Data Structure & Algorithms

Data :- Information → Real Time Entity

Structure :- Skeleton → Need - organize in a fashion
- use
- very easily can fetch
- update / delete

Algorithms :- set of rules / logic to perform particular operation

Data

name
value
id

no - int, long, double, float
name, char :- String
boolean - true / false

Structure :-

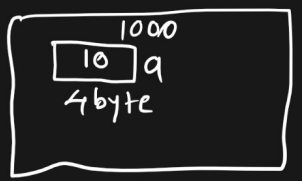
Linear

1) data type/variable
2) Array
3) Stack
4) Queue
5) Linked List

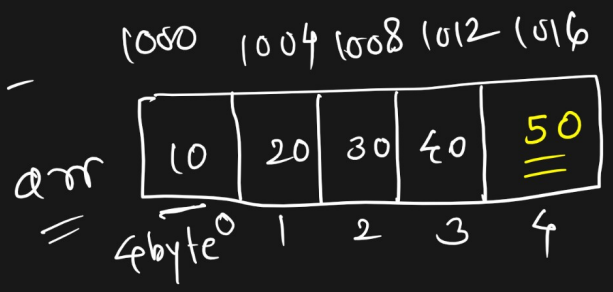
Non Linear

1) Tree
2) Graph

Variable :- int a = 10;



Array :-



arr[4] = 50

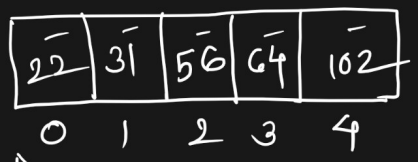
JAVA

a_1
a_2
a_3
a_4
⋮

int arr[] = new int[size] ⟹ static memory allocation

Decl^r { int arr[] = {10, 20, 30, 40}

Stack :-



arr[0] = 22 ✓
arr[1] = 31 ✓
arr[2] = 56 ✓
arr[3] = 64 ✓
arr[4] = 102 ✓

initialize { }

for(int i=0; i<5; i++) { arr[i] = ... } → 5 time 0 to 4

22 31 56 64 102

⇒ 102 64 56 31 22 → Stack ⇒ LIFO

Stack Implementation

Array LinkedList

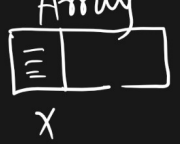
Queue ⇒ FIFO → Array – Fixed
→ Linked List – → Not fixed the size

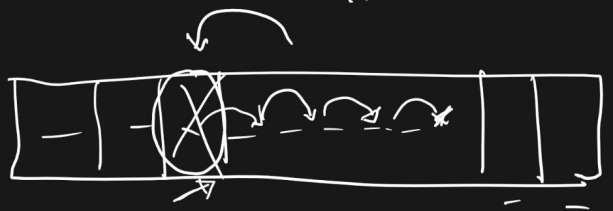
Appln :- Stack → LIFO
A) purchase order
B) Message / mail
C) Social Appln
D) Browser History
E) undo

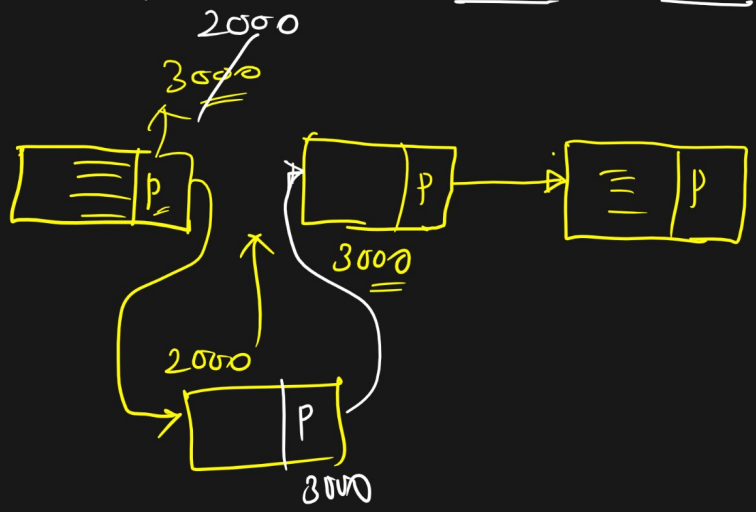
Queue Appln → FIFO
A) Ticket Booking
B) Task Scheduling
C)

Linked List :- Advantage :- 1) Dynamic Memory Allocation
2) store the multiple type of info
3) Operation insert / delete

Info with different data types
employee :- id – no
name – string
Add –

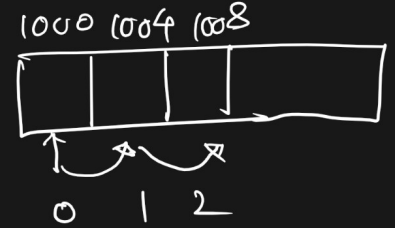


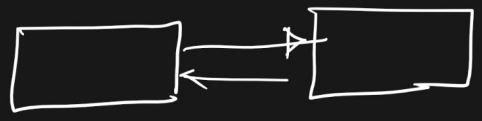




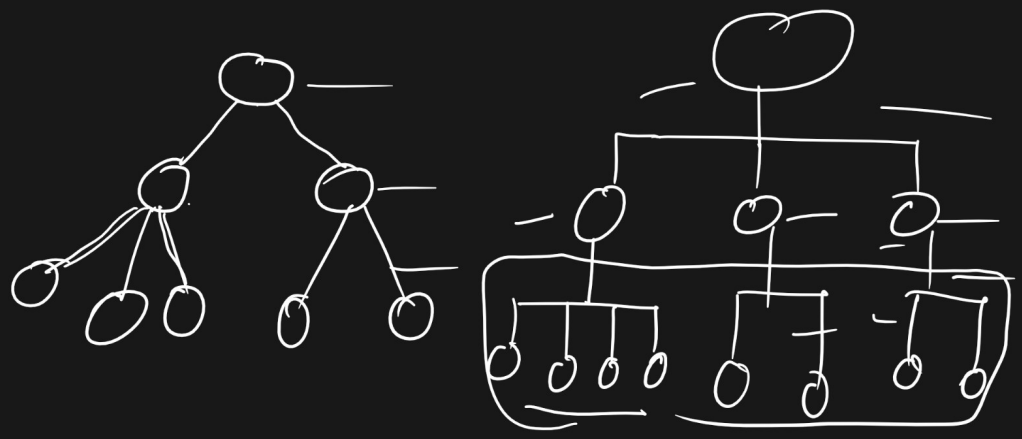
Appln :- 1) Music Play List
2) Tree
3) Graph
4) map

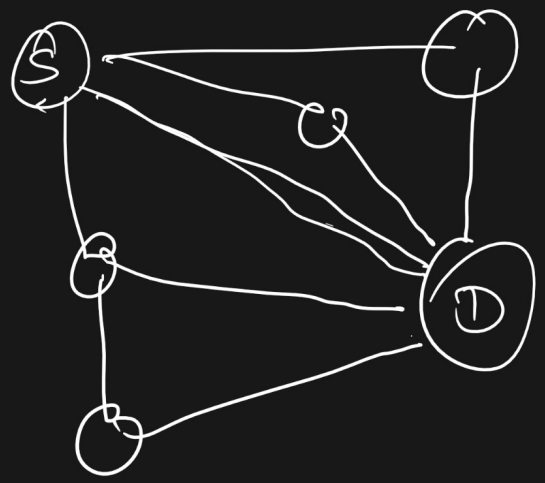
int arr





Non Linear DS





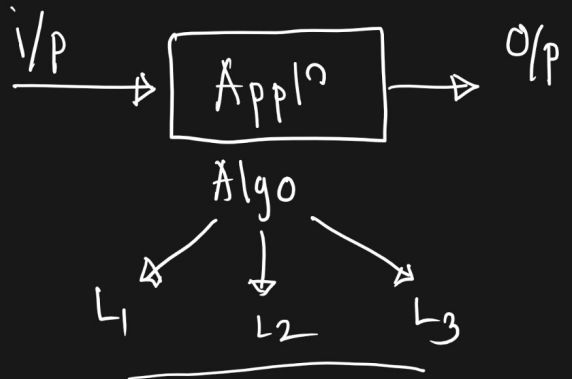
Appln :- map

Networking

Algorithms :-

1) Store the info
2) fetch the info
3) update / delete

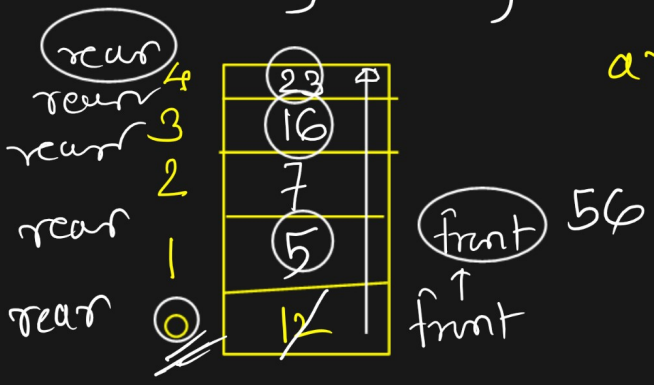
} Manage our data & structures by using algo

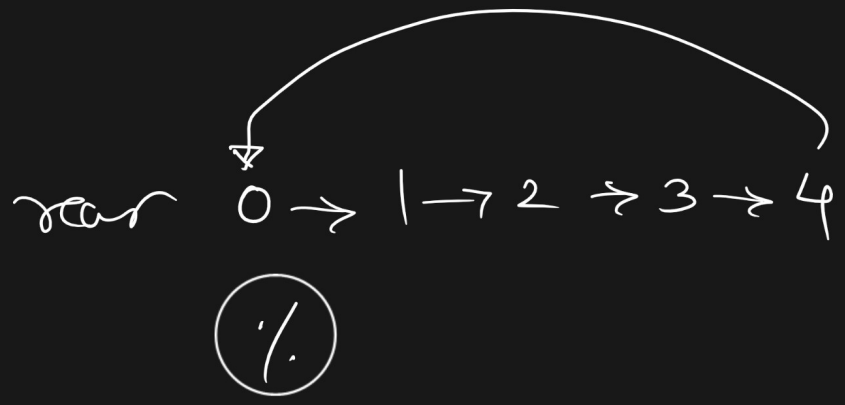


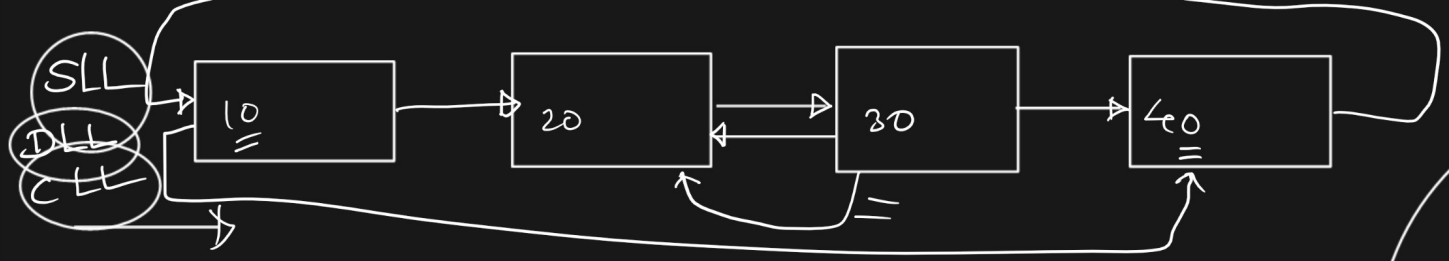
why we need different type of Algo :-

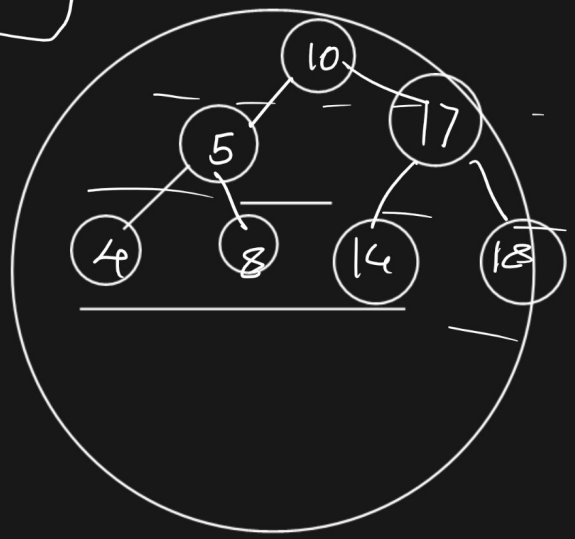
Queue using array :- FIFO

arr[5]







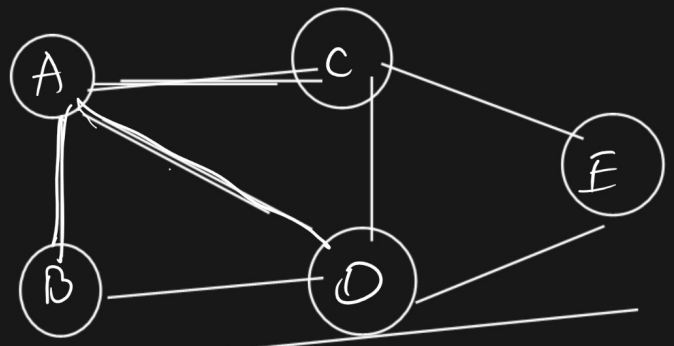


Preorder → 10 5 4 8 17 14 18

Inorder → 4 5 8 10 14 17 18

Postorder → 4 8 5 14 18 17 10

Levelorder 10 5 17 4 8 14 18



DFS A B D E C

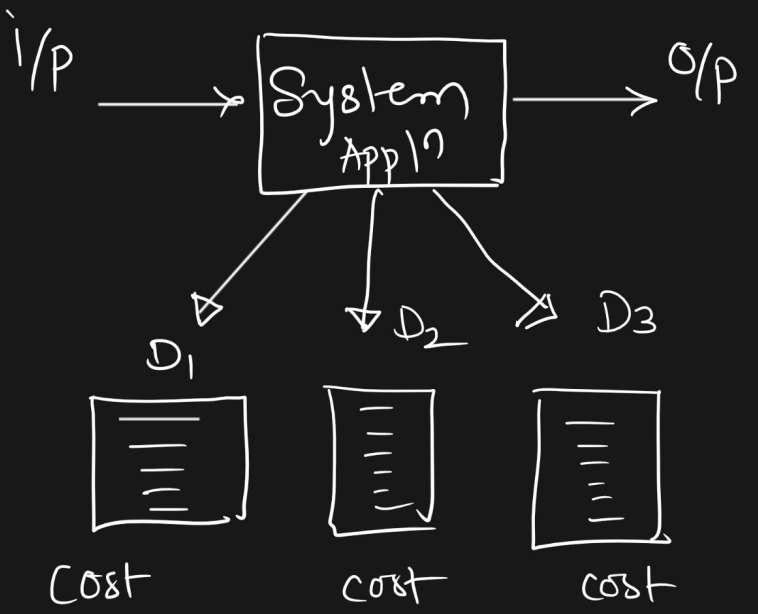
BFS A B C D E

A B D E

→ Algorithms

Algorithms Analysis

client



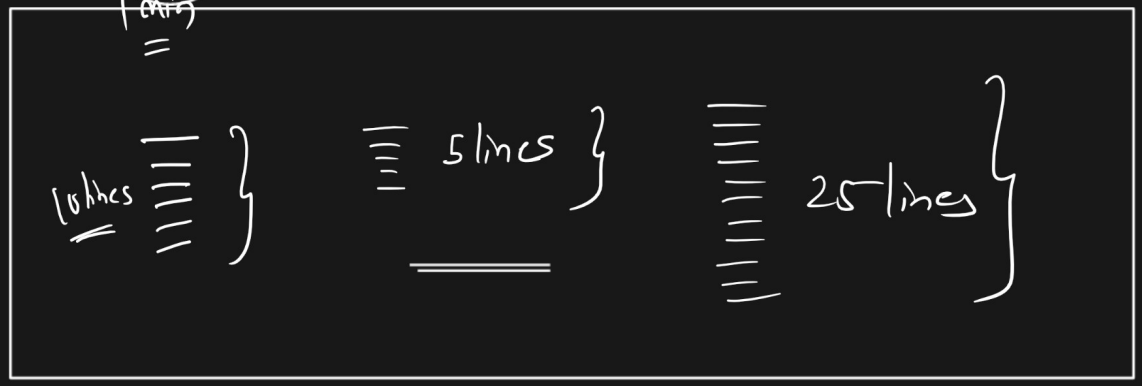
client :-

1) Cost → x
2) Cloud / Spuex
3) Time ✓

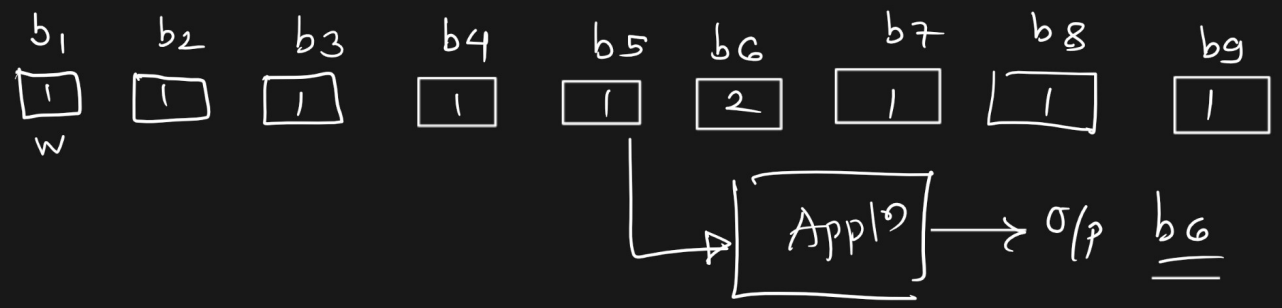
$100 — 1GB (min)

Free — 500MB

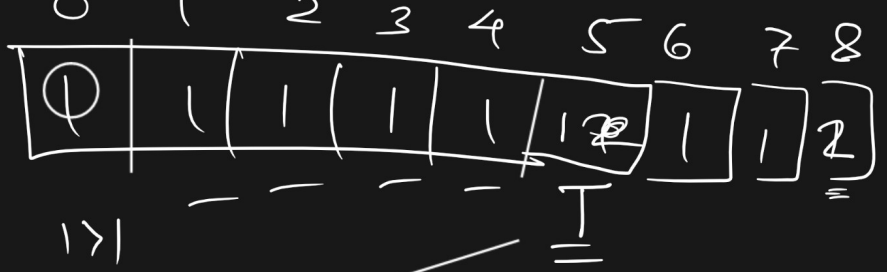
$45 → 5min — 0.5GB



5 lines :- D2



box [1, 1, 1, 1, 1, 2, 1, 1, 1]




```
int result = box[0]
for(int i=0; i<9 ; i++)
{
    if (box[i] > 1)
    { return i;
    }
}
```

5 lines

1 sec

1 sec $1>1$, $1>1$

8 times Comparison

8 sec

---

int A = box[0] + box[1] + box[2] = 1+1+1 = 3

int B = box[3] + box[4] + box[5] = 1+1+2 = 4

int C = box[6] + box[7] + box[8] = 1+1+1 = 3

3, 4

```
if (A > B)
{   if (box[0] > box[1])
    {
    }
else if (A < B)
{   if (box[3] > box[4])
    {
    }
else
{
```

```
return box[0]
else if (box[1] > box[0])
    return box[1]
else
    return box[2]

return box[3]
else if (box[3] < box[4])
    return box[4]
else
    return box[5]
```

Analysis of Algorithms

1) Space Complexity → How much space require (memory)
2) Time Complexity → How much time required to execute the code

Time Complexity :-

⇓

Represent

A) Best Case ⇒ Omega Notation ⇒ Ω
B) Average Case ⇒ Theta Notation ⇒ Θ
C) Worst Case ⇒ Big O Notation ⇒ O

Time Complexity $\Rightarrow$ O(T.C.)

Algo 1 Vs Algo 2

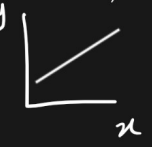
O(10sec) ✓ O(20sec)

?

How we can find out ?

⇓

Mathematical eq^n

1) Constant Eq^n $\Rightarrow$ Algo $\Rightarrow$ 2 sec, 5 sec $\Rightarrow$
2) Linear Eq^n  $y = mx + c$
3) Quadratic Eq^o
4) Qubic Eq^n
5) Bi quadratic Eq^n
6) Logrithmic Eq^n
7) Linear Logrithmic Eq^n
8) Expcnential Eq^n

Constant Eq^n

JAVA

```
p.s.v.m(  )
{
    int a = 10;   => 1sec
    int b = 20;   => 1sec
    int c = a+b;  => 2sec = (1+1)
    S.o.p(c);     => 1sec
}
```

Constant

5 sec ⟹ Time Complexity $O(1)$

Linear Eq^n :- Time Complexity

$y = mx + c$

$2n + 2$

```
1sec --> int value = 10      1sec  1sec   => 1sec
2n       for(int i=0; i<n; i++)              => 2n+1
         {
1sec -->   S.o.p("Hello");
         }
```

$1 + (2n+1)$

$= 2 + 2n$

increase most effective factor is n

$\approx n$

Time Complexity $O(n)$

1sec 1sec

```
for ( int i = 1 ; i < n ; i = i+2 )
{
    —          ≡ 1 sec
}
```

$\frac{2n}{2} + 2$ $\simeq O\left(\frac{n}{2}\right)$

$n = 10$

$i = 1$ —
$i = 3$ —
$i = 5$ —
$i = 7$ —
$i = 9$ —
$i \neq 11$

```
for ( int i = 1 ; i < n/2 ; i++ )
{

}
```

Time Complexity