Lec # 1 15-2-2021

Data structures & Algo

→ organizing the data efficiently.

Efficiency:

in terms of space and time.

So that

⇒ searching, processing and sorting of data gets better.

features:

→ Each data structure allows data to be stored in specific manner.

→ DS allows efficient data search and retrieval.

→ Certain DS are decided to work for specific problems.

→ ~~allows you to indexing as~~

Dictionary: —

Array:

int numbers [4] = {10, 20, 30, 40};

What is ADT;

→ two ways of looking at DS,

⇒ Mathematical / logical.

↳ DS follows some rules and regulations to store data, those rules can be termed as logical or model.

⇒ Implementation:-

using all those rules to implement DS practically with the help of any programing language.

Data types:

→ Defines a certain domain of values.

→ Defines operations allowed on those values.

int type

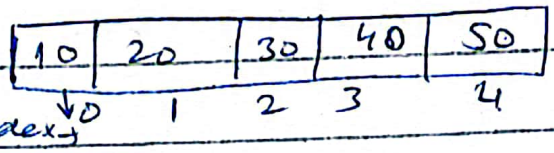
→ will take only integer values.

→ operations allowed sub, mul etc.

03/02

ADT:

ADTs are entities that are definations of data and operations but dont have implementation details.



Abstract view	Implementation view
→ Store set of elements. → Read elements by indexs → Modify elements by indexs	→ int arr[5] = {1,2,3,4,5}; cout<<arr[1]; arr[2] = 10;

Algorithm

finite number of steps to solve a particular problem.

S_1 : Read A

S_2 : Read B

S_3 : Sum = A+B

S_4 : Print (Sum).

→ instructions should be for finite time.

Analysis:

comparison of different algo's on a specific parameters.

i.e time, space.

no. of registers, throughput.

Two types of analysis.

1- priori

↳ Analysis before execution.

↳ find out the iteration of any function.

→ It gives approx value.

example:

S1 : Read A 1

S2 : Read B 1

S3 : Sum = A+B 1

S4 : Print(Sum) 1

Posteriori:

→ It gives exact value.

→ This analysis is H/W dependant.

```
main()
int a, b, sum;
scanf(a, b);
sum = a + b;
printf(sum);
```

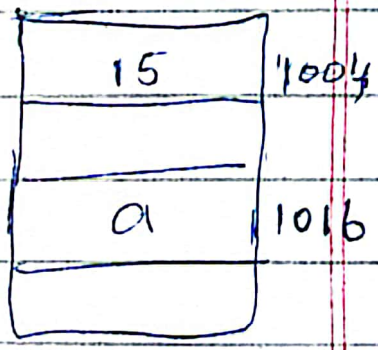
Lec #2 16/2/24

Primitive and refference data types -

Primitive types :

int, long, doublle, float, and char.

```
int i = 15;
char c = 'a';
```



data stores directly into memory.

Refference type :

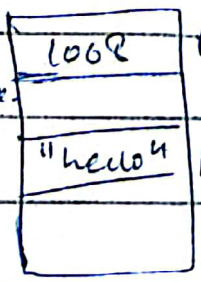
Types that hold complex values. (objects)

→ String is a refference type.

```
String s = "hello";
```

→ This will contains the address of value.

→ variable refferece the value.

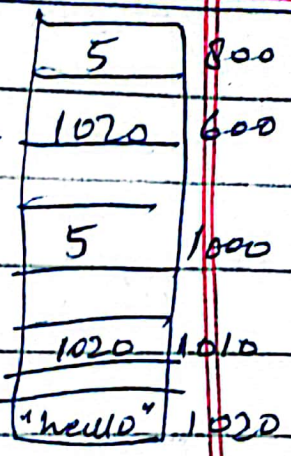


s → hello

it's act like just a pointer.

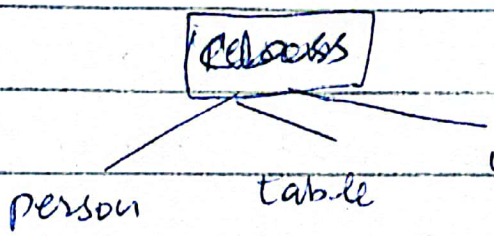
```
int i1 = 5;
int i2 = i1;

String s2 = "hello";
String s2 = s1;
```



Object and references:

Object is an instance of a class.

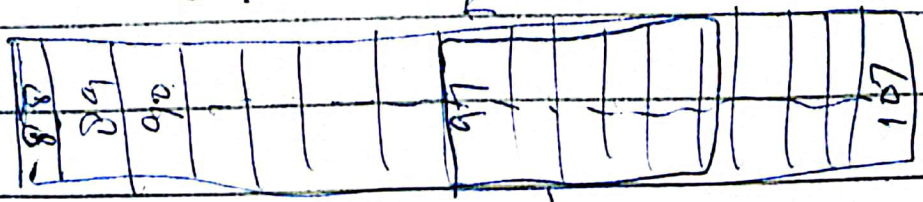


creating memory location

```
Person sally = new person();
```

Class name — Object name — constructor

sally = 97



→ object sally will actually refers to actual location of the class.

Common analogy:

class → House. → blue print.

object → actual house.

Memory Reference → gives the actual idea that where the actual house is, street, country etc.

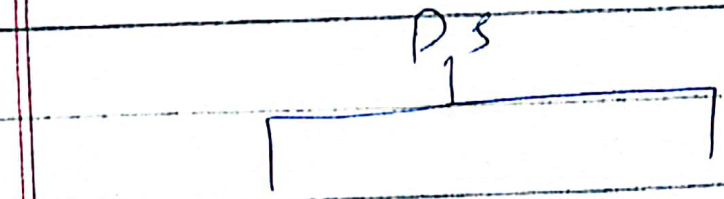
person Adeel = new person();

person Ali = new person();

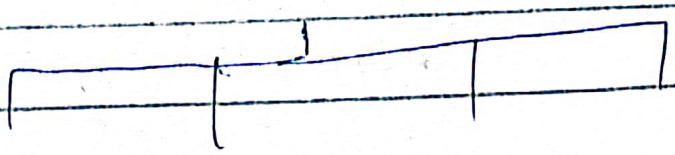
person imran = new person();

these actually are not persons but the ~~memory locations~~ references that refers to the memory locations.

Types of data structures:



→ when all the elements are arranged in a sequential order.

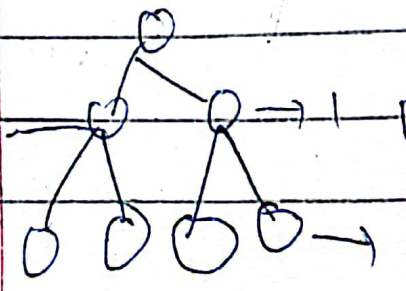


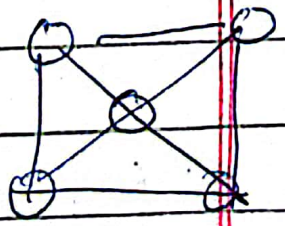
Array Ques stacks link list.

we also say that when a ds have only one predecessor and one successor.

Non-linear DS:

a ds when all the elements are not stored in a linear order.





graphs.

Interview questions:

Static and dynamic ds ?

Asymtotic Analysis :

A ds is the orgnization of the data in a way so that it can be used efficientey.
imp.

⇒ in order to use efficientey, we have to store it efficientey.

Efficiency of DS is allways meansured in tesms of time and space

⇒ ideal DS wonld be that takes less time and cousmes least memory location.

Q on what basis, we compare the time complexity of DS.

A On the basis of oparations performed on them.

Ex:

arr | 1 | 2 | 5 | 6 | 8 | 9 | 10 | 8 |

Assume this array can store upto 100 elements.

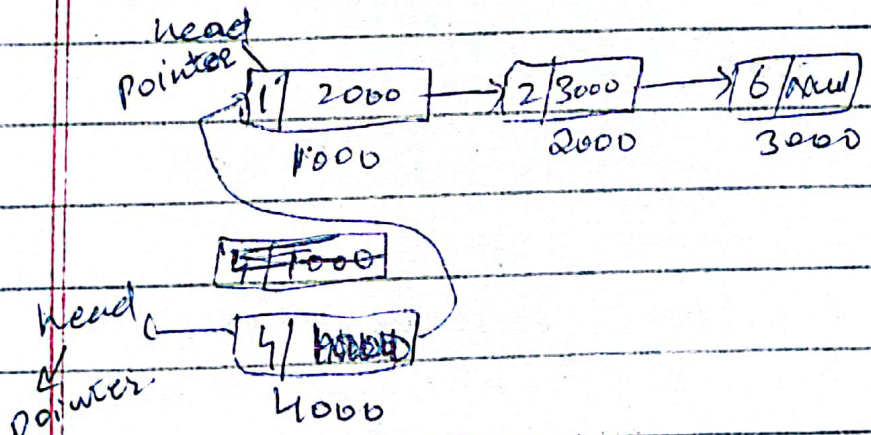
Add data at the begning.

4

⇒ we will shift all the elements to the right, obviously this will take much time.

take a linked list,

→ sequence of nodes connected with each other.



Conclusion:

Inserting an element at the begning of a linked list is way faster than adding in the array.

Logarithmatic time complexity.

$\log_2(8) = 3$

How many times to multiply 2 in order to get 8.

```
for (i=1; i<=n; i=i*2;)
{
    i = i*2;
}
```

iter 1	initially $i=1$	2^0
2	$i=2$	2^1
:	$i=4$	2^2
	$i=8$	2^3
	$i=16$	2^4
	:	:
its k	$i=n$	2^{k-1}

$n = 2^{k-1}$

How many iterations are there?

$\log_2 n = \log_2 2^{k-1}$

$k-1 = \log_2 n$

$k = \log_2 n + 1$

time complexity $\Rightarrow O(\log_2 n)$

```
for (i=1; i<=32; i=i*2;)
{
    printf("Hello");
}
```

$k = \log_2 32 + 1$

$= 5 + 1$

$k = 6$

number of iterations are 6 in total.

if-else construct:

```
scanf("%d", &n);
if (n == 0)
  {
    // statement;
  }
else
  {
    for (i = 1; i <= n; i++)
      // scanf;
  }
```

⇒ worst case running time =
test + "if" part or "else" part.
(whichever is larger).

for "if" part:

n == 0 will take a constant amount of time.

assumption, ~~constant~~ inside statement will also take constant time.

total time = 1 + 1 = Θ(1)

for "else" part:

n == 0, will take constant time.

time = 1 + n ⇒ Θ(n)

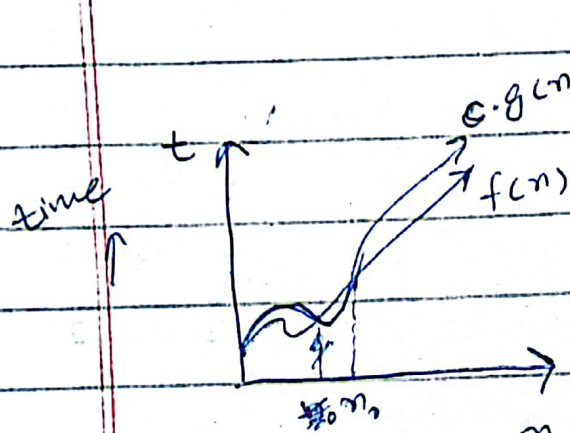
overall complexity = O(n)

Asymptotic Notations:

Big (Oh) → shows upper bound of a function.

↓ worst case

It's a mathematical way of representing the time complexity of the program or algorithm without being executed.



traversing array of size n

O(n)

$f(n) \leq c \cdot g(n)$

where $n \geq n_0$

$c > 0$, $n_0 \geq 1$

let's say

$f(n) = 2n + 3$

$\because f(n) = O(n)$

$2n + 3 \leq 5n$

$f(n) \leq c \cdot g(n)$

Scanned with CamScanner

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \cdots < 2^n < 3^n < 4^n$

lower-bound — average bound — upper bound.

$f(n) = 2n^2 + n \Rightarrow fn = O(\quad)$

dominant value

$2n^2 + n \leq c \cdot g(n)$

$2n^2 + n \leq 3n^2 \rightarrow$

$n \leq n^2$

$1 \leq n \;/\; n \geq 1$

$2n+3 \leq 2n+3n$

$fn = O(n)$

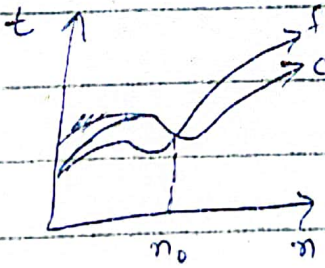
$2n+3 \leq 2n^2+3n^2 \rightarrow O(n^2)$

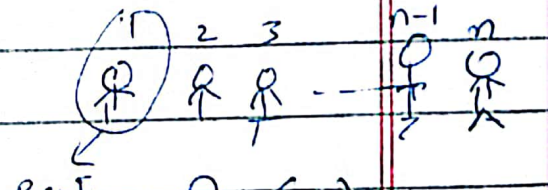
$O(\quad)$

$fn = O(g(n))$.

Big-omega (Ω) / best case

It is used to find best case of any algorithm.





$f(n) \geq c \cdot g(n)$

$n \geq n_0$

$c > 0,\ n_0 \geq 1$

$f(n) = \Omega(g(n))$.

$f(n) \geq c \cdot g(n)$

$2n + 3 \geq 1 \cdot n$

$f(n)$ $\quad c$ $\quad g(n)$

$\therefore f(n) = \Omega(n)$

$2n + 3 \geq 1 \cdot \log(n)$

$\Omega(\log n)$