lec-1

* Data Structures and Algorithms

Introduction to Data Structures & Algorithms:

1. Placement Prep.
2. C/C++

Main Memory → RAM

Data structure → way to arrange data in main memory for efficient usage

Ex: Arrays, ~~Stru~~ linked list, stack, Queue, BST etc.

Algorithms: → Sequence of steps to solve a given problem.

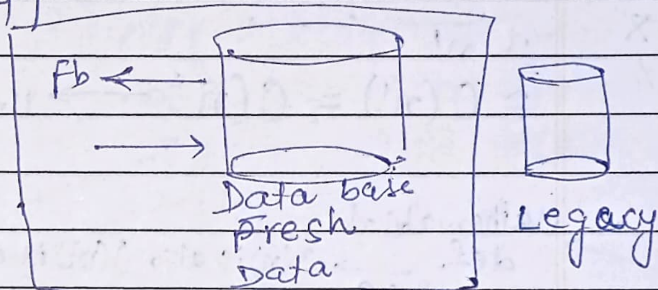
e.g. Sorting an Array
[1, 7, 9, 2]
↓
[1, 2, 7, 9]

Read, Retrieve, update

Data base: HDD

Data warehouse: Legacy data.

Big data:



memory layout of C programs

Heap:
Stack
Static+global
Code segment

Heap → Imp

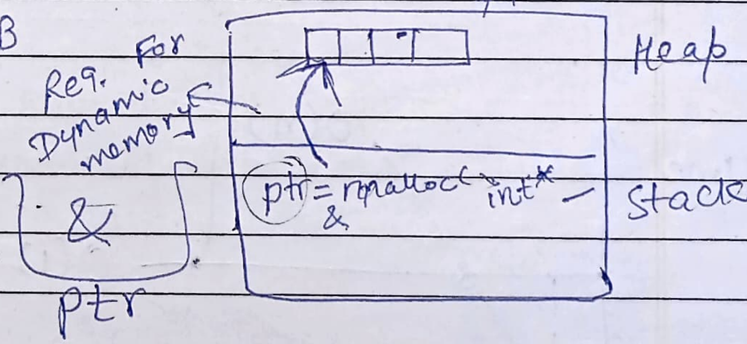
Stack → Main()

fn1()

Ram 16GB

```
fun1() {
  k = fun2();
}
Main() {
  fun1(); ←
}
```

Dynamic memory Allocation



Database:- collection of Information in parmanent storage for faster retrievel and updating.

Data Warehouse:- Management of huge data of Legacy data for better analysis.

Big Data:- Analysis of too large or complex data, which cannot be dealt with the traditional data processing applications.

(2) Time Complexity and Big O Notation: $\boxed{\text{Input}}$

Algo 1.

250 Kb → Internet

60 GB → physical visit HDD

Algo 2.

l_2 Kb/s

t_3 Kb

n kb

$$t^{(n)}_{algo2} = \underbrace{k_1 + k_2 + k_3 + k_4}_{1}$$

$$= k_1 n^0 + k_2 + k_3 + k_4$$

$$\cong n^0$$

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

n → Size of game.

Big O → constant

$$t^{(n)}_{algo1} = l_1 + \cancel{l_2}\left(\frac{n}{l_2}\right)$$

$$= l_1 + l_2' n$$

$$= \underline{l_1 n^0} + \underline{l_2 n^1}$$

$$\cong n^1$$

$$\cong O(n^1) = O(n) \longrightarrow \text{linear}$$

$\boxed{t = \frac{\text{Dist.}}{\text{speed}}}$

$S = \frac{D}{t}$

$t = D/s$

$O(n \log n)$

$O(n^2)$

$O(n^3)$

$O(n!)$

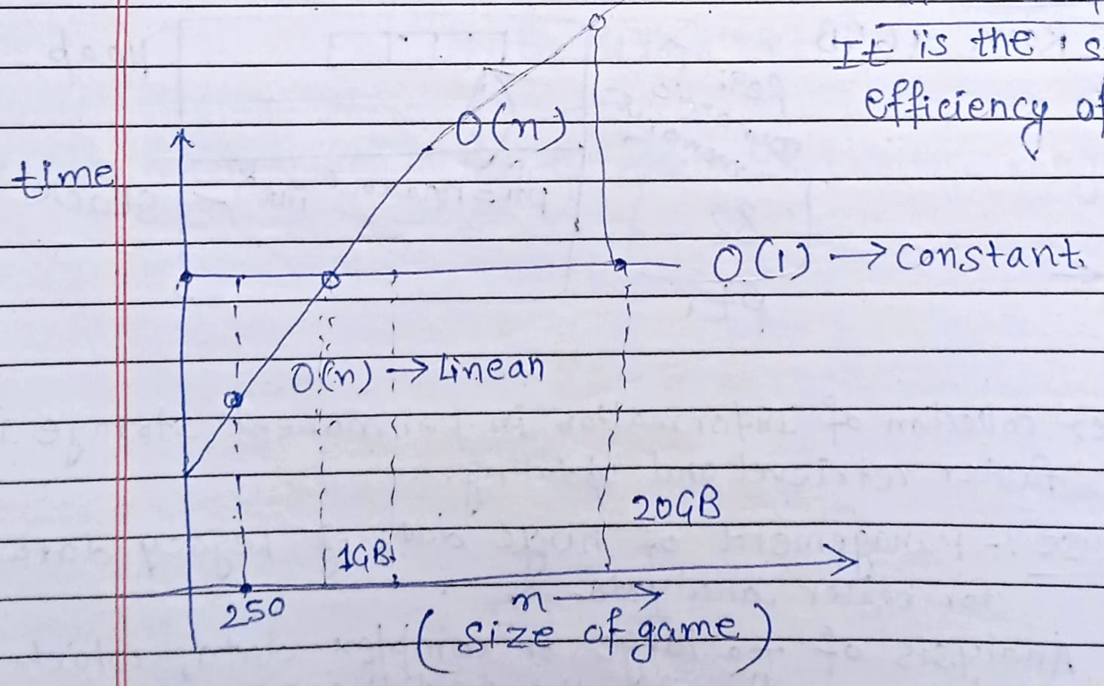
O → Mathematical def. Big O → O(n) is also O(n²) is also O(n³)

→ Industry Def. → order of

Minimum of these.

Time complexity

It is the study of efficiency of Algorithm.



③ Asymptotic Notations : Big O, Big Omega, & Big theta Explained:

Asymptotic Notations:

→ Big O
→ Big Omega
→ Big Theta

Big O :- A Function $f(n)$ is said to be $O\,g(n)$ if and only if there exists constant c & a constant n_0 such that (st.)

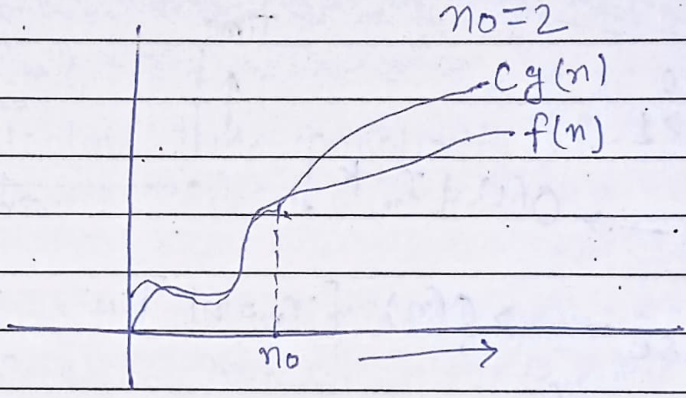
$$0 \le f(n) \le c\,g(n) \quad \forall \; n \ge n_0$$

$f(n) = n^3 + 1$
$g(n) = n^4$

$0 \le n^3 + 1 \le 1\,n^4$

$C = 1$
$n_0 = 2$

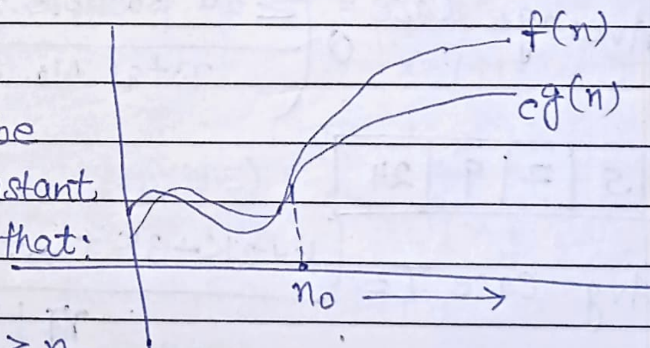
$f(n) \to$ Time
$n \to$ Input
$g(n) \to O(g(n))$



Big Omega → Big Ω :-

A function $f(n)$ is said to be $\Omega\,g(n)$ if there exist a constant C and constant n_0 such that:

$$0 \le c\,g(n) \le f(n) \quad \forall \; n \ge n_0$$



$f(n) = n^3 + 1$
$\Omega(1)$ - $0 \le c \le n^3 + 1$
$\therefore \; 0 \le 1 \le n^3 + 1$ $n_0 = 1$

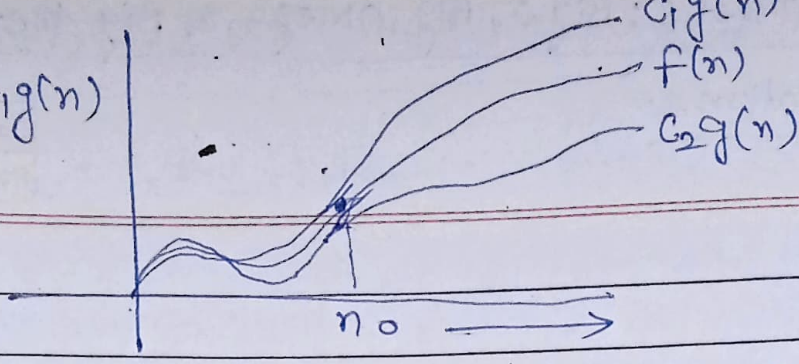
Big theta → θ

A function $f(n)$ is said to ~~be~~ be $\theta\,g(n)$ iff there exists constant c & a constant n_0 st

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

$c_2 g(n) \le f(n) \le c_1 g(n)$

$c_2 g(n) \le f(n) \le c_1 g(n)$



(4) Best Case, worse Case and Average Case Analysis of an Algorithm (with Notes):

Algo 1'

1	2	3	9	24	7

Sorted Array → arr [n] ⟶ n comparison

Algo 1 ⟶

1	2	3	5	7	24

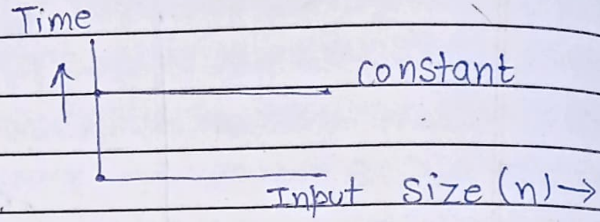
a

$a = 9 \rightarrow 0$

$a = 7 \rightarrow 1$

a=1 ⋮

Best case ⟶ $O(1)$ $[T_n = k]$



Worst case ⟶ $O(n)$ $[T_n = nk]$

Average Case $= O\left(\dfrac{\sum \text{all possible run times}}{\text{Total No. of possibilities}}\right)$

1	5	7	9	24

Avg case $T = \dfrac{[k + 2k + 3k + \text{-----} \; nk + nk]}{n+1}$

$= \dfrac{k[(1+2+3+ \text{---} +n) + n]}{n+1}$

$1+2+3+4+ \text{---} +n = \dfrac{n(n+1)}{2}$ A.P

GP & AP = Imp

$= \dfrac{k\left[\dfrac{n(n+1)}{2} + n\right]}{n+1} = k\left[\dfrac{n^2 + n + \cancel{2n}}{2(n+1)}\right]$ non dominate

$= k\,\dfrac{n(\cancel{n+1})}{2(\cancel{n+1})} = \dfrac{kn}{2}$

$= O(n)$

mid point

Algo 2 | 1 | 5 | 7 | 9. | 24 | 32 |

(a) ⑦ 1, 2, 3, ---- 12 ---- 25 ---- 50 ---- 100

Best Case ⟶ O(1)

worst Case ⟶ O(log(n))

O(log n)

8 → 4 → 2 → 1

8 → 3

16 → 8 → 4 → 2 → 1

16 → 4

2 → 1

$\log m^n = n \log m$

2 → 1

log 8 → 3
log 16 → 4
log 2 → 1

Lec.

⑤ How to Calculate Time Complexity of an Algorithm + Solved Questions

log

Trickes :

1. Drop the Non-dominant terms:
2. Drop the Constant Terms.
3. Break the code into Fregments.

```
int i
int k=0          ] → K1
for(i=0; i<n; i++){
    k=i;
    k++;          ] K2
    x=y+1.
}
```

$T_n = K_2 n + \cancel{K_1}$

$= O(n)$

```
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++){
        k=i
        k++;
        x=y+1;
    }
}
```

i	j
0	0
0	1
0	2
⋮	⋮
0	n
1	0
⋮	

i=0+1+... n

n+n+n+...n

n(1+1+1+1+...1) (n-1)

n × (n)

⇒ $\boxed{n^2}$

$T_n = \cancel{K_1} + K_2 n^2 +$

$T_n = K_2 n^2$

$T_n = O(n^2)$

* Time Complexity – Competitive Practice sheet: → onenote

1. Find the Time Complexity of the func1. function:

```
#include <stdio.h>

void func1 ( int array[], int length)
{
  int sum=0;              ]  f1 = k1
  int product=1;          ]
  for ( int i=0; i< Length; i++)   ]
  {                                 ]  f2 = k2 n
     sum += array [i];              ]
  }                                 ]
  for (int j= 0; i<length ; i++)    ]
  { product *= array[i]             ]  f3 = k3 n
  }                                 ]
}
int main ()
{
  int arr[] = {3,5,66}
  func1 (arr,3);
  return0;
}
```

$T_n = f_1 + f_2 + f_3$

$T_n = \cancel{k_1} + k_2 n + k_3 n$

$T_n \sim (k_2 + k_3) n$

$\sim k_4 n$

$\sim O(n)$

$\sim O(\text{length})$

2. Find Time Complexity of func function:

```
void func (int n)
{
  int sum=0;            ] → k1
  int product=1;        ]
  for (int i=0; i<n; i++) → 0 to n
  {
    for (int j=0; j<n; j++) → 0 to n
    {
      printf("%d, %d", i, j);  → k2
    }
  }
}
```

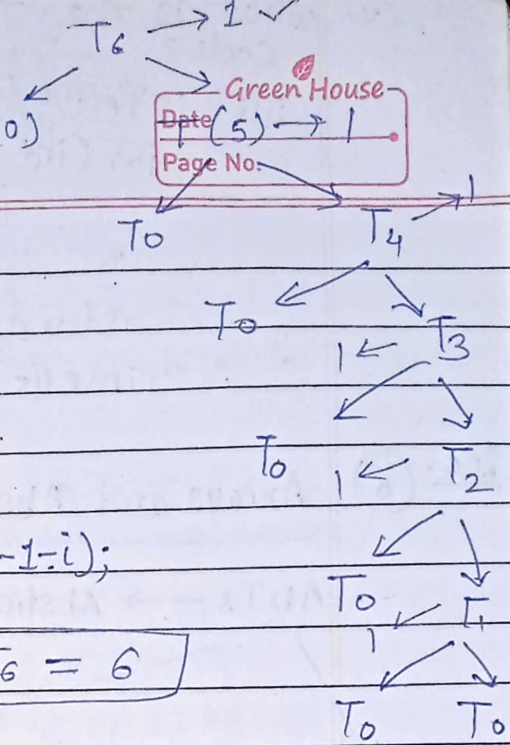
$[n + n + n + \ldots (n-1) n] k_2$

$n k_2 [1 + 1 + 1 + \ldots + 1]$

$n k_2 (n) = k_2 n^2 \sim O(n^2)$

(3) Find T(6)

```
int function(int n)
{ int i;   ]──────→ K₁=0
  if (n<=0)
  {
    return 0;
  }
else
  {
    i = random(n-1);  ──────→ 1
    printf("this \n");
     return function(i) + function(n-1-i);
  }
}
```



$\boxed{T_6 = 6}$

(4) which of the following are equivalent to O(N)? why?

a) O(N+P) where P<N/9 ⟶ O(N)

b) O(9N-k) ⟶ O(9N) ⟶ O(N)

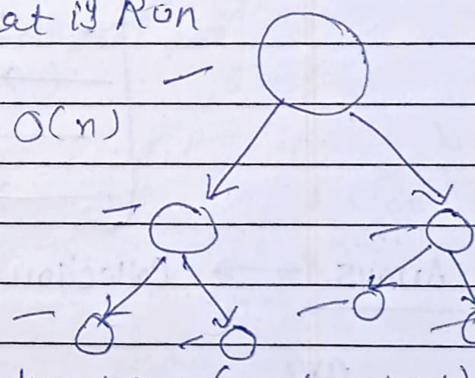
c) O(N+8logN) ⟶ O(N)

× d) O(N+M²)

(5) The following code sum all the values of all the nodes in a balanced binary search tree, what is Ron Time?

```
int sum(Node node)
{
  if (node == NULL)
   {
     return 0;
   }
  return sum(node.left) + node.value + sum(node.right)
}
```

O(n)



(6) Find the Time Complexity of following code which Tests whether a given no. is prime or not?

```
int isPrime(int n){
       if (n==1){        ]──────→ K₁
              return 0;
       }
     for (int i=2; i*i<=n; i++){
          if (n%i==0)  ] K₂
          return 0;
     }
     return 1;
}
```

$K_1 + K_2(\sqrt{n})$

$O(\sqrt{n})$

i=2
i=3
|
$i = \lceil \sqrt{n} \rceil$

$O(\sqrt{n})$

7. what is the Time Complexity for following snippet of Code?

```
int isPrime (int n){
    for (int i = 2; i*i<1000; i++) {
        if (n%i==0)
            return 0;
    }
    return 1;
}
isPrime (15);
```

$T_n = K_1 \longrightarrow \Theta(1)$

Lec. 6 **Arrays and Abstract Data Type in Data Structure:**

ADTs ⟶ Abstract Data Types — ADT's are the way of classifying data structures by providing a minimal expected interface & set of Methods.

Operations

Minimal Requirements

MRF ⟶ Minimal Requirement Functionality Required.

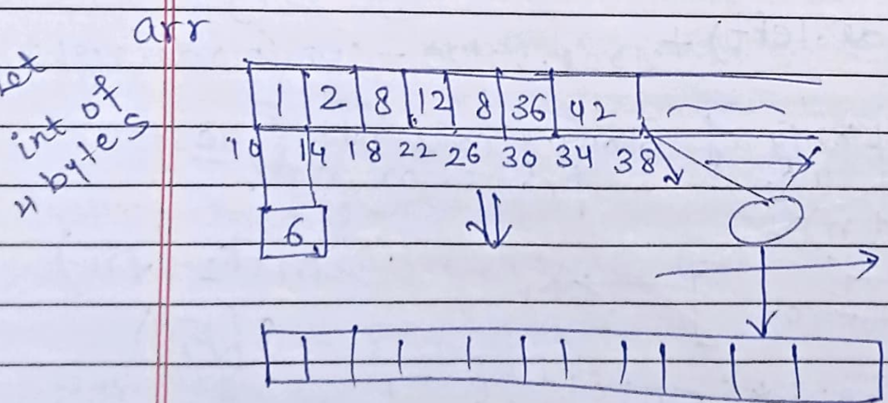
Arrays ⟶ [get (i), set (i, num)]

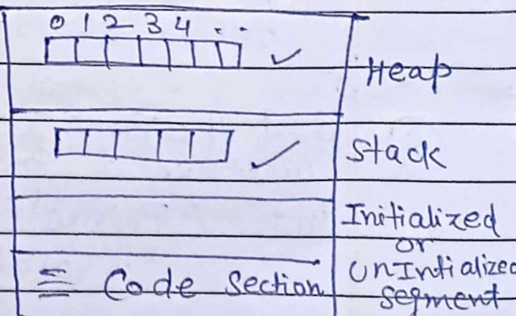
Hiding the details

Methods/Operations
- ⟶ Insert
- ⟶ delete
- ⟶ Add
- ⟶ Resize.

Abstraction ⇓ Implimentation ✗ Usage ✓

Arrays ⟶ Collection of elements accessible by an index.





int * a ⟶ (int *) malloc (10 * sizeof (int))