Q1→

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

Three types of asymptotic notation to represent the growth of any algo, are →

(i) → Big Theta (Θ)

(ii) → Big-oh (O)

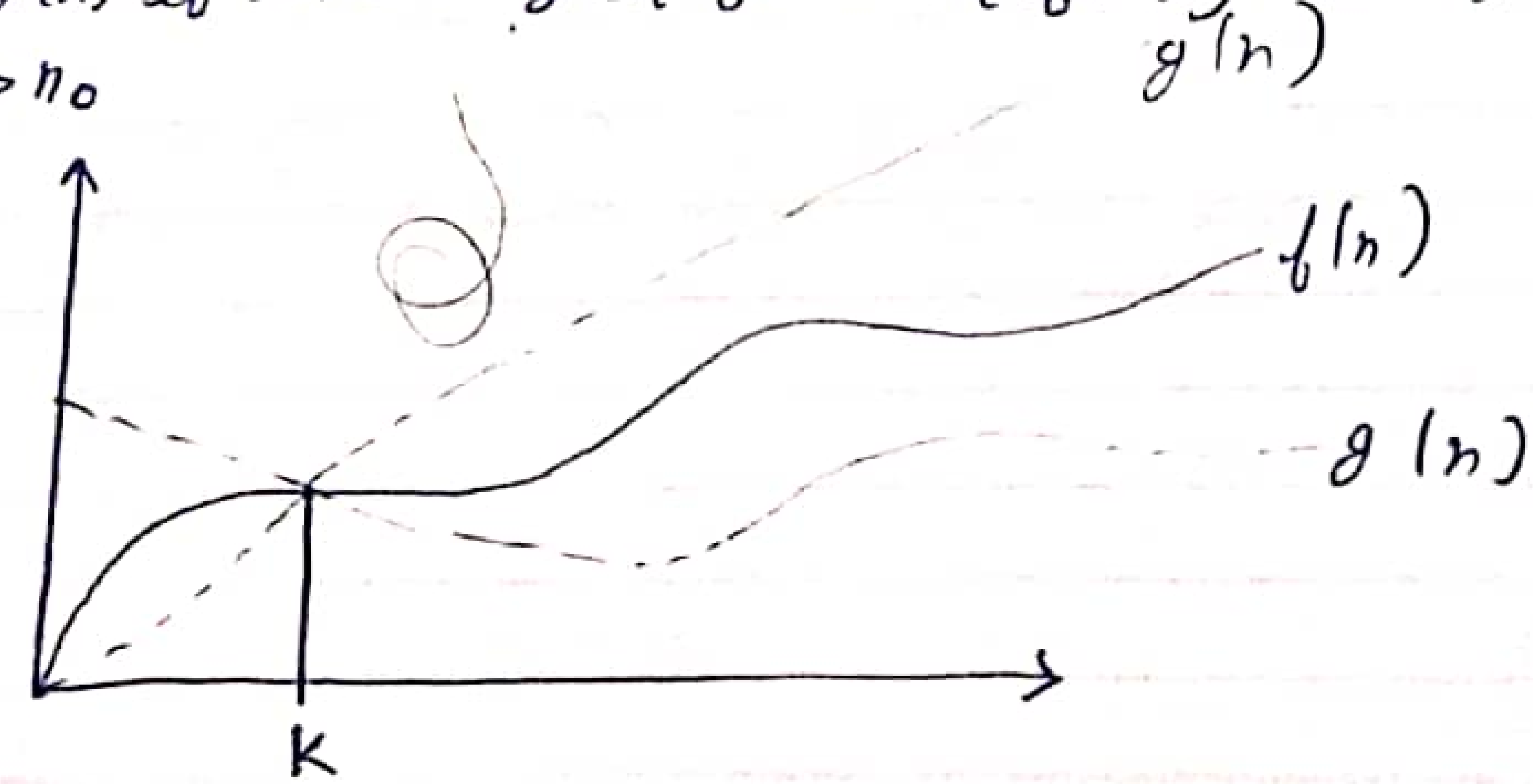
(iii) → Big omega (Ω)

(i) - The time complexity represented by the Big Θ notation is like the average value ~~or~~ within which the actual time of execution of the algo will be

eg. $3n^2+5n$

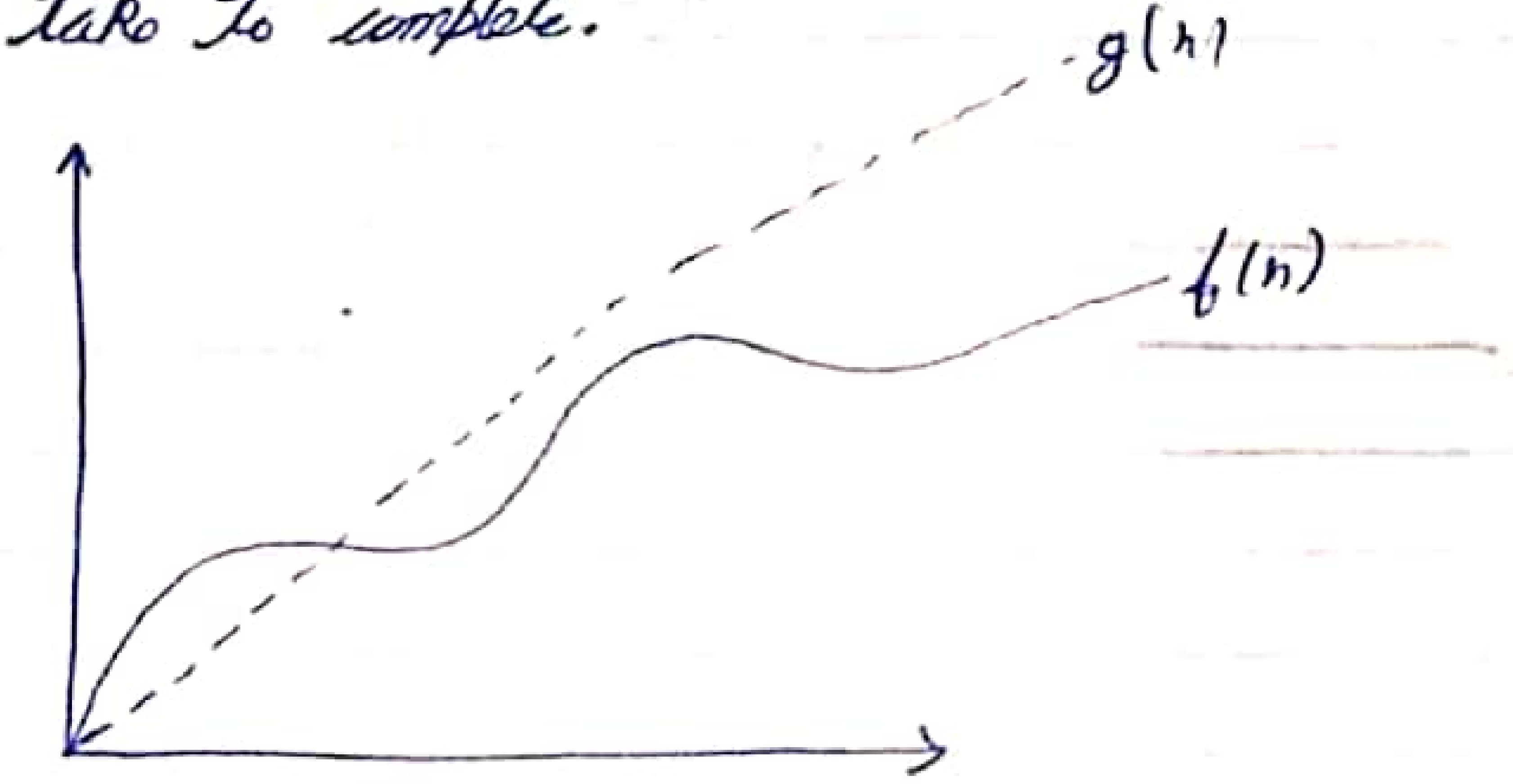
we use the Big Θ notation to represent this, then the time complexity would be $\Theta(n^2)$ ignoring the constant coefficient and remaining insignificant part, which is $5n$.

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



(ii). Big Oh Notation (O)

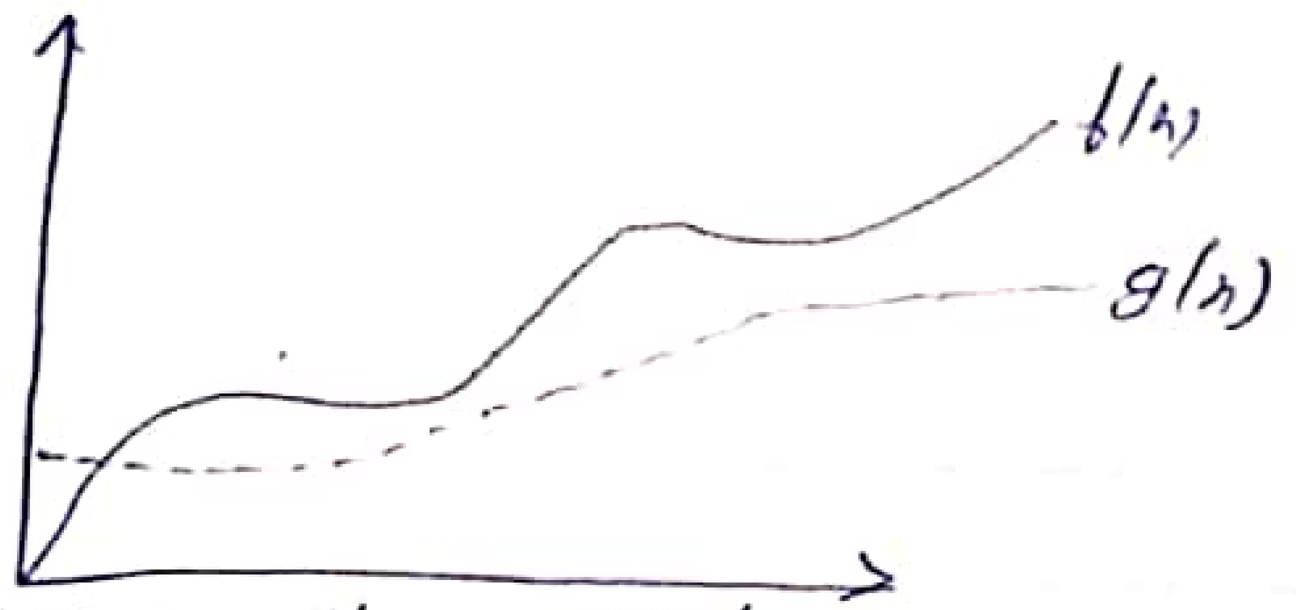
→ It is the formal way to express upper bound of all algorithm running time. It measures the worst case time complexity or the longest amount of time all algo can possible take to complete.



e.g → $O(f(n)) = \{g(n):$ there exist $c > 0$ and n_0 such that $f(n) \leq c.g(n)$ for all $n > n_0\}$

(iii) Omega notation (Ω)

– it is the formal way to express the lower bound of an algo.'s running time. It measure the best case time an algo can possibly take to complete.



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

2.

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

1, 2, 3, 4, 5 .....

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

3.

Sol- $T(n) = 3T(n-1)$ — (i)

put n=n-1 in equ (i) we get

$T(n-1) = 3T(n-2)$ — (ii)

put value of . T(n-1) in equ (i)

~~T(n-2)~~ ⇒ $T(n) = 3\{3T(n-2)\}$

$T(n) = 3^2 T(n-2)$ — (iii)

put ~~value of~~ n = n-2 in equ (i) we get

$T(n-2) = 3T(n-3)$ — (iv)

put the value

---

(2) ⇒ for (i=1 to n) {i=i*2}

series will be

$\underbrace{1, 2, 4, 8, 16, 32, 64, \text{------}, n}_{k \text{ term}}$

so using gp $a_n = a r^{k-1}$

$a_n = n$, $a = 1$, $r = 2$

$n = 1.2^{k-1}$

$n = \frac{2^k}{2} \Rightarrow 2n = 2^k$

taking log base 2 on both sid

$\log_2 2n = \log_2 2^k$

$\log_2 2 + \log_2 n = k \log_2 2$

$1 + \log_2 n = k$

$k = TC = O(\log_2 n) = \log n$

3.
Sol.

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

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

using substitution

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

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

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

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

for k terms

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

let $T(n-k) = T(0)$

$n = k$

Using $n = k$

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

$T(n) = 3^n \cdot 1 = 3^n$

4→

sol $T(n) = 2T(n-1) - 1$ $\quad T(0) = 1$

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

$T(n) = 2(2T(n-2) - 1] - 1 \rightarrow T(n-2) = 2T(n-3) - 1$

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

$T(n) = 2[4T(n-3) - 2] - 1 = 8T(n-3) - 7$

for k^{th} term

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

let $T(n-k) = T(0)$

$n = k$

using $k = n$

$T(n) = 2^n(T(0)) - (2^n - 1)$

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

$T(n) = 1$

5>
sol.

$i=1, S=1;$
while $(S<=n)$
$\{ i++; s=s+i; \}$

Here no. of steps

no. of steps	i	S
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15

so order of growth is not const. so general term can be taken as $\frac{k(k+1)}{2}$ let at n no. of terms is k so $n=\frac{k^2+k}{2} \Rightarrow 2n=k^2+k$

ignoring lower order terms

$n=k^2$

$k=\sqrt{n}$

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

6→ for(i=1; i*i <=n; i++)
count++
so i is moving from 1 to $\sqrt{n}$ with linear growth so
$T(n)=O(\sqrt{n})$

7→ for (i=n/2; i<=n; i++)
for (j=1; j<=n; j=j*2)
for (k=1; k<=n; k=k*2)
count++
k and j loop are moving from 1 to n with exponential growth so T(
for these two loop will be $O(\log n)$
and i loop going through $n/2$ to n so with constant aggrement
so $T(n)=O(n/2)=O(n)$
so overall complexity of the f^n will be $T(n)=O(n.\log n.\log n)$

8. $T(n)=T(n-3)+n^2$
$T(1)=1$
$T(n)=T(n-3)+n^2 \rightarrow T(n-6)+(n-3)^2$
$T(n)=T(n-6)+(n-3)^2+n^2 \rightarrow T(n-9)+(n-6)^2$
$T(n)=T(n-9)+(n-6)^2+(n-3)^2+n^2 \rightarrow T(n-12)+(n-9)^2$
$T(n)=T(n-12)+(n-9)^2+(n-6)^2+(n-3)^2+n^2$ so for kth term

$T(n)=T(n-k)+(n-(k-3))^2+(n-(k-6))^2+(n-(k-9))^2+(n-(k+12))^2+$
$------+(n-(k-k))^2$

let $T(n-k) = T(1)$

$n = 1+k = k = n-1$

$T(n) = T[n-(n-1)] + [n-(n-1-3)]^2 + [n-(n-1-6)]^2 + [n-(n-1-9)]^2 + \dots + n^2$

$T(n) = T(1) + 4^2 + 7^2 + 10^2 + \dots + n^2$

$T(n) = 1 + 4^2 + 7^2 + 10^2 + \dots + n^2$

$\Rightarrow$

$n = (3*k-2)^2$

$n = 9k^2 + 4 - 12k$

$n = k^2$

$k = \sqrt{n}$

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

9 → for (i=1 to n)
    for (j=1; j<=n; j=j+i)
        print("*")

steps

i	1	2	3	4	......	n
j	n	$n/2$	$n/3$	$n/4$	......	$n/n = 1$

so total complexity is

$T(\ = n + n/2 + n/3 + n/4 + n/5 + \dots + n/n$

$\Rightarrow n[1 + 1/2 + 1/3 + 1/4 + 1/5 + \dots 1/n]$

$\Rightarrow n \int_1^n \frac{1}{x} dx$

$\Rightarrow n[\log x]_1^n = n[\log n - \log(1)] = n \log n$

10 → if $c > 1$, then the exponential c^n far outgrows any term, so the ans is n^k is $O(c^n)$

11 → int i=1, j=0;
while (i<n)
{ i=i+j;
   j++;
}

$i = 0 \ 1 \ 2 \ 3 \ 6 \ 10 \ 15$ ---

$j = 1 \ 2 \ 3 \ 4 \ 5 \ 6$

so i will go on till n and general formula for k^{th} term is

$n = \frac{k(k+1)}{2}$

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

12.

Sol. $T(n) = T(n-1) + T(n-2) + C$

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

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

$T(n) = 2T(n-1) + C \rightarrow T(n-1) = 2T(n-2) + C$

$T(n) = 2[2T(n-2) + C] + C = 4T(n-2) + 3C \rightarrow T(n-2) = 2T(n-3) + C$

$T(n) = 4[2T(n-3) + C] + 3C = 8T(n-3) + 7C$

so on k^{th} term will be

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

and $T(n-k)$ for $k =$ first term n will be 1 for $n = k$

$2^n \cdot 1 + (2^n - 1)C = T(n)$

$T(n) = 2^n + 2^n C - C = 2^n[1+C] - C$

neglecting the constants

$T C = O(2^n)$

and max depth for recursion f^n is based on n, so space complexity is $O(n)$

13.

sol. $n(\log n)$

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j*=2)
        print ("*");
```

n^3

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
        for (int k=0; k<n; k++)
            print ("*");
```

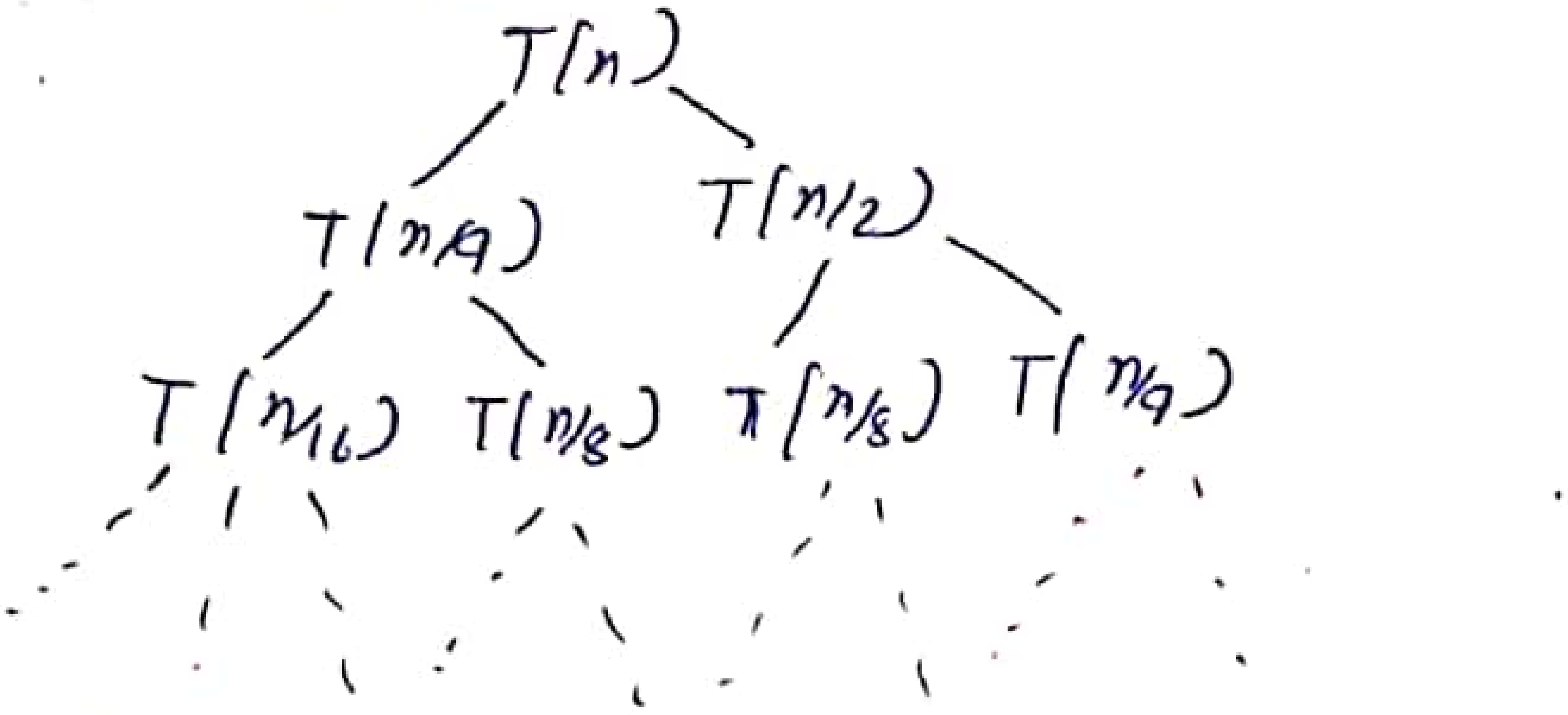
14. ~~$T(n) = T(n/4) + T(n/2) + Cn^2$~~

$\log(\log n)$

```
for(int i=0; i<n; i=i*1)
    print ("*");
```

14 →

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



$T(n) = c[n^2 + 5n^2/16 + 25n^2/256 + \cdots]$

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

15 → for (int i=1; i<=n; i++)

for (int j= 1; j<n; j+=1)

O(1);

$i = 1 \quad 2 \quad 3 \quad 4 \quad \cdots\cdots \quad n$

$j = n \quad n/2 \quad n/3 \quad n/4 \quad \cdots\cdots \quad n/n$

time complexity

$T(n) = n + n/2 + n/3 + n/4 + \cdots\cdots + n/n$

$T(n) = n[1 + 1/2 + 1/3 + 1/4 + \cdots\cdots + 1/n] = n\int_1^n \frac{1}{x}\,dx = n[\log n - \log 1] = n \log n$

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

O(1)

for any f^n increasing with exponential rate the TC become

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

18 → a → $100 < \log\log n < \log n < n\log n < root(n) < n < n^2 < 2^n < 2^{2n} < 4^n < \log(n!) < n!$

b → $1 < \log(\log n) < \sqrt{\log(n)} < \log n < \log 2n < 2\log n < n\log n < 2(2^n) < \log n! < n!$

c → $96 < \log_8(n) < \log_2(n) < n\log_6(n) < n\log_2(n) < 5n < 8n^2 < 7n^3 < 8^{2n} < \log(n!) < n!$

Scanned with CamScanner

19.

21. Search(a,n)
low = 0, high = n, key
while (low < high)
mid = $\frac{low + high}{2}$
if a[mid] = key
return ~~which~~ mid
else if a[mid] < key
low = mid+1
else high = mid+1

20. complexity

complexity	algo
$O(n^2)$	Bubble sort
$O(n^2)$	selection "
$O(n^2)$	Insertion "
$O(n \log n)$	merge "
$O(n \log n)$	Heap "
$O(n \log n)$	Quick "

22.

Sorting algo	Inplace	Stable	Online
Bubble	Yes	Yes	no
Selection	Yes	no	no
Insertion	Yes	Yes	Yes
Quick	Yes	no	no
merge	no	Yes	no
Heap	Yes	no	no

Scanned with CamScanner

23 -

Search (a, n, key)
low = 0, high = n
while (low < high)
mid = $\frac{low+high}{2}$
if a[mid] is key
return mid
else if a[mid] < key
low = mid+1
else high = mid-1

Recursion

- Search (a, low, high, key)
if low < high
mid = $\frac{low+high}{2}$
if a[mid] < key
Search (a, mid+1, high, key)
- else if a[mid] > key
Search (a, low, mid-1, key)
- else return mid

Linear search	TC = O(n)	SC = constant
Binary "	TC = O(log n)	SC = "

24 - $T(n) = T(n/2) + C$