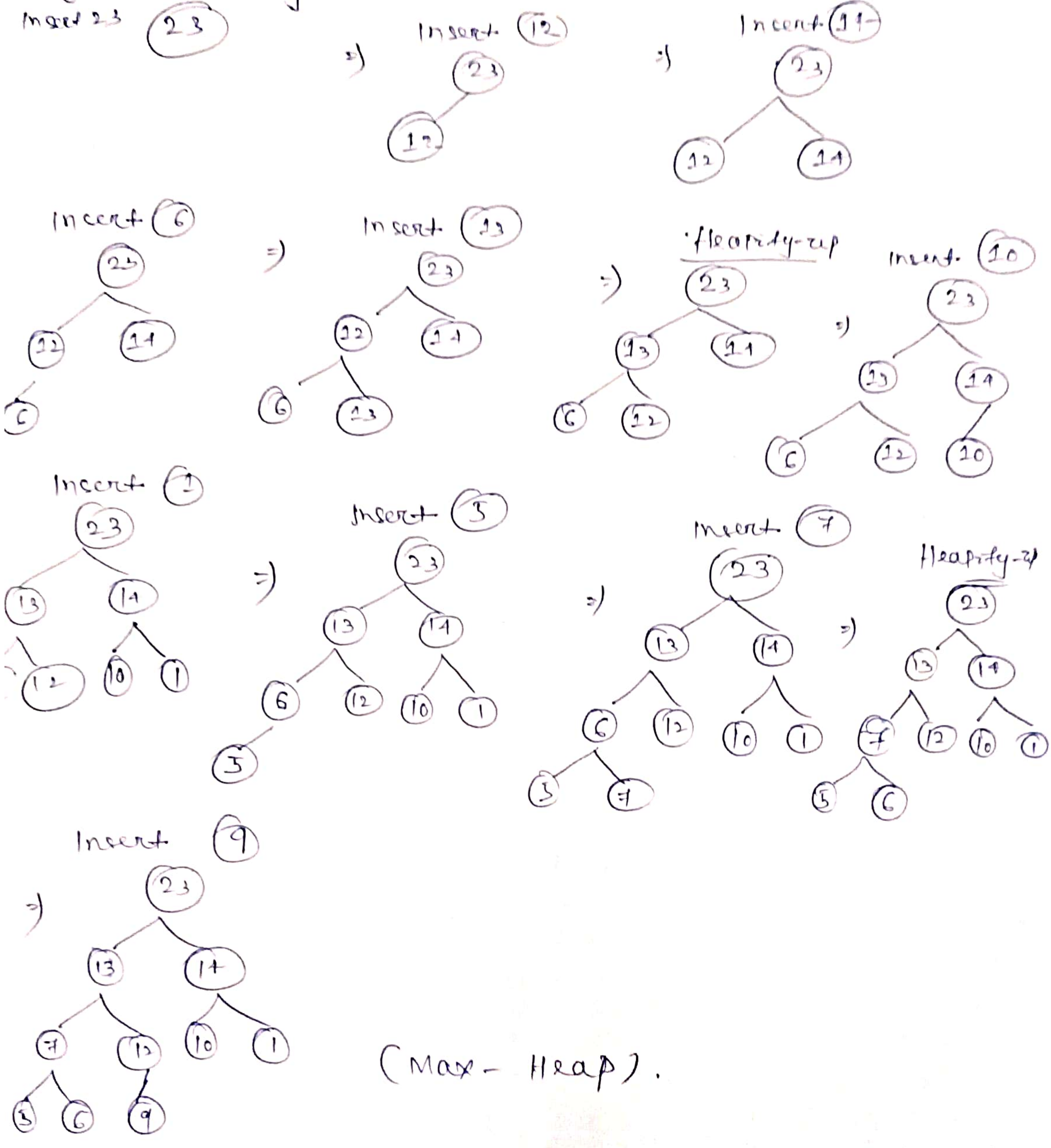
S.L. NO-4

What are the minm and maxm no. of elements in a heap of height h? Is the array with values $\langle 23, 12, 14, 6, 13, 10, 1, 3, 7, 9 \rangle$ a max-heap, if not build the max-heap.

Soln:- Minm no. of elements $= 2^h$

Maxm no. of elements $= 2^{h+1} - 1$.

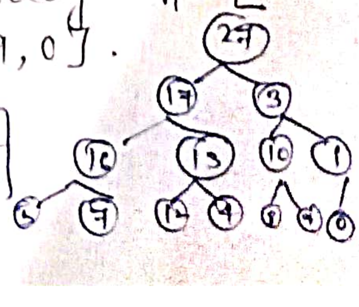
Given array is not a max heap.



(Max-Heap).

SL NO-5 Illustrate the operation of MAX-HEAPIFY(A, 3) on the array A = [27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0].

Soln :- $\boxed{16 > 5 \text{ and } 16 > 7}$



MAX-HEAPIFY (A, 3)

A[3] = 16

Here 16 maintains max-heap property, so array remains same.

S.L.NO-6 Write pseudocode for the procedures HEAP-MINIMUM, HEAP-EXTRACT-MIN, HEAP-DECREASE-KEY and MIN-HEAP-INSERT that implement a min-priority Queue with a min-heap.

Soln:- HEAP- MINIMUM

```
H. heapsize = A.length;
for (i = n/2 to 1);
    return (H, i);
```

HEAP- EXTRACTION - MIN

```
if A.heap-size < 1
   return "heap underflow";
min = A[1]
A[1] = A[A.heap size]
A.heap_size = A.heap_size - 1
MIN- HEAPIFY (A,1)
  return min;
```

HEAP- DECREASE -KEY

```
HEAP - DECREASE - KEY (A, i, key)
 if key > A[i];
return "New key > current key".
  A[i] = key
while i > 1 and A[parent(i)] > A[i]
  swap A[i] with A[parent(i)]
   i = parent (i);
```

MIN- HEAP - INSERT

```
MIN- HEAP- INSERT (A, key)
   A. heap- size = A. heap- size + 1
   int n = sc.nextInt();
   A[A.heap-size] = n;
HEAP- DECREASE- KEY (A, A.heap-size, key)
```

S.L.NO-7 Illustrate the operation of MIN-HEAP-INSERT (A, 10) on the heap A = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1, 7].

Soln.- Initial Heap: A = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]

Step-1 Increase heap-size by one and ~~add~~ Insert 10 at the end.

Step-2

$A[12] = 10$

$\text{Parent} = \frac{i-1}{2} = \frac{12-1}{2} = \lfloor 5.5 \rfloor = 5$

$A[5] = 8$

Here $10 > 8$, no swap needed.

Step-3

Final Heap:

A = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1, 10]

S.L.No-8 Prove that $\lg n = O(\sqrt{n})$, however $\sqrt{n} \neq O(\lg n)$.

Soln.- To show $\log n = O(\sqrt{n})$

To find $c > 0$ and n_0 such that $n \geq n_0$.

$$\log n \leq c \cdot \sqrt{n}$$

For large n,

$$\log n \leq c \cdot \sqrt{n}$$

Example calculation

Let $n = 1\,000\,000$

$\log_2 n \approx 20$, $\sqrt{n} = 1000$

So, $20 \leq 1000 \Rightarrow \log n < \sqrt{n}$ (for large n)

Therefore, exists c $(c = 1)$ and n_0 such that

$\log n \leq c \cdot \sqrt{n}$ $\forall\, n \geq n_0$.

$\therefore$ $\log n = O(\sqrt{n})$. (Proved).

S.L.NO-9 Find TC and SC.

```
function (int n) {
    if (n==1)
        return 1;
else
    function(n/3); function(n/3); function(n/3);
    for (int i = 1; i <= n; i++)
        n = n + 1; }
```

Soln. - TC =

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

Here $a = 3$, $b = 3$, $f(n) = O(n)$

Applying Master's theorem,

$T(n) = n^{\log_3 3}[U(n)] = n[U(n)]$

$U(n)$ depends on $h(n)$

$h(n) = \frac{f(n)}{n^{\log_3 3}} = \frac{n}{n} = 1$

Here, $U(n) = \frac{(\log_2 n)^{0+1}}{0+1} = \log_2 n = \log n$

$$\boxed{\therefore T(n) = O(n \log n)}$$

SC =

Each recursive call with input size n/3 takes up space on call stack.

Depth of recursion is $O(\log_3 n) = \boxed{O(\log n)}$.

S.L.NO-10

```
Void function (int n){
    Temp = 1;
  Repeat
      for i = 1 to n
         temp = temp + 1
      n = n/2;
  Until n <= 1 }
```

Find TC and SC.

Soln. - $T(n) = n + \frac{n}{2} + \frac{n}{4} + \ldots$ $\boxed{\therefore T(n) = O(n)}$

SC: Algo. uses a few variables which require constant space. So $\boxed{SC = O(1)}$.

S.L.NO-11 Solve.

(a) $T(n) = \begin{cases} 1 & , n = 2 \\ 1 & , n = 4 \\ T(\frac{n}{2}) + 2T(\frac{n}{4}) + \Theta(n^2) & , n > 4 \end{cases}$

(n is power of 2).

Applying master's theorem,

Soln:- For $T(\frac{n}{2})$ $a = 1, b = 2$

$T_1(n) = n^{\log_2 1} = n^0 = 1$

For $2T(\frac{n}{4})$ $a = 2, b = 4$

$T_2(n) = n^{\log_4 2} = n^{1/2}$

$f(n) = \Theta(n^2)$

Growth rate of $f(n)$ is faster than $T_1(n)$ and $T_2(n)$

So, $\boxed{T(n) = O(n^2) \quad , n > 4}$

(b) $T(n) = T(\frac{n}{5}) + T(\frac{4n}{5}) + O(n)$

Applying master's theorem,

Soln:- For $T(\frac{n}{5})$ $a = 1, b = 5$

$T_1(n) = n^{\log_5 1} = n^0 = 1$

For $T(\frac{4n}{5})$ $a = 1, b = 5/4$

$T_2(n) = n^{\log_{5/4} 1} = n^0 = 1$

$f(n) = O(n)$

Growth rate of $f(n)$ is faster than $T_1(n)$ and $T_2(n)$

So, $\boxed{T(n) = O(n)}$

(c) $T(n) = 3T(\frac{n}{2}) + cn^2$

Soln,- Applying master's theorem,

$a = 3$, $b = 2$, $f(n) = n^2$

$T(n) = n^{\log_2 3} \cdot U(n) = n^{1.585} \cdot U(n) \approx n^2 \cdot U(n)$

$U(n)$ depends on $h(n)$

$h(n) = \frac{f(n)}{n^{\log_2 3}} = \frac{n^2}{n^{1.585}} = n^{0.415} \approx O(1)$

$$\boxed{\therefore T(n) = O(n^2)}$$

(d) $T(n) = 4T(n/2) + cn^2$

Soln,- App. master's theorem,

$a = 4$, $b = 2$, $f(n) = n^2$

$T(n) = n^{\log_2 4} \cdot U(n) = n^2 \cdot U(n)$

$U(n)$ depends on $h(n)$

$h(n) = \frac{n^2}{n^2} = 1 = \frac{(\log_2 n)^{0+1}}{0+1} = \log n$

$$\boxed{\therefore T(n) = O(n^2 \log n)}$$

(e) $T(n) = 3T(\frac{n}{4}) + n\log n$

Soln :- App. master's theorem,

$a = 3$, $b = 4$, $f(n) = n\log n$

$T(n) = n^{\log_4 3} \cdot U(n) = n^{0.792} \cdot U(n) \approx n \cdot U(n)$

$U(n)$ depends on $h(n)$

$h(n) = \frac{n\log n}{n} = \log n$

$$\boxed{\therefore T(n) = O(n\log n)}$$

(f) $T(n) = 3T(n/3) + \sqrt{n}$

Soln,- appl. master's theorem

$a = 3$, $b = 3$, $f(n) = \sqrt{n}$

$T(n) = n^{\log_3 3} \cdot U(n) = n \cdot U(n)$

$U(n)$ depends on $h(n)$

$h(n) = \frac{\sqrt{n}}{n} = n^{-1/2} \approx O(1)$

$\boxed{\therefore T(n) = O(n)}$

(g) $T(n) = \sqrt{n}\,T(\sqrt{n}) + \log n$

Soln,- ~~appl. master's theorem~~

Let $n = 2^m \Rightarrow \sqrt{n} = 2^{m/2}$

$T(2^m) = 2^{m/2}\,T(2^{m/2}) + \log_2 2^m$

Let $T(2^m) = S(m)$

$\therefore S(m) = 2^{m/2}\,S(\frac{m}{2}) + m$

By substitution method,

$S(m/2) = 2^{m/4}\,S(m/4) + m/2$

$S(m) = 2^{m/2}\,[2^{m/4}\,S(m/4) + m/2] + m$

$= 2^{3m/4} \cdot S(m/4) + 2^{m/2-1} \cdot m + m$

$= 2^{m(1-k/2)}\,S(m/2^k)$

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

$T(n) = O(\log n \cdot \sqrt{n}) = O(\sqrt{n}\log n)$.

SI.NO 12. Linear_Search (A, n, el)

1. for i = 1 to n do
2. if A[i] = el then
3. return i
4. return NIL

Write recursive version, recurrence relation. Compare TC and SC with iterative version.

Soln. Recursive_Linear_Search (A, n, i, el)

1. if i > n
2. return NIL
3. if A[i] = el then
4. return i
5. return Recursive_Linear_Search (A, n, i+1, el)

$$T(n) = \begin{cases} O(1) & \text{if } n = 0 \\ T(n-1) + O(1) & \text{if } n > 0 \end{cases}$$

$$\boxed{T(n) = O(n)}$$

Iterative	Recursive
TC: $O(n)$	TC: $O(n)$
SC: $O(1)$	SC: $O(n)$

SI.NO 13. Find TC and SC.

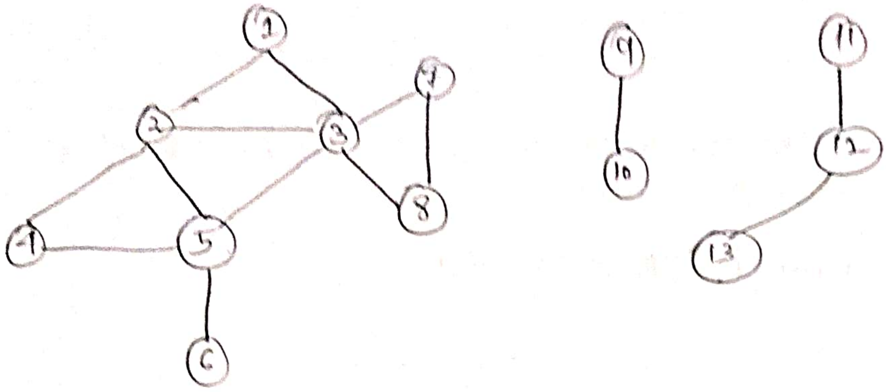
```
int function (int n) {
    if (n <= 2)
        return 1;
    else
        return (function (floor (sqrt (n))) + 1);
}
```

Soln. TC: $T(n) = O(\log(\log(n))) + O(1)$

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

SC: $O(\log(\log(n)))$

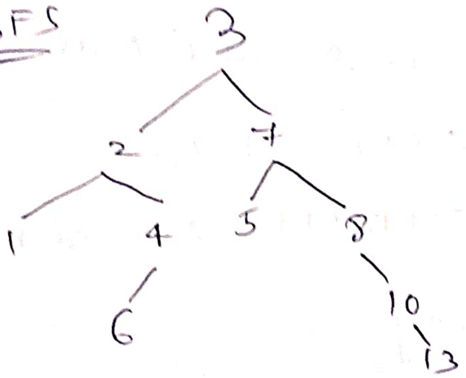
S.L No - 11 — Draw BFS and DFS tree of the following of G. Consider node 3 as root.



(G)

Soln..

BFS



DFS