

Divyanshu Kandpal
B.Tech (CSE) Sem - 5
Sec - C Rollno - 1961046.

DA A

Assignment - 1

- i) Asymptotic notations are the mathematical notation used to describe the running time of an algo when the I/P tends towards a particular value or limiting value.
There are mainly 3 asymptotic notations:-

* Big-O-notation: It represent the upper bound of running time of an algo.

- This notation is called as upper bound of an algo or a worst case of algo.
- $O(g(n)) = \alpha f(n)$: There exist positive constraint c and n_0 such that $0 \leq f(n) \leq g(n)$ for all $n > n_0$, where $c > 0$ and $n \geq n_0$.
- eg: $f(n) = 3\log n + 100$
 $g(n) = \log(n)$
 $3\log n + 100 \leq c \times \log(n)$
 $c = 180$ and $n \geq 2$ (undefined at $n=1$)

* Big Omega (Ω) notation: It represent the lower bound of the running time of an algo.

- This notation is known as lower bound of an algo, or best case of an algo.
- $\Omega(g(n)) = \alpha f(n)$: There exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n) \forall n > n_0$.
- eg: $f(n) = 3n + 2$
 $cg(n) \leq f(n)$

($C = \text{constant}$, $g(n) = n$)

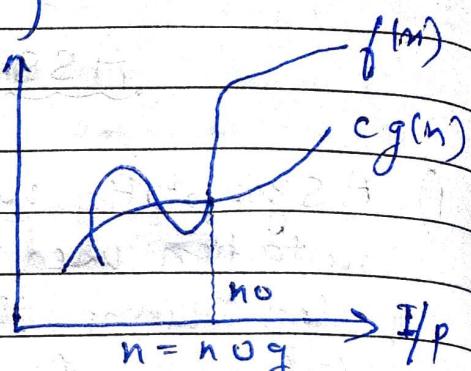
$$Cn \leq 3n+2$$

$$n(C-3) \leq 2$$

$$\Rightarrow n \leq \frac{2}{C-3}$$

if we assume $C=4$, then $n_0 = 2$

($C=4, n_0 = 2$).



* Theta (Θ) notation - It encloses the function from above and below. Since it represent upper and lower bound.

- This is known as tight bounds of an algo, or an average case of algo.

- $\Theta(g(n)) = \Omega(f(n))$: There exist positive constant C_1, C_2 , and n_0 such that:

$$C_1 * g(n) \leq f(n) \leq C_2 * g(n) \quad \forall n > n_0.$$

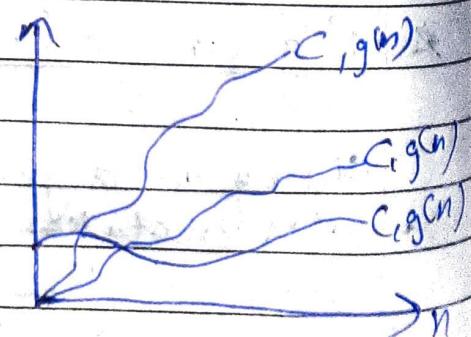
- e.g.: $f(n) = 5n^3 + 16n^2 + 3n + 3$

$$5n^3 \leq 5n^3 + 16n^2 + 3n + 3 \leq (5+16+3+3)n^3$$

$$5n^3 \leq f(n) \leq 32n^3$$

$$[C_1 = 5; C_2 = 32; n_0 = 1]$$

$$f(n) \Leftrightarrow \Theta(n^3)$$



2) $i = 2, 4, 8, 16 \dots$ \leftarrow K^{th} term $\rightarrow n$

$$G_n = ar^{n-1}$$

$$G_n = 1(2)^{K-1}$$

$$n = 2^{K-1}$$

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

$$(K = \log_2 n + 1)$$

$$O(n) \in \log n \Rightarrow O(1) \cdot O(\log n) = O(\log n)$$

3) $T(n) = 23T(n-1)$ if $n > 0$. (i.e. $n \geq 1$)

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

$$T(n) = 3 \times 3T(n-2) \leftarrow T(n-2) = 3T(n-3)$$

$$T(n) = 3 \times 3 \times 3T(n-3) \leftarrow T(n-3) = 3T(n-4)$$

$$T(n) = 3^3T(n-3) \leftarrow T(n-3) = 3T(n-4)$$

$$T(n) = 3^3 \times 3T(n-4)$$

$$T(n) = 3^4 \times T(n-4)$$

General form :

$$T(n) = 3T(n-1) \dots \textcircled{1} \quad (T(0)=1)$$

$$T(n-i) = T(0)$$

$$\cancel{n-i=0} \Rightarrow [n=i]$$

Putting $n=i$ in $\textcircled{1}$;

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

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

$$\cancel{3^n} \cdot [T(n) = O(3^n)].$$

4) $T(n) = 2T(n-1) - 1 \quad \leftarrow T(n-1) = 2T(n-2) - 1$

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

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

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

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

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

$$\downarrow \quad T(n-3) = 2T(n-4) - 1$$

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

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

General form:

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

$$T(n-i) = T(0)$$

$$n-i=0$$

$$(n=i)$$

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

$$[T(0)=1]$$

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

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

$$2-1$$

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

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

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

5) No of steps (K)

s

i

0

1

0

1

2

1

2

3

3

3

4

6

4

5

10

5

6

15

6

K

21

7

n

1

$$T(n) = O(K)$$

$$i = 0, 1, 3, 6, 10, \dots, n = \frac{(K-1)K}{2} = \frac{K^2 - K}{2}$$

$$S_n = 1 + 3 + 6 + 10 + \dots + n = \frac{(n+1)n}{2}$$

$$\frac{S_n}{2} = 1 + 3 + 6 + 10 + \dots + (n+1) + n$$

$$O = 1 + 2 + 3 + 4 + 5 + \dots + n$$

$$n = 1 + 2 + 3 + 4 + \dots + K \text{ step}$$

$$n = \frac{K}{2} [2(1) + (K-1)1]$$

$$2n = K[2 + K - 1]$$

$$2n = K^2 + K \Rightarrow 2n = \left(\frac{K+1}{2}\right)^2 - \left(\frac{1}{2}\right)^2$$

$$2n + \left(\frac{1}{2}\right)^2 = \left(\frac{K+1}{2}\right)^2$$

$$K + \frac{1}{2} = \sqrt{2n + \left(\frac{1}{2}\right)^2}$$

$$K = \sqrt{2n + \left(\frac{1}{2}\right)^2} = 1/2$$

$$T(n) = T(K)$$

$$T(n) = T\left(\sqrt{2n + \left(\frac{1}{2}\right)^2} - 1/2\right) \Rightarrow O(\sqrt{n})$$

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

6) Since, i is moving from 1 to \sqrt{n} with linear growth so,

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

7) $O(n \log n \log n)$

$$[O(n(\log n)^2)]$$

$$8) T(n) = T(n-1) + n^2 [T(n-1) = T(n-2) + (n-1)]$$

$$T(n) = T(n-2) + n^2 + (n-1)^2 [T(n-2) = T(n-3) + (n-2)]$$

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

General formula:

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

$$T(n) = T(n-i) + n^2 + (n-1)^2 + (n-2)^2 + \dots + (n-i)^2$$

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

$$n = i+1 \Rightarrow [n=1=i]$$

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

~~$$T(n) = T(n-(n-1)) + n^3 + (n-1)^2 + (n-2)^2 + \dots + (n-(n-1))^2$$~~

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

$$T(n) = 1 + 1^2 + 2^2 + 3^2 + \dots + n^2$$

$$T(n) = \frac{n(n+1)(2n+1)}{6} \Rightarrow [T(n) = O(n^3)]$$

$$9) O(n\sqrt{n})$$

10) If $c > 1$, then exponential c^n for outranks any term, so the answer is: n^R is $O(c^n)$

$$11) i = 0, 1, 3, 6, 10, 15 \dots$$

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

& i will go n times, so general formula for Kth term is $n = \frac{K(K+1)}{2}$

$$\therefore T \in O(\sqrt{n})$$

$$12) T(n) = T(n-1) + T(n-2) + C$$

$$T(n-2) \approx T(n-1)$$

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

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

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

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

$$\overbrace{T(n-2)}^C = 2T(n-3) + C$$

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

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

General form:-

$$T(n) = 2^n T(n-i) + (2 + 2^1 + 2^2 + \dots + 2^{n-1}) C$$

$$n-i = 0$$

$$n=i$$

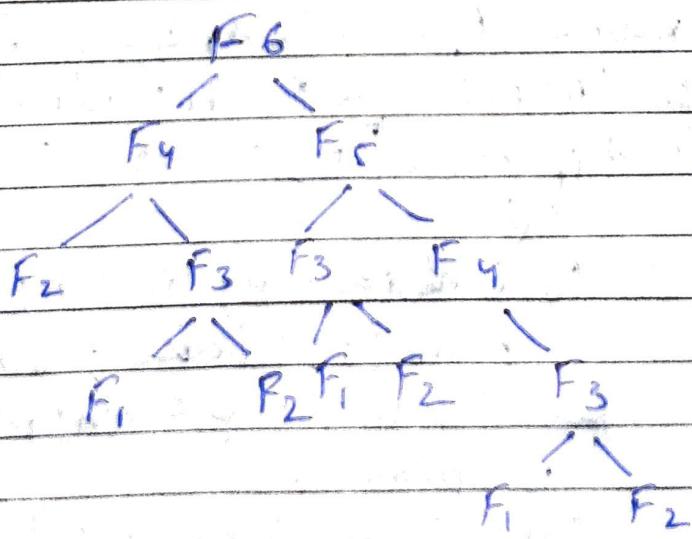
$$T(n) = 2^n T(0) + (2 + 2^1 + 2^2 + \dots + 2^{n-1}) C$$

$$T(n) = 2^n (1) + \frac{2^0 (2^{n-1} - 1)}{2-1} C$$

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

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

Fig →



The max depth is proportional to N , hence the space com of Fibonacci recursive is $O(N)$.

13) i) $n \log n$

```
void func()
{
    int i, j;
    for (i = 1; i <= n; i++)
        for (j = 0; j <= n; j = j + 2)
            printf("%#");
    printf("\n");
}
```

ii) n^3

```
void fun(int n)
{
    int i, j, k;
    for (i = 0; i <= n; i++)
        for (j = 0; j <= n; j++)
            for (k = 0; k <= n; k++)
                printf("%#");
}
```

iii) $\log(\log n)$

```
Void sieve of Eratosthenes (int n)
{
    bool prime[n+1];
    memset(prime, true, sizeof(prime));
    for (int p=2; p*p<=n; p++)
        if (prime[p] == true)
            for (int i = p*p; i <= n; i += p)
                prime[i] = false;
    for (int p=2; p <= n; p++)
        if (prime[p])
            cout << p << endl;
}
```

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

$$T(1) = C$$

$$n = n/2$$

$$T(n/2) = T(n/8) + T(n/4) + C(n^2/4)$$

$$T(n) = T(n/4) + 2 + (n/16) + C(n^3/16 + n^2/4 + n^c)$$

$$\begin{array}{ccccc}
 & T(n) & & & \\
 & \swarrow & \searrow & & \\
 T(n/4) & & T(n/2) & & T(n/4) \\
 & \downarrow & \downarrow & & \\
 T(n/16) & T(n/8) & T(n/8) & T(n/4) &
 \end{array}$$

$$T(n) = C[n^2 + \frac{5n^3}{16} + \frac{25n^4}{256} + \dots]$$

$$T(n) = n^2 C \left[1 + \frac{5}{16} + \frac{5^2}{16^2} + \dots \right]$$

$$[T(n) = O(n^c)]$$

15) For $i=1$; inner loop is executed n times.

for $i=2$; inner loop is executed $n/2$ times.

for $i=3$; inner loop is executed $n/3$ times.

for $i=n$; inner loop is executed n/n times.

$$\begin{aligned}
 \text{Total time} &= n + n/2 + n/3 + \dots + n/n \\
 &= n(1 + 1/2 + 1/3 + \dots + 1/n) \\
 &= n \log n.
 \end{aligned}$$

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

$$16) O(\log(\log n))$$

18) a) $\log n$, $\log \log n$, $\log n$, $n \log n$, n , $n \log n$, n^2 , 2^n , 2^{2n} , 4^n , $n^{\frac{1}{2}}$

b) 1, $\log(\log(n))$, $\sqrt{\log(n)}$, $\log n$, $\log(2n)$, $\log(n^6)$, $2\log(n)$, n , $2n$, $4n$, $n \log(n)$, n^2 , $2^{(2^n)}$, n^6 .

c) 96, $\log n$, $\log_2 n$, $\log(n^6)$, $5n$, $n \log n$, $n \log_2 n$, $8n^2$, $7n^3$, $8n^6$, n^6 .

19) Linear Search (A , key)

$comp \leftarrow 0$, $t \leftarrow 0$

for $i = 1$ to $A.length$

$comp \leftarrow comp + 1$

if $A[i] == key$

print "Element found"

$f = 1$

if $f == 0$

print "Element not found"

print comp.

20) Iterative method of Insertion Sort \rightarrow

for $j = 2$ to $A.length$

Key = $A[j]$

$i \leftarrow j - 1$

while $i > 0$ and $A[i] > key$

$A[i+1] = A[i]$

$i = i - 1$

$A[i+1] = key$

Recursive Method \rightarrow

Insertion Sort (A, n)

If $n \leq 1$

return

Insertion Sort ($A, n-1$)

Key = $A[n-1]$;

$j = m-2$;

while $j \geq 0$ and $A[j] > \text{key}$

$A[j+1] = A[j]$

$j = j - 1$

$A[j+1] = \text{key}$.

Insertion sort considers "one I/p element per iteration and produces a partial sort" without considering future elements. That's why it is called online sorting.

Other sorting algs that have been taught:

- Bubble sort • Merge sort • Selection sort
- Heap sort • Quick sort • Counting sort.

Q1)

	Best case	Avg case	Worst case.
Bubble sort	$\Omega(N)$	$\Theta(N^2)$	(N^2)
Selection	$\Omega(N^2)$	$\Theta(N^2)$	(N^2)
Insertion	$\Omega(N)$	$\Theta(N^3)$	(N^2)
Merge	$\Omega(N \log N)$	$\Theta(N \log N)$	$(N \log N)$
Heap	$\Omega(N \log N)$	$\Theta(N \log N)$	$(N \log N)$
Quick	$\Omega(N \log N)$	$\Theta(N \log N)$	(N^2)
Counting	$\Omega(N+K)$	$\Theta(N+K)$	$O(N+K)$

22)

	In place	stable	Online
Bubble	Yes	Yes	Yes
Insertion	Yes	Yes	Yes
Selection	Yes	No	Yes
Merge	No	Yes	Yes
Quick	Yes	No	Yes
Heap	Yes	No	Yes
Cant	No	Yes	Yes

23) • Linear Search (A ; Key)

found $\leftarrow 0$

```
for (i = 0 to N)
    if A[i] == Key
```

 found $\leftarrow 1$

print "Element found"

break

if found == 0

print "Element not found"

 \rightarrow Time complexity = $O(n)$ \rightarrow Space complexity = $O(1)$

• Binary Search (A, beg, end, Key)

while beg \leq end

mid = beg + (end - beg) / 2

if mid == key

return mid

↑ Iterative

if A[mid] < key

beg = mid + 1

if A[mid] > key

end = mid - 1

return -1

→ Time complexity - $O(\log n)$

→ Space complexity - $O(1)$

- Binary search (A , beg, end, key)

if $\text{end} > \text{beg}$

$\text{mid} = (\text{beg} + \text{end}) / 2$

if $A[\text{mid}] \leq \text{item}$

return $\text{mid} + 1$

else if $A[\text{mid}] < \text{item}$

return binary-search (A , $\text{mid} + 1$, end , key)

else

return binary-search (A , beg , $\text{mid} - 1$, end)

return -1

→ Time complexity - $O(\log n)$

→ Space complexity - $O(1)$

$$24) T(n) = T\left(\frac{n}{2}\right) + c.$$