1. Time And Space Complexity

08 June 2024 12:20

what?
why?

DSA

DSA → DS

DSA → Algo

DS → DS are diff. types of structures to store the data

Algo → Is a well defined sequence of steps to solve a problem. Will have a I/P, O/P.

Eg:- unsorted → $\boxed{\text{sorting algo}}$ → sorted

DS, Algo → $\boxed{\text{efficiency}}$

efficiency → Time

efficiency → Space (memory)

imp. → Time

Time → what? → I/P size

Time → How?

what?

- Comp. speed. ×
- how much loaded ×
- environmental

How?

- × Exact time
  - Eg:: → 0.932 sec
  - 0.964 sec
  - 0.899 sec
- in terms of I/P size → (T)

→ Write a program to calculate sum

→ Write a program to calculate sum of first n natural numbers.

n

m-I :-

sum = 0
for (i=1; i<=n; i++)
    sum = sum + i
print (sum)

Work done :-

$an+b$

m-II :- $sum = \frac{n(n+1)}{2}$

Work done :-

a

m-III :-

1 + 2 + 3 + 4 + 5 → (1+1+1+1)

(1) (1+1) (1+1+1)

2+3+4 —— +(n+1)

1+2+3 —— +n

```
sum = 0
for(i=1; i<=n; i++)
{  for(j=1; j<=i; j++)
   { sum++;
   }
}
```

Work done :-

an^2+bn+c

II < I < III

I is better than II.

eg :- 3n+10 ...

eg:- $3n + 10$ ... I ...
$5n^2 + 8$

→ $10n + 25$ → ($n = 5$) 75   I is better than II

→ $0.1n^2 + 2$ → 4.5

Order of growth :-

$10n + 25$ → n ✓

$0.1n^2 + 2$ → n^2 ✓

$3n^2 + 8n + 7$ → n^2 ✓

→ $3n^2 + 8n + 7$ → n^2

→ $4n^2 + 3n + 2$ → n^2 → same

⇒ **Asymptotic Analysis** :-

Analysis of f^n as I/P reaches ∞.

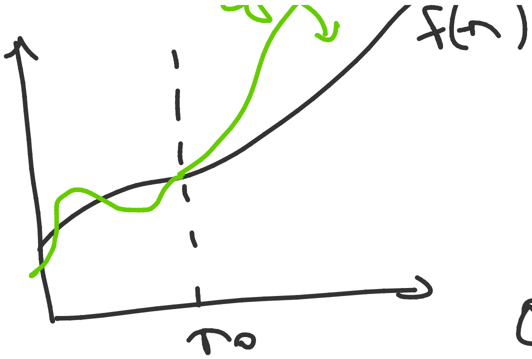
Asymptotic Notations :-

O Ω Θ → upper bound

① **Big O Notation** :- $f(n), g(n)$

→ $O(g(n))$: $\{ f(n) \mid \exists$ +ve constants c, n_0 s.t $0 \le f(n) \le c\,g(n), \forall n \ge n_0 \}$.

eg:- $n^2 + 5n + 7 \in O(n^2)$



eg:- $\underbrace{3n^2+5n+7}_{f(n)} \in O(\underbrace{n^2}_{g(n)})$

$0 \le 3n^2+5n+7 \le cn^2, \forall n \ge n_0$

$0 \le 3n^2+5n+7 \le 3n^2+5n^2+7n^2$

$3n^2+5n+7 \le 15n^2, \forall n \ge 1$

$\boxed{c=15,\ n_0=1}$

$n-1 \rightarrow an+b$

$TC = O(n)$

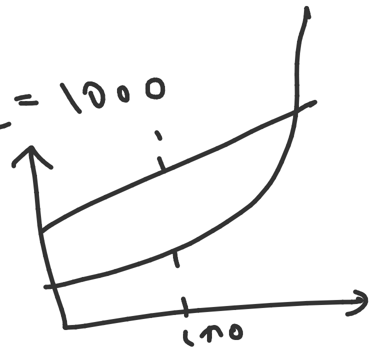
$\rightarrow 3n^2+7n+5 \in O(n)$

$0 \le 3n^2+7n+5 \le cn$

$0 \le 3n^2+7n+5 \le 1000n$

n_0 ✗

C = 1000



$\rightarrow 3n^2+5n+7 \in O(n^3)$ ✓

$0 \le 3n^2+5n+7 \le cn^3$

$\boxed{c=15,\ n_0=1}$ $\rightarrow O(n^3)$

$3n^2+7n \rightarrow O(n^2)$

$4n^2+8 \rightarrow O(n^2)$

$an^2+bn+c \rightarrow O(n^2)$

$an+b \rightarrow O(n)$

$a \rightarrow O(n^0) \rightarrow O(1)$

$O(n^0) = O(1)$

$\boxed{n^0 = 1}$

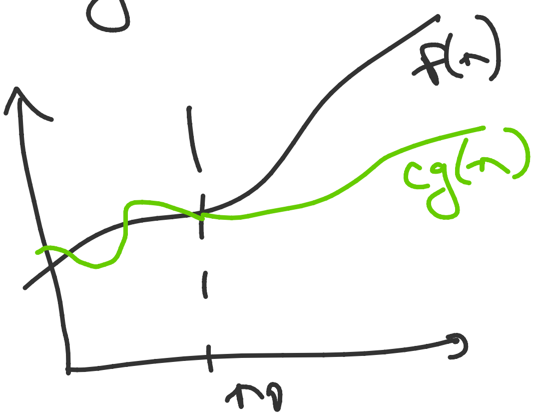
$n^1 = n \quad n^2 = n \times n$

$\boxed{n^0 = n^{b-b} = \frac{n^b}{n^b} = 1}$

(Ω) lower bound

⇒ Big Omega Notation (Ω) → lower bound

$\Omega(g(n)) : \{ f(n) \mid \exists$ +ve constants c, n_0 s.t $0 \le c g(n) \le f(n), \forall n \ge n_0 \}$



eg:- $3n^2+5n+7 \in \Omega(n^2)$

$0 \le cn^2 \le 3n^2+5n+7$

$(c=1, n_0=1)$

$0 \le n^2 \le 3n^2+5n+7$

eg:- $3n^2+5n+7 \in \Omega(n^3)$ ×

$0 \le cn^3 \le 3n^2+5n+7$

eg:- $3n^2+5n+7 \in \Omega(n)$

$0 \le cn \le 3n^2+5n+7$ $(c=1, n_0=1)$

⇒ Q/. Check whether 0 is present in array.

I/P → n

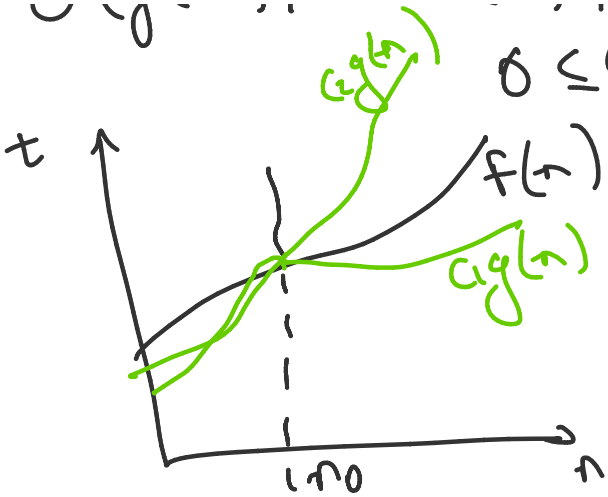
$\Omega(1)$ $O(n)$

⇒ Theta (Θ) Notation :- → exact bound → tight bound.

$\Theta(g(n)) : \{ f(n) \mid \exists$ +ve constants c_1, c_2, n_0 s.t $0 \le c_1 g(n) \le f(n) \le c_2 g(n), \forall n \ge n_0 \}$

$c_2 g(n)$

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n), \ \forall n \ge n_0$$



eg:- $3n^2+5n+7 \in \Theta(n^2)$

$0 \le c_1 n^2 \le 3n^2+5n+7 \le c_2 n^2$

$c_1 = 1, c_2 = 15, n_0 = 1$

eg:- $3n^2+5n+7 \in \Theta(n)$ ✗

eg:- $3n^2+5n+7 \in \Theta(n^3)$ ✗

→ If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ then $f(n) \in \Theta(g(n))$ and vice versa is also true.

→ Note :-

① $3n^2+5n+7 \in O(n^2)$ (ideally)

$3n^2+5n+7 = O(n^2)$ ✓

$O(n^2) = 3n^2+5n+7$ ✗✗

→

Worst case	Avg case	Best case
→ 1.3 → worst TL=1 ✗ TL=1.5 ✓	difficult to calculate ✗	→ 0.7 ✓ → TL → use not good estimation

x ✓ | X | not good estimation

Eg:- Check whether 0 is present in array.

Best case → 1 step

worst case → n steps.

avg case →

array → n = 5 $0 \le a[i] \le 7$ $\frac{\sum steps[i]}{Total\ calls}$

8 8 8 8 8 → 8^5

20 40 30

$\frac{20+40+30}{3} = 30$

n → worst case → $\Omega(n)$

algo → $\Omega(n)$ X

algo → $\Omega(1)$

best case → O(1)

algo → O(n) ✓

algo → O(1) X

O(n)

Algo → TC → best

⇒ Some standard loops & their TCs:-

① for(i=1 ; i<=n ; i++)   TC = O(n)

{   //constant

② for(i=1; i<=n; i=i+K) $\quad K<<n$

{

// constant $\quad TC = O\left(\frac{n}{K}\right) = O(n)$

}

③ for(i=1; i<=n; i=i×2)

{

// constant $\quad n=32$

$i = 1, 2, 4, 8, 16, 32, 64$

}

$i = 2^0, 2^1, 2^2 \text{------} 2^K$

$2^K \le n \quad K \le \log_2 n$

$TC = O(K)$

$= O(\log_2 n)$

④ for(i=n; i>=1; i=i/2)

{

// constant

}

$i = 2^K, 2^{K-1} \text{---} 2^0$

$2^K = n \rightarrow K = \log_2 n$

⑤ for(i=1; i<=n; i++) $\quad i = 1, 2, 3 \text{------} n$

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

(5) for(i=1; i<=n; i++)

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

{ // constant

}

}

i=1, ... ↓ ↓ ↓ ... ↓ n n n ... n

TC = $O(n^2)$

(6) for(i=1; i<=n; i++)

{ for(j=1; j<=i; j++)

{ // constant

}

}

i = 1, 2, 3 ——— n

↓ ↓ ↓ ——— ↓

1 + 2 + 3 + ——— + n

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

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

{ // constant

}

for(i=1; i<=√n; i++) → $O(\sqrt{n})$

{

}

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

{

}

$O(\sqrt{n})$

i = 2, 4, 16, 256 ----

$2^{2^0}, 2^{2^1}, 2^{2^2}, 2^{2^3}$ ---- 2^{2^p}

16 × 16 = 256

16
96
160
256

$2^{2}, 2^{2}, 2, 2 \ldots \ldots$

(P) $2^{K_1}, 2^{K_2} \text{———} 2^{K}$

$2^{K} = M \quad K = \log M$

$\rightarrow 2^{P} = \log M$

$P = \log \log_2 M$

$\underline{TC = O(\log \log_2 M)}$

⇒ <u>Space Complexity</u> :-

$a[n] \rightarrow O(n)$

int a, b, c, d ;
$O(1)$

$\text{int } a[n][n] \rightarrow O(n^2)$

space req. to store I/P + Auxilliary space

Auxilliary space ↓ extra space required to solve the problem.

eg:- I/P → int a[n]
sum = 0
(sum of array ele)

$I/P \rightarrow O(n)$
$AS \rightarrow O(1)$
$SC \rightarrow O(n)$

eg:- int a[n]
int dp[n][n]

$I/P \rightarrow O(n)$
$AS \rightarrow O(n^2)$
$SC \rightarrow O(n^2)$

⇒ Eg:- Given an array a, reverse it.

$a = [1,2,3,4,5] \to [5,4,3,2,1]$

for i=0 to n-1: $b = [5,4,3,2,1]$

$b[i] = a[n-i-1];$ $a = [5,4,3,2,1]$

for i=0 to n-1: $I/P \to O(n)$

$a[i] = b[i];$ $AS \to O(n)$

$SC \to O(n)$

$a = [1,2,3,4,5,6,7]$ $n=7,\ i=3$

$[7,6,5,4,3,2,1]$

$[1,2,3,4 | 5,6,7,8]$ $n=8,\ i=3$

for(i=0; i<=(n-1)/2; i++)

{ swap(a[i], a[n-i-1]); $AS = O(1)$

→ (a, b) → swap

c = a

a = b

b = c

$a = 5,\ b = 10$

$a = a+b \to 15$

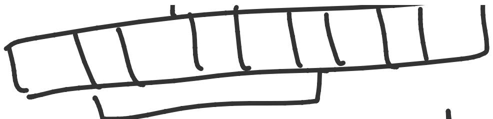
$b = a-b \to 5$

$a = a-b \to 10$

⇒ Terminologies :-

① Subarray :-

① Subarray :-



→ It is a continuous segment of the array.

→ It is obtained when you delete some/none of the elements from start & end of array.

eg :- a = [1, 2, 3, 4, 5]   [1, 2, 3, 4, 5] ✓

[2, 3] ✓   [1, 2, 4] ✗

[3, 4, 5] ✓   [3, 2] ✗

Array → n , # subarray = ?

[1, 2, 3, 4]

[1, 2, 3, 4] → 1

[1, 2, 3] [2, 3, 4] → 2

[1, 2] [2, 3] [3, 4] → 3

[1] [2] [3] [4] → 4

n → n, n−1, n−2, ..., 1

$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$

[l, r] $a = [1, 2, \underbrace{5, 8, 10}, 12, 15]$

$l = 2, r = 4$

② Subsequence :-

→ when you delete some/none of the elements from the array, the remaining portion is called a subsequence.

portion is called ...

eg:- [1,2,3,4,5] [1,2,3,4,5] ✓ [3,1] ×

[1,3] ✓ [ ] ✓

[2,3,4] ✓ [2,4] ✓

→ Order of elements must be same as original array.

Array → n, # subsequence = ?

a = {1,2,3}

{1}, {2} {3}

{ }, {1,2}, {1,3}

{2,3}, {1,2,3}

$\underline{2}\ \underline{2}\ \underline{2} \ldots \underline{\quad}\ \underline{\quad}\ \underline{2} \rightarrow 2^n$

⇒ <u>Prefix</u>:- It is a subarray whose starting point is same as original array.

a = [1,2,3,4,5]

→ [1], [1,2], [1,2,3] [1,2,3,4] [1,2,3,4,5]

⇒ <u>Suffix</u>:- It is a subarray whose ending point is same as original array.

a = [1,2,3,4]

→ [4], [3,4] [2,3,4] [1,2,3,4]

⇒ a = [5, 3, 2, 1, 9, 7, 10]

[5, 5, 9, 9, 10]

=> a = [5, 3, 2, 1, 9, 7, 10]

pre_max = [5, 5, 5, 5, 9, 9, 10]

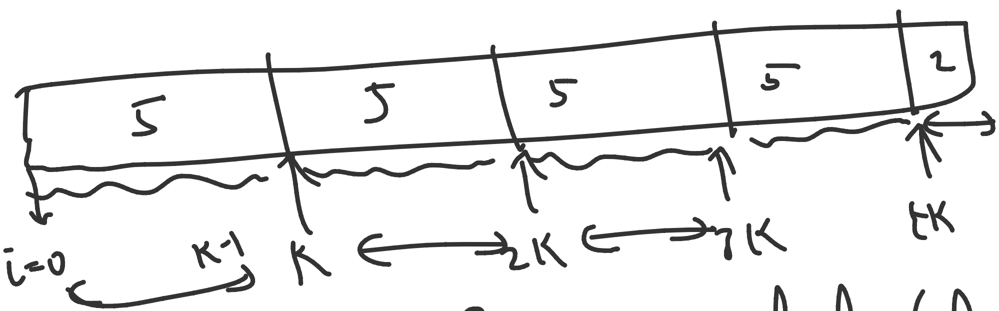
pre_max(i) → maximum element of prefix ending at i.

pre_sum = [5, 8, 10, 11, 20, 27, 37]

[5, 3, 2, 1, 9, 7, 10]

=> for i = 0 to n:
{
  → O(3n)
}
  → O(n)
for i = 0 to n:
{
}
for i = 0 to n:
{
}

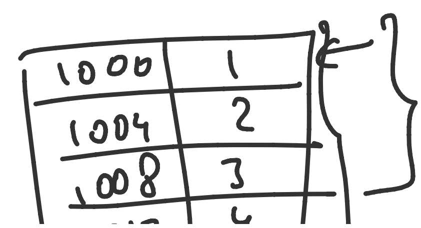
n = 22, K = 5



[1, 2, 3, 4, 5]

while(l < r)

Pass by value



Pass by value

main( )

{ arr

}

reverse (arr)

{

}

1004	
1008	3
1012	4
1016	
1020	
1024	1 → 4
28	2 → 3
32	3 → 2
36	4 → 1

a = [1, 2, 3, 4, 5]

[5, 1, 2, 3, 4]

[4, 5, 1, 1, 2, 3]

[3, 4, 5, 1, 2]

[2, 3, 4, 5, 1]

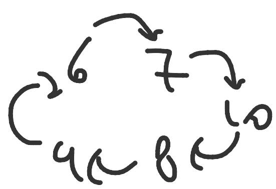
4, 5

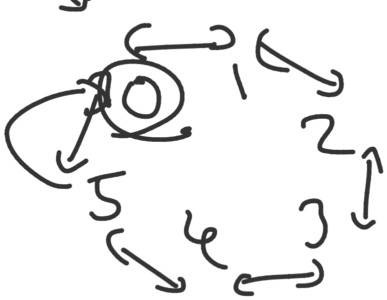
6, 7, 10, 8, 9

[1, 2, 3, 4, 5]

6, 7, 10, 4, 9

arr[i] > arr[(i+1) % n] → 1



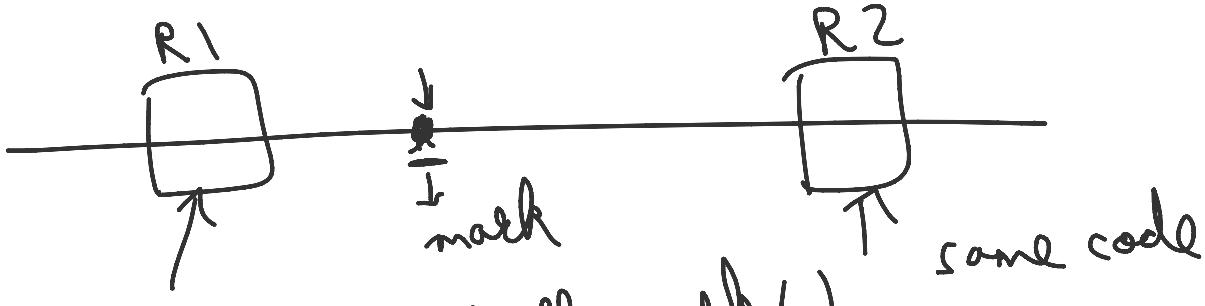


n = 6

(i+1) % n → n = 6

i = 2 → 3

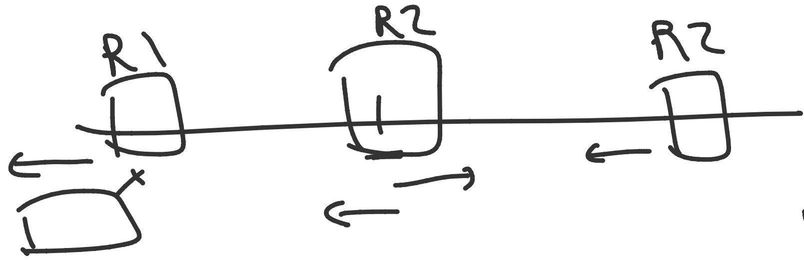
i = 5 → 0



call_mark( )

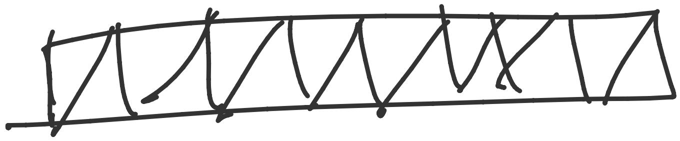
ML-1() MR-1 cross_mark()

ML-2() MR-2 cross_robot()




```
while (cross_robot() != F)
{
  if (cross_mark())
    ML-2();
  else
    ML-1();
}
```

array :-



sum = 0 → [~~2~~, ~~4~~, ~~1~~, ~~3~~, ~~8~~]

sum = 4+3+1 = 8