Name - Pallavi
Course - B.Tech CSE (H)
Semester - 5th
Section - B.
University Roll no - 1961121
Assignment - 1.

Ques1. What do you understand by Asymtotic notations. Define different Asymtotic notation with examples.

Answer- Asymptotic notations are used to write fastest and slowest possible running time for an algorithm. These are also referred to as 'best case' and 'worst case' respectively.

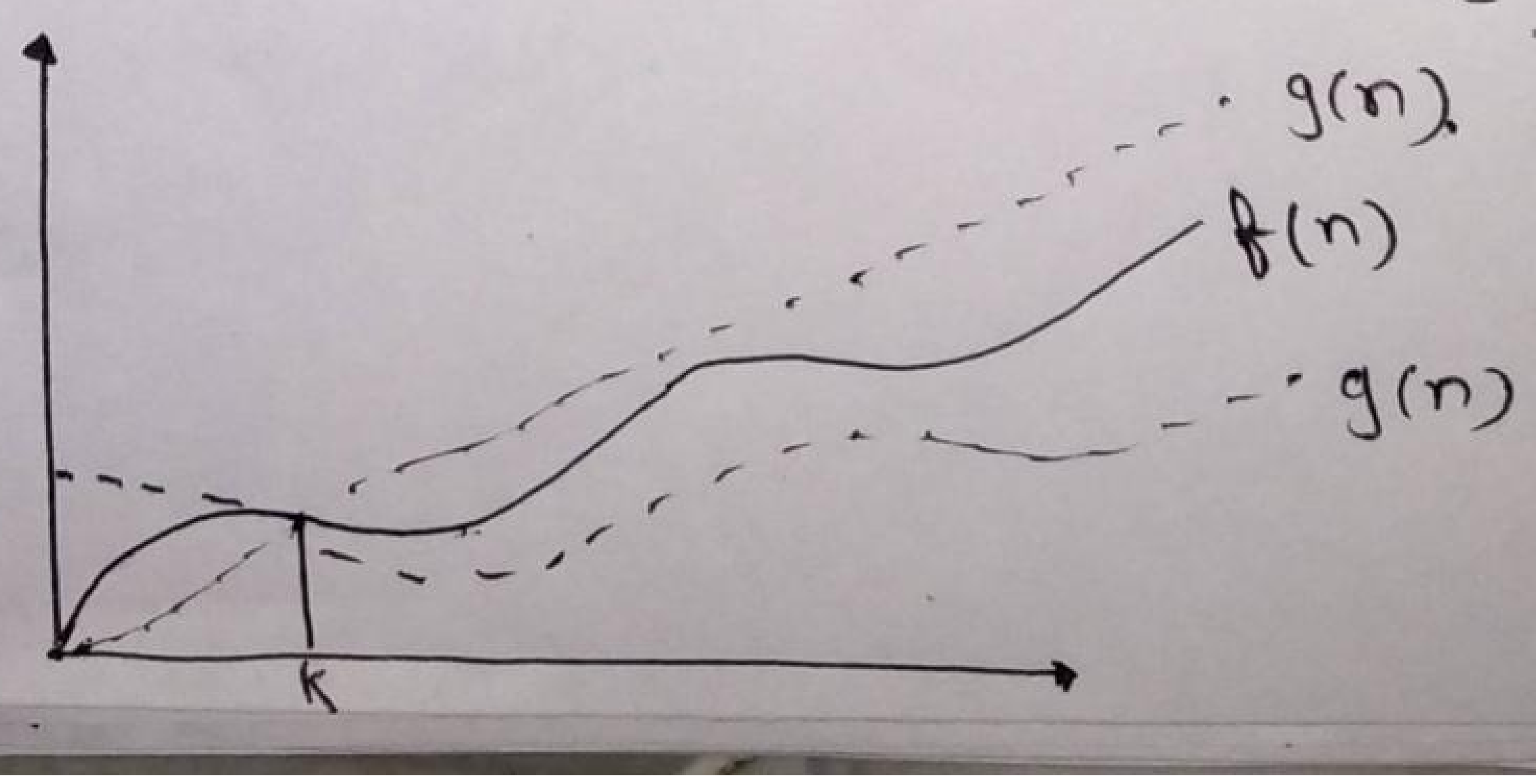
Three types of asymtotic notations to represent the growth of any algorithm, as input increases.

1. Big Theta (θ)
2. Big Oh (O)
3. Big Omega (Ω).

(Big θ) ① The time complexity represented by the Big θ notation is like the average value or range within which the actual time of execution of the algorithm will be.

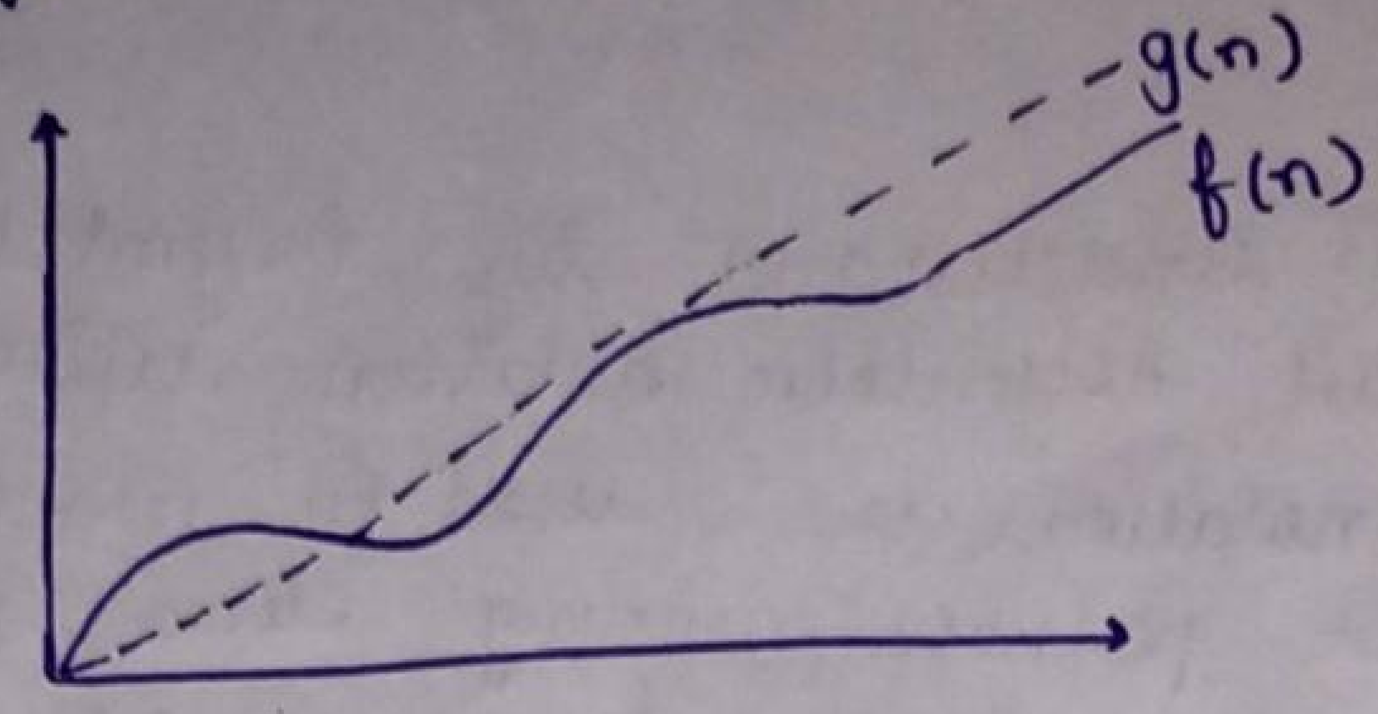
eg. $3n^2 + 5n$.
We use the Big θ notation to represent this, then the time complexity would be $\theta(n^2)$ ignoring the constant coffecient and removing insignificant part, which is $5n$.

$\theta(f(n)) = \{g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0 \}$



2. **big Oh Notation, O**

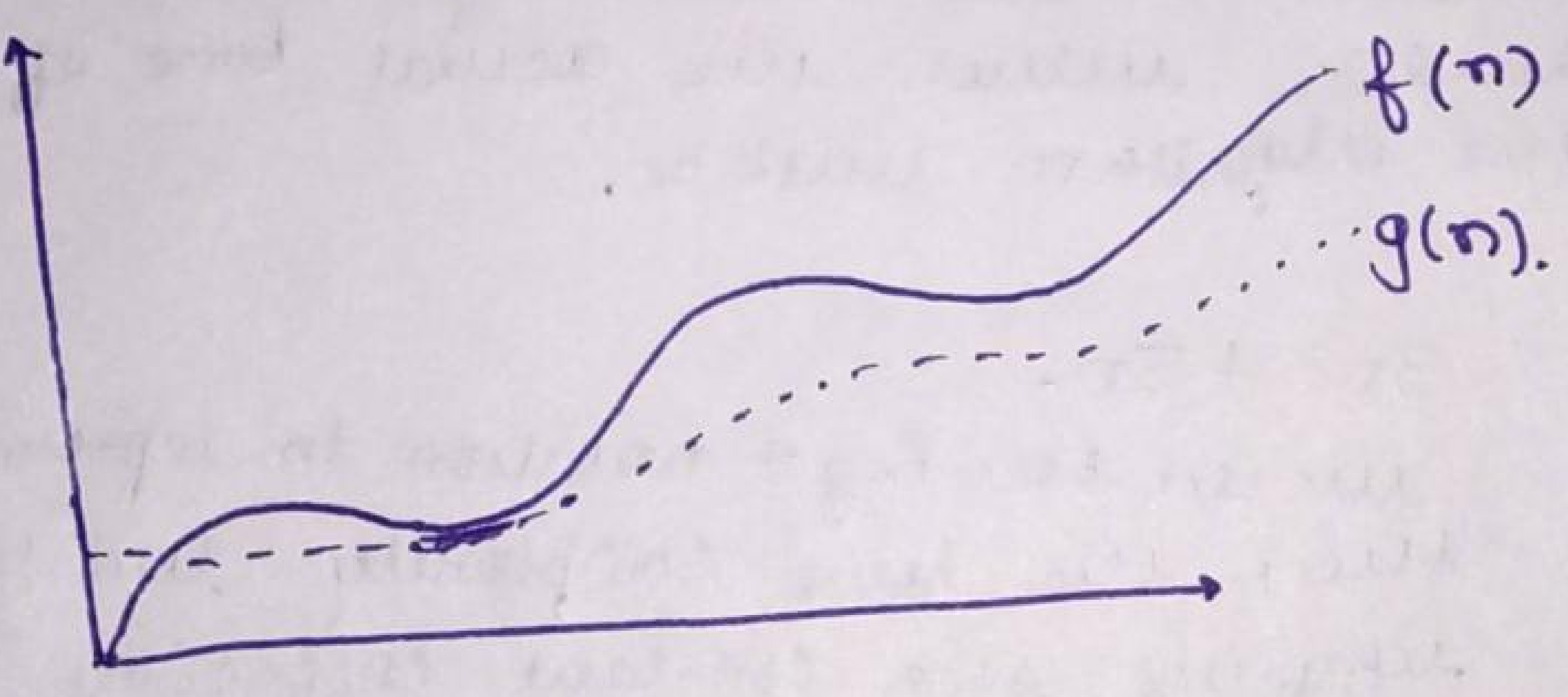
It is the formal way to express the upper bound of an algorithm running time. It measures the worst case time complexity or the longest amount of time an algorithm can possible take to complete.



eg. $O(f(n)) = \{ g(n)$: there exists $c > 0$ and n_0 such that $f(n) \le c \cdot g(n)$ for all $n > n_0$. $\}$

3. **Omega Notation (Ω)**

The notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm's running time. It measure the best case time an algorithm can possibley take to complete.



$\Omega(f(n)) \geq \{ g(n)$: there exists $c > 0$ and n_0 and such that $g(n) <c \cdot f(n)$ for all $n > n_0$. $\}$

Scanned by TapScanner

Question 2. what should be time complexity of - (3)

for (i=1 to n) {i = i*2}.

for (i=1 ; i<n; i=i*2)

1, 2, 3, 4, 8 ----

$T(n) = O(\log_2 n)$

Scanned by TapScanner

(4)

Ques3. $T(n) = \{3T(n-1)$ if $n>0$ otherwise $1\}$.

Let us solve this using substitution.

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

Put $n = n-1$ in eqn 1, we get

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

$$\boxed{T(n-1) = 3T(n-2)} - 2$$

Put the value of $T(n-1)$ from 2 in 1, we get

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

$$\boxed{T(n) = 3^2(T(n-2))} - 3$$

Put $n = n-2$ in eqn 1, we get.

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

$$\boxed{T(n-2) = 3T(n-3)} - 4$$

Put the value of $T(n-2)$ from 4 to 3, we get.

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

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

so

$$T(n) = 3T(n-1)$$
$$= 3(3T(n-2)$$
$$= 3^2(T-2)$$
$$= 3^3(T-3)$$
$$---$$
$$---$$
$$= 3^n T(n-n)$$
$$= 3^n \{ T(0)$$
$$= 3^n (1)$$

$$\boxed{T_n = 3^n}$$

so time complexity of this function is

$O(3^n)$

Scanned by TapScanner

Ques 4. Find the complexity

$$T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n>0 \\ 1, & \text{otherwise.} \end{cases}$$

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

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

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

$= \cancel{2^3(T(n-3)-1)-2-1}$

$= \cancel{2^3(T(n-3))}$

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

$= 2^3\,T(n-3) - 2^2 - 2^1 - 2^0$

$\cdots$

$= 2^n\,T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \cdots$
$\cdots\, 2^2 - 2^1 - 2^0$

$= 2^n(1) - 2^{n-1} - 2^{n-2} - 2^{n-3} \cdots 2^2 - 2^1 - 2^0$

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

$= 2^n - 2^n + 1$

$= 1$

$\boxed{\text{Time complexity is } O(1).}$

Ques 5. Find the complexity of the below program.

```
void function (int n)
{   int i= 1, s=1
    while (s<=n)
    {
        i++;
        s+ =i;
        printf ("#");
    }
}.
```

Answer. We can define the terms 's' according to relation $s_i = s_{i-1} + 1$.

If k is total number of iterations taken by the program.

then while loop terminates

if $1 + 2 + 3 \dots + k$

$= \left[\frac{k(k+1)}{2}\right] > n$

so $k = O(\sqrt{n})$

Time complexity of the above function $O(\sqrt{n})$.

6. Time complexity of -

```
void function (int n) {
    int i, count = 0
        for (i = 1; i*i <= n; i++)
            count++.
```

Ans. if k is the total no. of iterations taken by program.

$\therefore$ then loop terminates

$(1)^2, (2)^2, (3)^2, \dots (\sqrt{n})^2$.

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

Question 7. → $T(n) = O(n * \log_2 n * \log_2 n)$

$T(n) = O(n * (\log_2 n)^2)$

$T(n) = O(n(\log_2 n)^2)$

Scanned by TapScanner

8) Time complexity of

```
function (int n) {
  if (n==1) return;
  for (i=1 to n) {
        for (j= 1 to n) {
              printf("*");
        }
  }
  function (n-3)
}.
```

$T(n) = T(n-3) + n^2$ — ①

$T(n-1) = T(n-\cancel{3}1-3) + (n\cancel{2}-1)^2$

$\boxed{T(n-1) = T(n-4) + n^2}$ — ②

Put the value of $T(n-1)$ from ② in ①, we get

$T(n) = T(n-4) + n^2 + (n-1)^2$

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

⋮

$T(n) = T(n-k) + (n^2 + (n-1)^2 + (n-2)^2 \ldots (k-2)$ terms.

$T(n-k) = 1$

$k = n-1$

$T(n) = T(1) + (n^2 + (n-1)^2 + (n-2)^2 + \ldots (n-3)$ terms

$T(n) = T(1) + (\cancel{n^2}\, 4^2 + 5^2 + \ldots n^2)$

$T(n) = T(1) = \left(\frac{(n-3)(n-2)(2n-5)}{6}\right)$

$T(n) = 1 + \left(\frac{2n^3 + \ldots}{6}\right)$

$T(n) = n^3$

$T(n) = O(n^3)$.

Scanned by TapScanner

9. Time complexity of -

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

$i = 1$ n terms

$i = 2$ $1, 3, 5, \ldots n/2$

$i = 3$, $1, 4, 7 \ldots n/3$

$i = n$ 0

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

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

Q10 for the relation n^k and a^n, what is the relation.

$k \geq 1$ and $a \geq 1$

relation is n^k is $O(c^n)$.

Q11

```
void func (int n)
  int j = 1, i = 0;
  while (i<n)
  {
      i = i+j;
      j++;
  }
```

$0, 3, 6, 10, 15 \ldots n$

$$k\text{th term} = \frac{k(k+1)}{2} = \frac{k^2+k}{2}$$

$k \simeq \sqrt{n}$ $T = O\sqrt{n}$.

Q12. Recurrence Relation of fabonacci Series is

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

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

$$T(n) = 4T(n-4)+3$$

$$T(n) = 8T(n-6)+7$$

$$T(n) = 16T(n-8)+15$$

$$T(n) = 2^k T(n-2k) + (2^k - 1)$$

for $T(n-2k) = T(0)$

$$n = 2k$$

$$k = n/2$$

$$T(n) = 2^{n/2}\, T(0) + (2^{n/2} - 1)$$

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

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

Hence space complexity of fibonacci Series is $O(n)$ as it depends on heiged of recurrsion and it is equal to n in fabonacci Series.

Ans 13 → $n(\log n)$

```
for (int i = 0; j<n; i++)
{
    for (int = 0; i<n; i = i* 0)
    {
        print ("*");
    }
}
void main()
{ fun ();
}
```

Scanned by TapScanner

→ n^3

```
# include <stdio.h>
void main()
{ int n;
  cin>>n
  for (int i=0; i<n; i++) {
     for (int j=0; j<n; j++) {
       for (int k=0; k<n; k++) {
       } x++
     }
  }
}
```

→ log (logn)

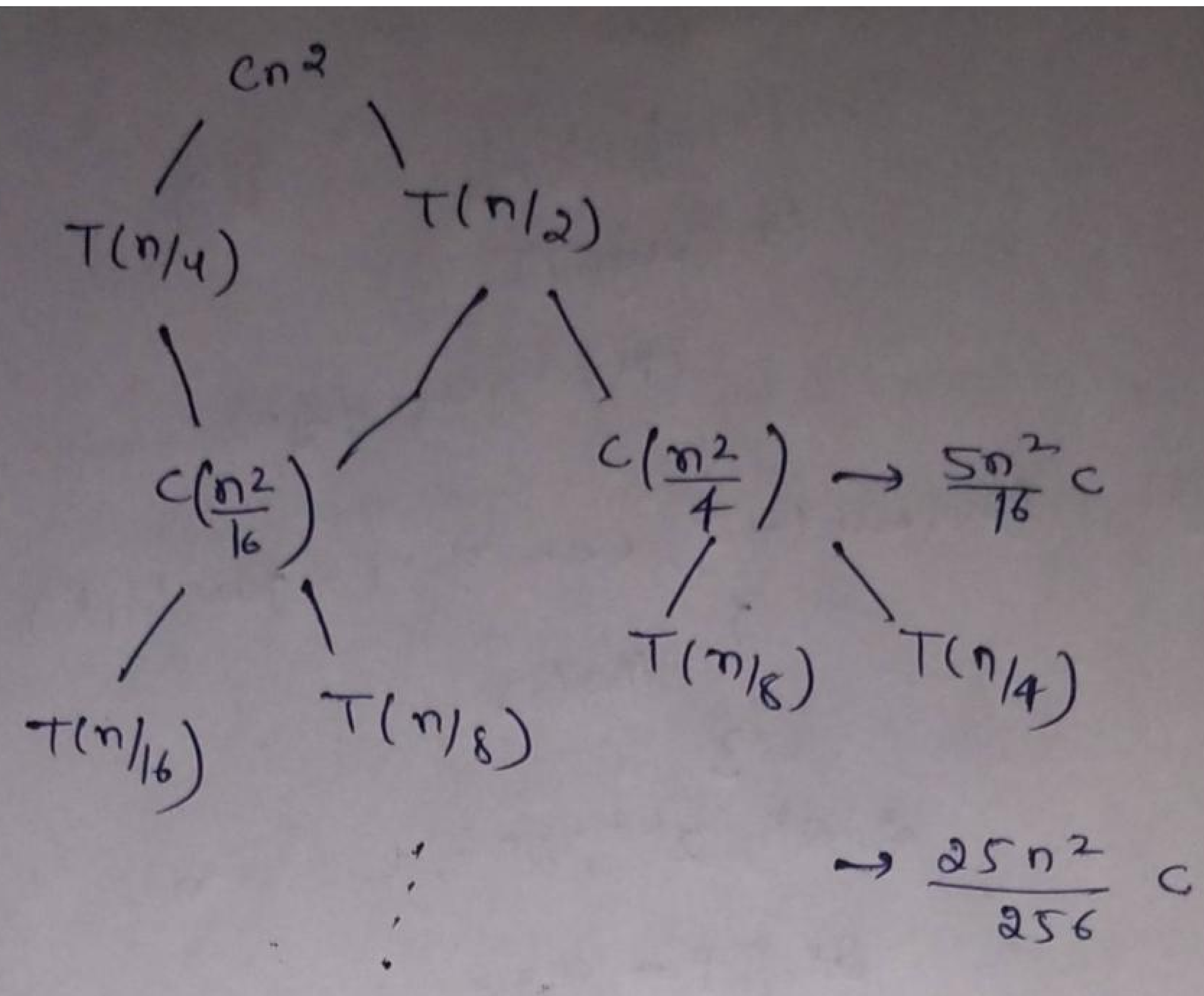
```
#include <bits/stdc++.h>
void func (int n)
{ if (n==2)
    return 1;
  else
    fun(sqrt(n));
}
void main()
{ fun(100);
}
```

Ans14.

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

$$T(1)=0$$

$$T(0)=0$$



$T(n) =$ Cost of each level.

$T(n) = cn^2 + \frac{5cn^2}{16} + \frac{25cn^2}{256} + \cdots$

It is a G.P.

with $a = n^2$

$r = 5/16$

So sum of GP

$T(n) = cn^2\left(\frac{1}{1-\frac{5}{16}}\right) = \frac{16cn^2}{11} = \frac{16cn^2}{11}.$

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

Answer 15:

```
for (int i to n)
{
for (int j = 1; j < n; j += i)
  {
   // O(1)
  }
}
```

$n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5} \cdots\cdots 1$

k times

$k = \log_2 n$

$n(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4} - - - \frac{1}{n})$

$(n(\log n))$

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

Answer 16. for (int i = 2; i <= n; i = pow(i, k))

{

// O(1)

}

$2, 2^k, 2^{k^2}, 2^{k^3}, \ldots n$

It G.P $a = 2$

$r = 2^k$

k^{th} term $= ar^{k-1}$

$n = 2(2^k)^{k-1}$

Let $k^{k-1} = x$

$k\log_k k = \log x$

$k = \log x$ — (1)

$n = 2x$

$\log_2 n = x\log_2 2$

$x = \log_2 n$

$\log x = \log(\log n)$

from (1)

$k = \log(\log(n))$

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

Answer 17.

Pivot is divided in 99% and 1%.

so

$$T(n) = T\left(\frac{99}{100}N\right) + T\left(\frac{N}{100}\right) + N$$

Now so here we can use 2 extreme of a tree where starting point is N

$$N \rightarrow \frac{99N}{100}, \quad \frac{N}{100}$$

$$\frac{99N}{100} \rightarrow \left(\frac{99}{100}\right)^2 N, \quad \frac{99N}{(100)^2}$$

$$\frac{N}{100} \rightarrow \frac{99N}{(100)^2}, \quad \frac{N}{(100)^2}$$

N

N

$$N\left(\frac{(99)(99)}{(100)(100)} + \frac{99(1)}{(100)(100)}\right) + \frac{1}{100\times 100}N = \frac{99N}{100} + \frac{N}{100}$$

$$= N$$

$$= N$$

So cost of each level is N only.

Total cost = height * cost of each level.

So for first stream $= N, \frac{99}{100}N, \left(\frac{99}{100}\right)^2 N$ ——

$$\left(\frac{99}{100}\right)^{h-1} N = 1$$

$$\left(\frac{99}{100}\right)^{h-1} = \frac{1}{N}$$

$$N = \left(\frac{100}{99}\right)^{h-1}$$

$$\log N = h\log(1)$$

$$h = \log N \text{ or}$$

$$h = \frac{\log N}{\log\left(\frac{100}{99}\right)} + 1$$

height of second stream.

$N, \frac{N}{100}, \frac{N}{(100)^2} + \frac{N}{(100)^3} + \cdots]$

$$N\left(\frac{1}{100}\right)^{n-1} = 1$$

$$N = (100)^{n-1}$$

$$(n-1)\log 100 = \log N.$$

$h = \frac{\log N}{\log 100} + 1$ & $h = \log N$ (appnx)

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

So time complexity is $O(N\log N)$

height of both extreme is $\frac{\log N}{\log 10} + 1$ of $\frac{1}{100}$

and $\frac{\log N}{\log\left(\frac{100}{99}\right)} + 1$ of $\frac{99}{100}$

So we conclude that if division is done more and more than height of tree will be more and when division ratio is less than height is less.

Answer 18- n, n!, log n, log log n, root(n), n log n, 2^n, 2^{2n}, 4n, 100.

$O(100) < O\log\log N) < O\log N < O(\sqrt{n}) < O(n) < O(n\log N)$
$< O(n^2) < O(2^n) < O(2^{2n}) < O(4^n)$.

(b) $2(2^n)$, $4n$, $2n$, 1, $\log(n)$, $\log(\log(n))$, $\sqrt{\log n}$, $\log 2n$, $2\log n$, n, $\log(n!)$, $n!$, n^2, $n\log n$

$O(1) < O(\log(\log(n))) < O(\log(n)) < O(\log 2n) < O(2\log n)$
$< O(n) < O(n\log(n)) < O(\log(n!)) < O(2n)$
$< O(4n) < O(n^2) < O(n!) < O(2(2^n))$.

Scanned by TapScanner

(c) $O(96) < O(\log_8(n) < O \log n(n) < O(\log n) < O n \log(n)) <$
$O(n \log_2(n)) < O(5n) < O(8n^3) < O(7n^3) < O(n!) <$
$O(8^{\wedge}2n)$.

Ans 19.

```
void linear search (int arr[], int n, int n)
{ for (i=0 to i=n)
      if arr[i] = = key
          cout << "found";
      else
          continue.
}.
```

Iterative Insertion Sort -

```
void insertion sort (arr, n) {
   int i, temp, j
      for (i-1 to n)
      { temp = arr[i]
        j = i-1
          while j>=0 && arr[j] > temp.
          { arr[j+1] = arr[j]
              j--
          }
          arr [j+1] = temp;
      }
   }.
```

Insertion

```
{  if n <= 1
       return;
   insertion sort (arr, n-1),
   last = arr [n-1];
       j = n-2
     j while (j>=0 and arr [j] > last]
        { arr [j+1] = arr [j]
            j--
        }
        arr [j+1 = last'
```

Insertion sort is called online sorting because it don't know the whole input, it might make decision that later turn out to be not optimal. other algorithm are off-line algorithms.

Answer 21.

	Time Complexity			Space
	Best	Avg	Worst	
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$ {due to recursion}
Quick sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(n)$
Heap Sort.	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$.

22.

	Inplace	Stable	Online sorting
Bubble sort	Yes	Yes	NO
Selection sort	yes	NO	NO
Insertion sort	Yes	yes	yes
Merge sort	NO	yes	NO
Quick sort	yes	NO	NO
Heap sort.	yes	NO	NO

Answer 23.

Binary Search (arr, int n, key)
{
  beg = 0
  end = n-1
  while (beg <= end)
    mid = (beg+end) / 2
    if [arr [mid] == key]
      found
      else if arr [mid] < key
          beg = mid + 1
      else
          end = mid - 1
    }
}

Time complexity of Linear Search = $O(n)$
Space " " " " - $O(1)$

Time Complexity of Binary Search = $O(\log n)$
Space Complexity of Binary Search = $O(n)$

Ans 24. $T(n) = T(\frac{n}{2}) + 1$