Q9 → Time complexity –

```
void function (int n)
{
    for (i = 1 to n)          → n
    {
        for(j = 1 ; j <= n; j = j + i)
            printf("*");
    }
}
```

Soln →
$j = 1, 2, 3, 4$
$j = 1, 3, 6, 10$ ----

$i = 1$ n times
$i = 2$ $n/2$ times.
$i = 3$ $n/3$ ''

$$n + \frac{n}{2} + \frac{n}{3} + \cdots + 1$$

complexity = $O(n \log n)$

(10) → $n^k \, (k \geq 1)$ $c^n \, (c > 1)$

c^n ~~gro~~ grows faster than n^k.

(7) - void function (int n)

```
{
    int i, j, k, count = 0;
n/2 <-      for(i = n/2; i <= n; i++)
            {
log_2(n) <-     for(j = 1; j <= n; j = j * 2
log_2(n) <-         for(k = 1; k <= n; k = k*2 )
                        count++;
}
```

Sol^n → $\frac{n}{2} \times \log_2(n) \times \log_2(n)$

complexity $= O(n \log^2(n))$

(8) → Time complexity of - function (int n)

```
{
    if(n==1) return;
    for(i=1 to n){                  -> n
        for (j=1 to n)              -> n^2
        { printf("*");
        }
    }
    function (n-3);                 -> T(n-3)
}
```

Sol^n → $T(n) = O(n^2) + T(n-3)$ $\quad$ $T.C = O(n^2)$

with the input size (n)

(ii) Omega(Ω) :- Represents a lower bound on the growth rate. If an algorithm has a time complexity of $\Omega(n^2)$, it means the worst case time running grows at least quadratically with the input size n.

(iii) Theta (θ) :- Represents both upper & lower bound, indicating a ~~sign~~ tight bound on the growth rate of an algorithm has a time complexity of $\theta(n)$, it means the worst case running time ~~groth~~ grows linearly and there is well defined constant factor

Sol^n 2 → for(int i = 0; i < n; ~~i+=2~~ i = i * 2)

{

=

}

Taking log on both sides

$\log_2 n = (K-1)\log_2 2 \Rightarrow \log_2 n = K-1$

$K = \log_2 n + 1$

→ Time complexity = $O(\log_2 n)$

③ →

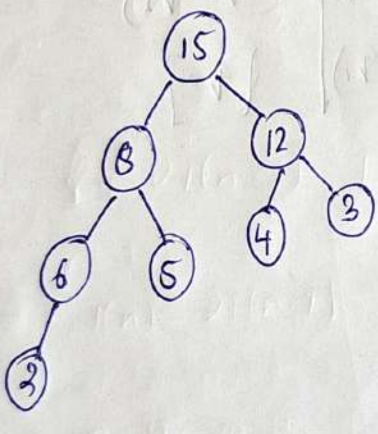
$$T(n) = \{3T(n-1)$$

DAA assignment

④ What will be the time complexity for ~~and~~ extractMin()?
given a min heap of n nodes.

extractMin - turns the minimum element for heap &

② → Given the foll^n max heap, delete an element and show the heap after every step



Sol^n →

~~③ T(n) T(n) = {3T(n-1)~~

① Asymptotic notation describe the growth rate of algorithms as their input size approaches infinity

3 commonly used notat^ns are

(i) Big O (O) :- Represents the upper bound on the growth rate
Example → if an algorithm has a time complexity of O(n), it means the worst case running time grows linearly with

(5) int i = 1, S = 1;

```
while (s <= n)
{ i++;
  S = S + i;
  printf("#");
}
```

$i = 1, 2, 3, 4, 5, \ldots$

$S = 1, 3, 6, 10, 15 \ldots n$

$$S = \frac{i(i+1)}{2}$$

$$\frac{i(i+1)}{2} \leq n$$

$$i(i+1) \leq 2n$$

$$i^2 + i - 2n \leq 0$$

$$i < \frac{-1 + \sqrt{1 + 8n}}{2}$$

complexity = $O(\sqrt{n})$

(6) void function (int n)

```
{ int i, count = 0;          — ①
  for (i = 1; i*i <= n; i++)
      count++;
}
```

$i = 1, 2, 3, 4 \quad \text{———} \quad \sqrt{n}$

$i^2 = 1, 4, 9, 16 \ldots \quad n^2$

complexity = $O(\sqrt{n})$

$\{ \; \ldots T(n-1) - 1 \quad - ①, \; n > 0 \text{ otherwise } 1 \}$

Let $n = n-1$

$T(n-1) = 2T(n-1-1) - 1 = 2T(n-2) - 1$

put $T(n-1)$ in ①

$T(n) = 4T(n-2) - 3 \quad - ②$

put $n = n-2$

$T(n-2) = 2T(n-2-1) - 1 = 2T(n-3) - 1$

put in ②

$T(n) = 4(2T(n-3) - 1) - 1 = 8T(n-3) - 4 - 1$

$(n-k) = 1$

~~$n = 1+k$~~

$k = n-1$

$T(n) = 2^{n-1} T(n-n+1) - 5 = 2^{n-1} T(1) = 5$

$= \frac{2^n}{2} = 2^n = O(2^n)$

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

(3) → $T(n) = \{3T(n-1)$ if $n>0$ otherwise $1\}$

$T(n) = 3T(n-1)$ put $n = n-1$

$T(n-1) = 3T(n-2)$ put $n = n-2$

$T(n-2) = 3T(n-3)$

And so on

$T(K) = 3^{k} T(n-K)$

put $n-K=0 \rightarrow n=K$

$T(n) = 3^n T(0)$

$T(n) = 3^n$

(4) $T(n) = \{2T(n-1) - 1$ if $n>0$ otherwise $1\}$

~~$T(n) = 2T(n-1)$~~ -1 -1