Fatas Satter, KUET (FSC)

Classroom:

DSA 1

29.01.25

using C / C++

C# .net framework

"The less source, the more demanding the work"

Equit* framework (free)

"Competetive Programming is the most reliable"

↓

not language

↓

learn the concept and apply it

↓ syntax

Documentation?

↓

better than Book? → concept (general) → language (specific)

"5:5 rule" or, "5 is to 5 rule" ∧ concerned with stuff only when it would be relevant in 5 years.

"1 hour of focus for 5 minutes of entertainment/gaming/reward"

Book: Atomic Habits → If a work takes less time then just get over it

Privacy concern → Pathao scandal

GitHub usage

Book: The art of thinking clearly

↓

Imposter Syndrome

"নিজেকে নিয়ে যখন doubt হবে তখন বুঝবে আমি দক্ষ হয়ে যাচ্ছি"

↙ pep talk

DSA book: IUT writer's Book

↓

Sm will provide

Diploma or BSc?

classroom: uo bq sks

DSA-1 04.02.25

course outline:

- Introduction
- complexity
- Array (1D, 2D, 3D) → Linked list
- Searching
- sorting → Mid
- Stack
- queue
- .....

Book: Schaum series + IVT এর writer এর বই

(contains mistakes)

- Class Lecture materials are provided before the class in the Classroom
- Must follow the Book pattern ← when you study externally

classroom: x5zfb2n

DSA-1 [Lab] 01

04.02.25

Introduction to Data Structure:

- Regular Lab assessment

notice the mark updates

Makeup assignment (hardcopy)

- Mark Distribution:

Attendance (20) → { Attendance (5), Continuous Lab Performance (15) }

Lab Performance (30) → { Labtest-1 (15), Labtest-2 (15) }

Quiz (30) → { Quiz-1 (10), Quiz-2 (10) } with Labtests, Assignment (10)

Viva (20) → Presentation (video project)

↓

explaining Algorithm

or, group presentation

{ Report (5), Presentation (15) }

C Language concepts: (Important for DSA)

- pointer
- recursion
- function

Syllabus until mid: 07. ~~Mid Queue~~ ← course outline

Project

Data structure (2 types) → linear, non-linear

Book: no specific books

Organized sequential contiguous memory → Linear Data Structure

non-contiguous memory → non Linear Data structure

Linked between two separated memory locations

Category:

Data structure
- Linear
  - Dynamic → List, Stack, Queue
  - Static → Array
- Non-Linear → Graph, Tree

Q. Why learn Data structures?

↳ Q. Why learn Programming Language?

⇒ we learn programming to communicate with computers easily (rather than with machine code)

⇒ Application of Data structures, learning how Data works in computer operating. function is a basic structure.

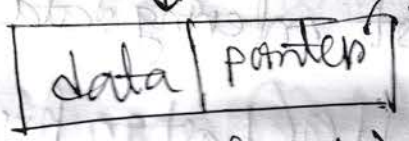
Abstraction? ignoring the internal complexity of Built-in functions.

we learn the internal workings in datastructure (some)

Vector? Array like data structure but it is dynamic unlike Array.

Linked list : first topic of non-linear data structure

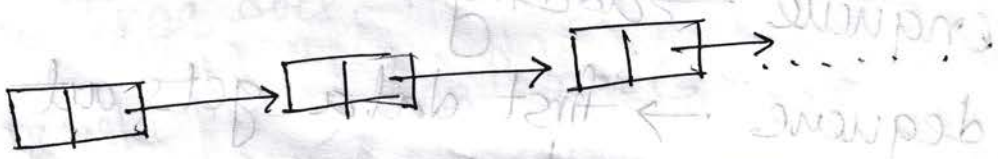
Node



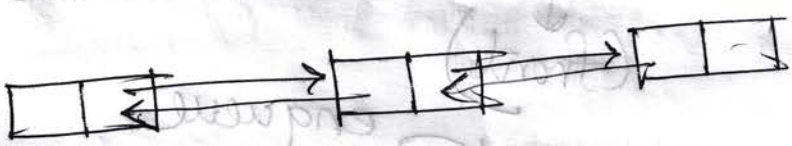
(Parts of node)

types : (1) singly connected Linked list
(2) Doubly Linked list
(3) Circular Linked list

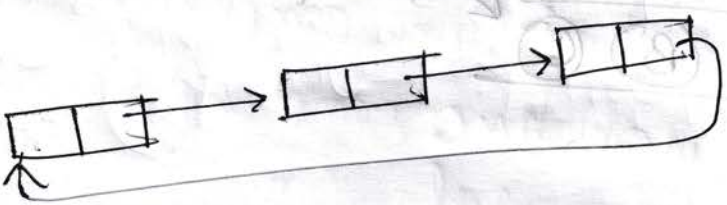
(1)



(2)

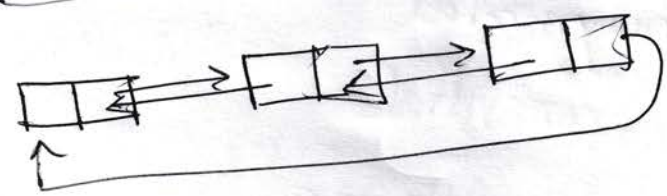


(3)

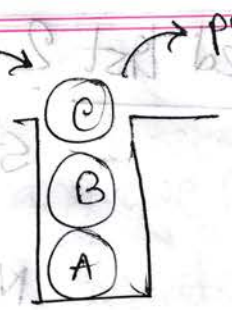


(Circular + doubly)

and,



Stack: (2m)



push → putting data

pop → pulling out data

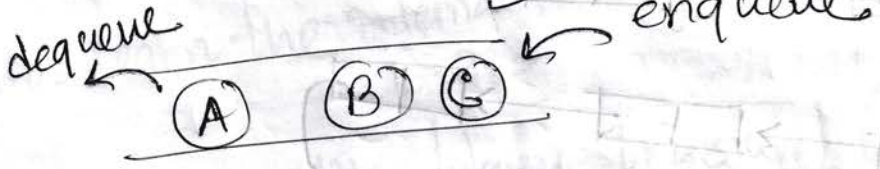
↓

Top-down approach (last entering data is removing first) → LIFO (Last in first out) principle

Queue: follows FIFO principle (first in first out)

enqueue → adding element

dequeue → first data gets out (front)



→ Graph theory → Graph traversal

#Graphs: ① undirected
② Directed
③ weighted (added costs of each node)
(and unweighed)

Representation: ① Adjacency list
② Adjacency matrix

#Tree: graph does not maintain a hierarchy.
Tree does. Root node, Parent node,
child node, leaf node

Time Complexity Analysis: (Runtime)

algorithm → set of instructions

↓

finite program (not infinite loop)

Definition: the time taken to execute the set of codes (instructions), and not the total time

mathematical concepts

↑ **Asymprotic notation / relation:** (DM book! and most efficient)

used to represent time complexity:

- → Big O (upper bound / worst case)
- → Big Omega (Ω) (lower bound / best case)
- → Big theta (Θ) (tight bound / average case)

Big O notations are most common

- $O(1)$ = constant time complexity
- $O(n)$ = linear time complexity
- $O(n^3)$, $O(n^2)$ = polynomial "
- $O(2^n)$ exponential "
- $O(n \log n)$, $O(\log n)$ = logarithmic "

$O(n \times m)$ notation indicates nested loops of n, m (outer, inner)

if $(n = m)$ then $O(n \times n) = O(n^2)$ → Polynomial time complexity

code:

```
for(int i = 0; i = n; i++)
{   for (int j = 0; j = m; j++)
    {       }
}
```

- Binary search works with pre-sorted (ascending) array.

Linear search → $O(n)$

Binary search → $O(\log n)$

} the most "Not efficient" as said in Discrete mathematics

Traversal algorithm: most common algorithms in Data Structure.

Graph traverse:

starting level 0 / Branch

Depth / lowest level

- BFS (Breadth first search)
- DFS (Depth first search)

node / vertex (V)

edges (E)

- notation: $O(VE)$ and $O(V \times E)$ → same as $O(n \times m)$

"Dense node"?

"class" and "structure"

- finding shortest path (lowest weight cost)

☑ Traverse? boolean type variable "flag" to represent "traversed"

flag = false ← global scope

Online judge

#Space complexity: (?)

- Our concern is to eliminate the cause of Big O
- computer can explore/traverse $10^7 \approx 10^8$ step per second. $1\ s = 10^7$ steps
  ↓ we consider
- Time limit depends on loops. (loop ∝ Time complexity)
- Runtime and time limit? ~~lower~~ upper limit ~~enter~~ on Runtime (?)
  - time taken to run the code
- value assigning to a variable takes constant time complexity O(1)
- compilation time is the time taken to convert C code into machine code before Running (?)

for(i=1 ; i<=n ; i++) ⟶ checks (n+1) times

{ //statements ⟶ checks (n) times

}

∴ time complexity = $O(n+1+n)$

$= O(2n+1)$

$\simeq O(n)$

↓

- we ignore variable assignment operation times. (ignores constants)
- machines only concern with the values of n.

$\boxed{1s = 10^7}$

$O(n) \rightarrow 10^7$

$O(n^3)^2 = 10^6$

$O(n+1+m+1)$

$O(n+m+2)$

$= O(n+m)$

$= O(n)$; if $(n > m)$

∴ m cycles are counted within n cycles.

(does not separately check for m) / operate

~~while~~ after each n, m is checked

```
for( ; i<=n; )
  { }
for( ; j<=m; )
  { }
```

$O(\sqrt{n})$

```
for(i=1; i<=sqrt(n); i++)
  { ... }
```

$O(\sqrt{n})$

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

- Tasks with ~~time~~ limit (CP problems)
- Online judge & → Problem solving sites like Codeforces.

$O(\log n)$

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

$O(\log n)$

```
for(i=256; i>=1; i=i/2)
{
}
```

$\frac{N}{2^k} \to 1$ → গুণিতক আকারে বা ভাগ/কমা → logarithmic

meaning logarithmic time complexity

$\Rightarrow N = 2^k$

$\Rightarrow \log_2 2^k = \log_2 N$

$\Rightarrow 1 \cdot k = \log_2 N$

- logarithmic operations → $1s = 10^{18}$
- $\sqrt{N}$ → $1s = 10^{14}$

conclusion: $O(N) < O(\sqrt{N}) < O(\log N)$

Basic CPP syntax:

C header → #include <stdio.h>

CPP header → #include <iostream>
using namespace std;

scanf() → cin >>

printf() → cout <<

↓

shifting operator (?)

hw → cover page

short explanation on, (+ code)

→ hard copy

O(1)

O(N)

O(√N)

O(N²)

O(log N)

O(n log N)

DSA-1 11.02.25

C1.3 (Schaum)

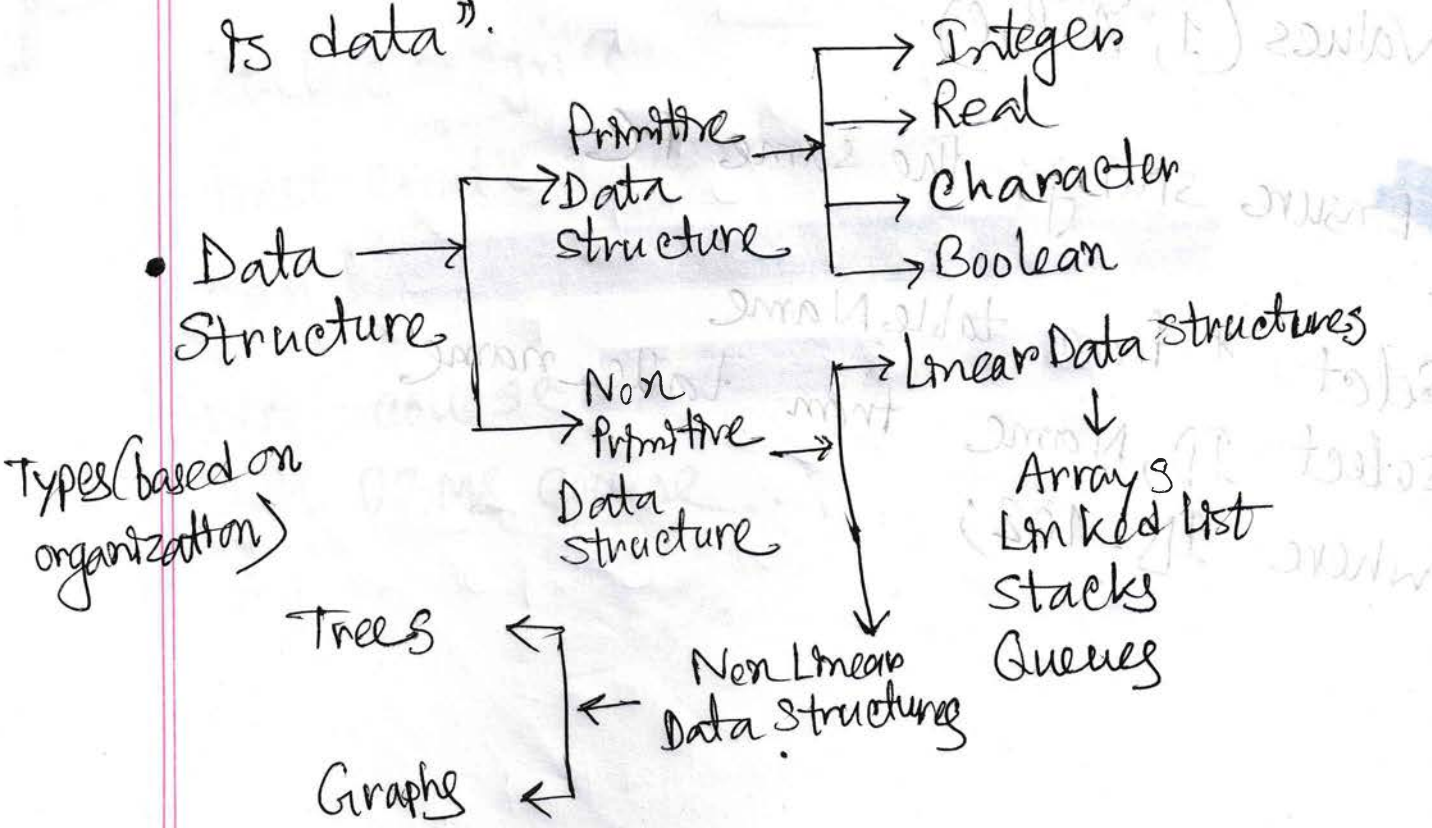
"The logical or mathematical model of a particular organization of data is called a data structure"

"Data are values or set of values"

elementary data and group data

information = meaningful data

"not all data is information but all information is data".



- Data needs to be easily organizable

Linked lists? non-linear/sequential or broken apart - set of data that each part knows the memory location of immediate previous/after data.

Graph?

- Two considerations of Data structures?
  - workable facilities
  - simple

Algorithm? List of steps to solve a particular problem.

C 1.6 : Algorithms, complexity, time-space trade off?

- ADT?

Assignment

DSA-1 Lab 02 11.02.25

Array Traversal:

Q1 **Pseudocode:**

steps:
- Take array size as input
- Run a loop from 0 to array size
  (Take array elements as input)
- Loop end
- Print a newline
- Start a loop from 0 to array size
  - (print array element)
  (print a space)
- end loop

Code:

```
#include <iostream>
using namespace std;

int n; cin >> n;
int a[n];
for (int i=0; i<n; i++)
```

for loop block scope

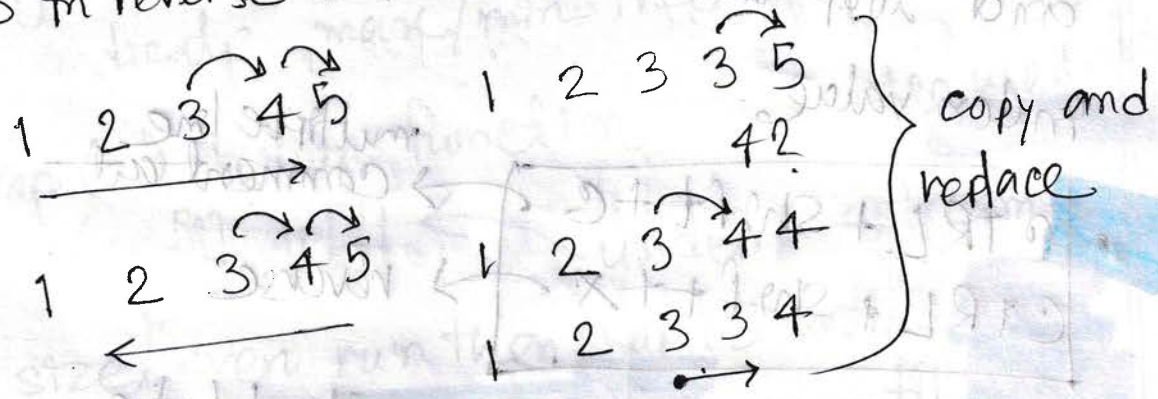
```
{ cin >> a[i]; } cout << endl; for (int i = 0; i<n; i++){
  cout << a[i] << " "; } return 0; }
```

- Scope? access level?

- local
- global

Right shift Q2 input "position" and "value" of an array. shift elements to the right to prepare the position to replace with the input "value".

- loop in reverse so previous values exist



(alt) using shifting operator? eliminates the possibility of going out of bounds in array. &

→ * cpp doesn't show error

```
(...)
int pos, val; cin >> pos >> val;
for (int i = n-1; i >= pos; i--)
{ a[i+1] = a[i]; } a[pos] = val;
for (int i = 0; i < n+1; i++){ cout << a[i] << " "; }
```

Left Shift

Q3 Delete an element?

- shift to the deleted elements position and loop ~~until the~~ without the last index value.

- CTRL + shift + C → multiple line comment out
- CTRL + shift + X → reverse

- Static data structure can't delete an element but can only overwrite.

(...)

cin >> pos; for (int i = pos+1; i < n; i++)

{ b[i-1] = b[i]; }

for (int i = 0; i < n-1; i++){

cout << b[i] << " "; }

Q4 Reversing an array? $O(\frac{n}{2}) \rightarrow O(n)$

(methods)

- basically reversing the traversal (does not affect array structure)
- using "Two pointers technique" ↓ also used in Binary search
- we indicate max/min index and replac
- swap( ) builtin function
  swap(ar[i], ar[j]) ← under <iostream>
- if size is even run then swap( ) until $i < j$
  if odd then stop at $i = j$ or until $i != j$
- if n used instead of j (using the size n) then it might change (∵ n--)

```
int n; cin>>n; int a[n];
for(int i=0; i<n; i++){ cin>>a[i]; }
        // Two Pointers technique
int i=0, j=n-1;
while(i<j){   //included i=j case
swap(a[i],a[j]);
i++, j--; } for(int i=0; i<n; i++){
    cout<<a[i]<<" "; }
```

make sure to update the position

- to run object file

g++ -o reverse_arr reverse_array.cpp

reverse_array.o

runs

Passing parameters in main function

```
int main(n...){
```

Q. How to run .c/cpp from terminal with prompt from terminal?

(Main function with argument)

$\sqrt{O(n)}$

$O(\log n)$

exponential way তে কোনো কিছু কমলে/ বাড়লে

```
for (i=10; i<n; i=i*2)
{
}
```

	i	
1	$1 = 2^0$	1
2	$1\times2 = 2^1$	1
3	$2\times2 = 2^2$	1
⋮	⋮	⋮
k	2^{k-1}	

```
for (i=n; i>=1; i=i/2)
{
}
```

$i = n$

$2^{k-1} = n$

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

$\Rightarrow (k-1)\,1 = \log_2 n$

$\Rightarrow k = \log_2 n + 1 \approx O(\log_2 n)$

nested cases → nested $O(\log n)$

$O(n \log n)$

```
for (i=1; i<n; i++)        <- O(n)
{
    for (i=1; i<n; i=i*2)  <- O(log n)
    {
    }
}
```

$n \times \log n$

$O(\sqrt{n})$

assume $i = \sqrt{n}$

$i * i = n$

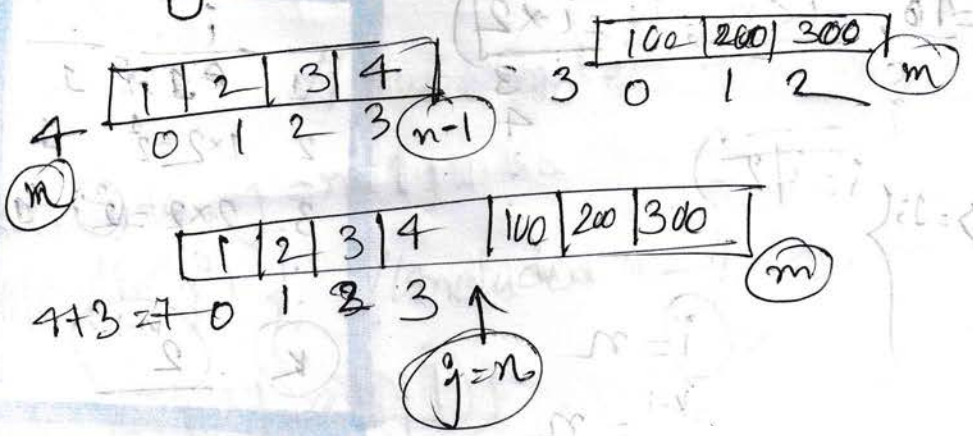
$k * k = n$

$\Rightarrow k^2 = n$

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

Q5. Merge array

- adding two arrays in a 3rd array




```
int n,m; cin>>n>>m; int a[n], b[m];
for(int i=0;i<n;i++){cin>>a[i];}
cout<<"\n break \n";
for(int i=0; i<m; i++){cin>>b[i];}
int c[n+m]; for(int i=0;i<n;i++){
    c[i]=a[i];} int j=n;
for(int i=0;i<m; i++){c[j]=b[i]; j++;}
for(int i=0;i<n+m;i++){cout<<c[i]<<" ";}
  return 0;}
```

next →

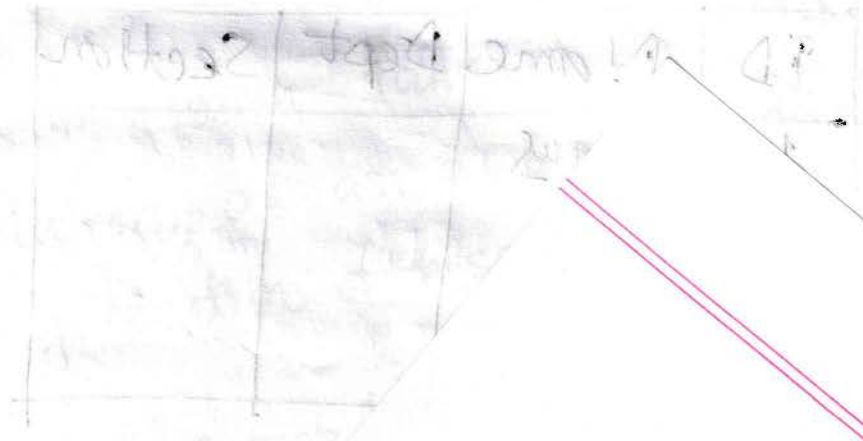
string as array

prefix array

multidimensional array

vector (dynamic)

Practice Problem → Online judge



is

searching takes

DSA-1 18.02.25

C2.3 Algorithmic Notations:

~~usn~~ inspired from Pascal language

formal notation:

- (assign operator) := ← =
- (ending block operators) Exit ← }
- (Procedure/module) ← user defined function code

↑ main function code is called "Algorithm"

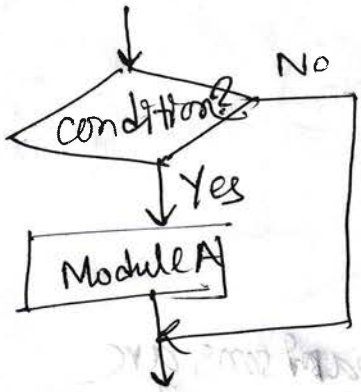
- (Input) ~~Scanf~~ Read ← Scanf
- (Output) Write ← Printf

control structure (sequential flow, conditional flow, / Selection / logic
/ Logic
Top-down approach Iteration / repetative logic/flow)

if block/scope

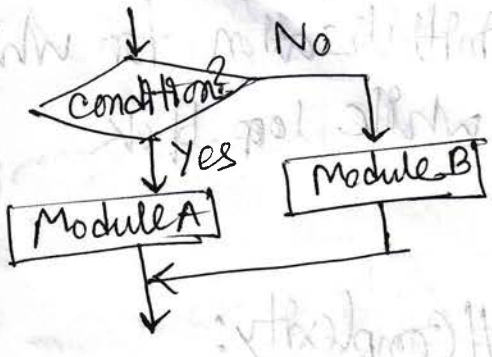
python does not used { } for defining scope but indentation

(single alternative)




```
if( ){ }
```

(Double alternative)




```
if( ){ }
else{ }
```

(Multiple alternative)

```
if( ) { }
else if( ){ }
else{ }
```

} creates instances for each conditional statement but chooses only one

↑ metaphor

for loop → initialization
↓
condition check
↓
statement execute
↓
increament/decreament

- Repeat for structure
- Repeat while structure

initialization for while loop happens before the while loop block

#Complexity:

big-O

binary search $\log_2 n$

linear search n

} comparisons are observed clearly for big steps / no. of data

np-hard
np-complex
} $O(n!)$

→ Discrete mathematics

(ask sir to explain later) why these cannot be solved

Bounty money on solving these problems

→ Movie: Back to the future

1900s sci-fi stories

- jimmy falon show → lenovo

☐ Assignment → complexity graph plot Book (Page 49)

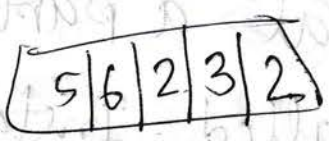
$\log n, n, n\log n, n^2, n^3, 2^n$

DSA-1 Lab

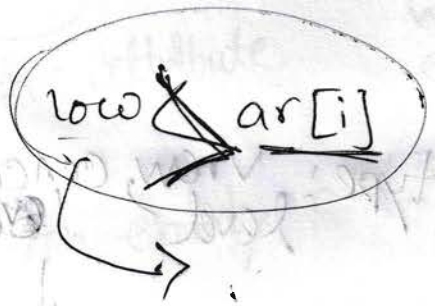
#4 and #9

2D array, tag, element

string type (array problems) → Problem sheet from Task-2



Tips: if



(flag not needed as min < a[0] skips that)

start array index from 1 since min = a[0] already

faculty changed to? Oditthi ~~ma~~ Ghosh. ma'am

DLD

19.02.25

multiplexer

19.02.25

DSA-1

$f(x) = y$

$f(2) = y$

$x^2 + 2 = y$

$f(x) = 1$

for every input the output is 1 hence O(1)

each step takes the constant amount of time

n \ g(n)	$\log_2 n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	10^2	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^9	10^9	10^{300}

fig. 2.6 – Rate of Growth of Standard functions

"wait in thought"

Complexity calculation:

$$|f(n)| \leq M|g(n)|$$

high bound

positive value.

$$8n^3 - 576n^2 + 832n - 248 \leq 9n^3$$

$f(n)$ $g(n)$

- If I can represent the equation in this format then the value on the right is the result complexity.

1 unit greater than the coefficient on the left

- worst case means "high bound" or highest limit of the execution time.

"Primary key"

DSA-1 25.02.25

#C 2.6 - Best case scenario / lower bound :

Omega notation

$$|f(n)| \geq M|g(n)|$$

Q. $f(n) = 18n+9$

$18n+9 \geq 18n$

$18n$ → M, $g(n)$

complexity: $\Omega\, g(n)$

Q. $90n^2+18n+6 \geq 90n^2$

complexity: $\Omega\, g(n)$

Theta notation

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$$

Q. $f(n) = 18n+9$

$18n \leq 18n+9 \leq 19n$

$18n$ → M_1, $g(n_1)$ $= M_1 g(n)$

$19n$ → M_2, $g(n_2)$ $= M_2 g(n)$

- **Time complexity** is basically the time required for an algorithm to complete executing all steps. Big(O) means highest required time

output for the corresponding input value:

Big(Omega) means lowest required time → Best case

Big(theta) means average or, in-between the highest and lowest required time

→ If we address this case then we can address the worst case scenario - meaning our algorithm can be no "lesser" efficient and and only be towards Omega and theta (can only improve)

PUT writer's book → C-2

 # Array :

- compiler remembers only the starting index.
- each increasing index increases size by 4 bytes.
- to find a memory location of a specific index,

formula

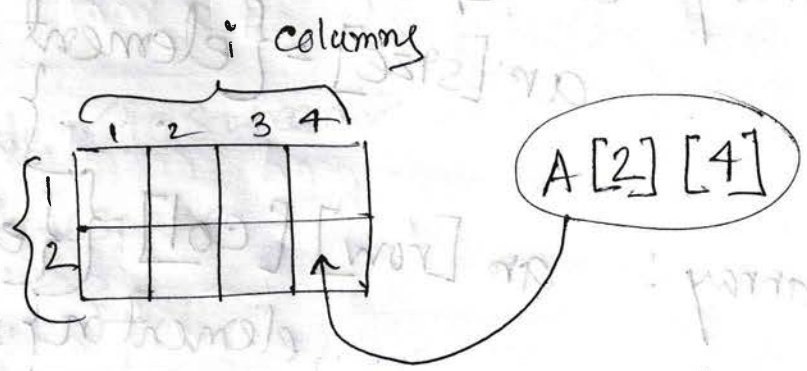
let, A[097] ; 1000 + (097 - 1) × 4.

= 4984 → mem address

- in 1D array, each location takes 4 byte space

therefore taking 1000+4 = 1004 bytes space after the first index.

2D array:



row col
~~row~~ ~~col~~
↓ ↑
(m x n)

$$A[0] + \{ n \times (i-1) + (j-1) \} \times 4$$

→ row-major order

→ why (-1)?

↑ starting value

$$Base(A) + \{ (j-1) \times m + (i-1) \} \times w$$

→ column-major order

→ data type size (4 for int)

Task:

- [ ] watch yt video (previous class, time complexity)

DSA-1 Lab 03

26.02.25

#Multidimensional Array: (2D array)

1D array declare/initialization:

ar[size] = {element 1, element 2, ...}

2D array: ar[row][col] = {{element 1, element 1, ..., element n}, ..., {element 1, element 2, ..., element n}}

column →

	0	1	2
0	2	3	4
1	5	6	7
2	8	9	11

row ↓

3×3 →

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

for 2D array we need a nested loop.

```
for(int i=0; i<row; i++)
{ for(int j=0; j<col; j++)
  { cin >> ar[i][j];
  }}
```

3D array : ar[ ][ ][ ]

code:

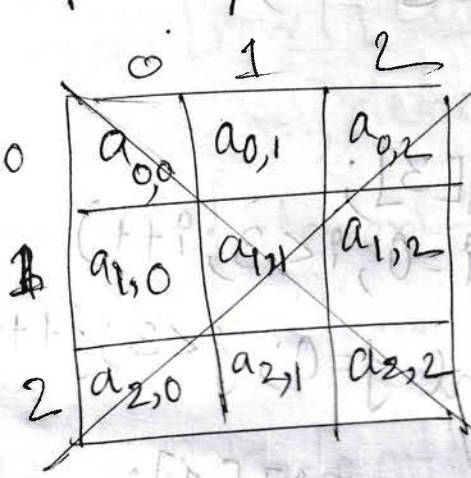
```
#include <iostream>
using namespace std;
int main(){
    int a[3][3];
    for(int i=0;i<3;i++)
    { for(int j=0;j<3;j++)
      { cin >> a[i][j];
      }}
    for(int i=0;i<3;i++)
    { for(int j=0;j<3;j++)
      { cout << a[i][j] << "  ";
      } cout << endl;
    } return 0;
}
```

Matrix types?

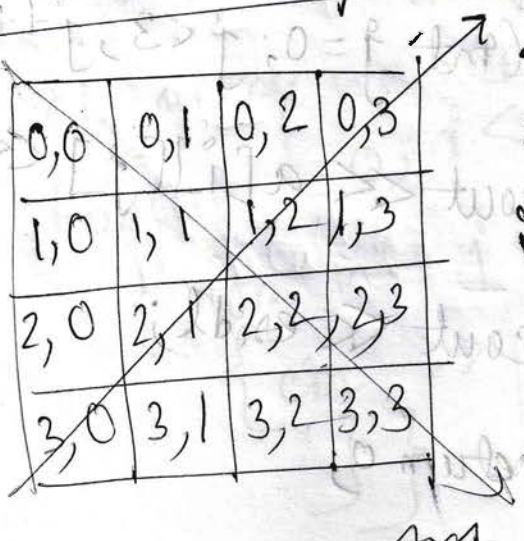
⇒ row matrix, column matrix, square mat, diagonal of a mat

↓

primary and secondary



to understand the pattern of secondary diagonal:



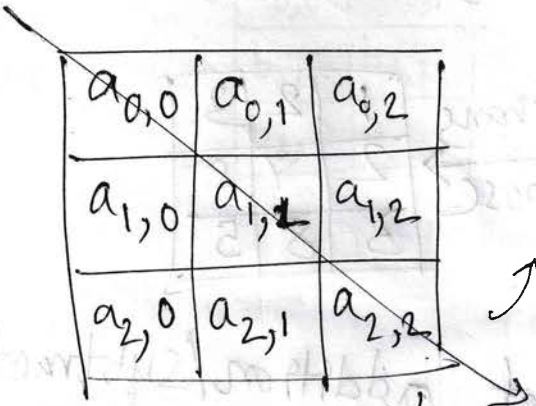
row/col = 4
= row-1 = col-1

2+1=3

if (i+j == row-1)

1+2 = 3 = row-1
3+0 = 3 = row-1
i+j = row-1
or, i+j+1 = row

Triangular Matrix?



not diagonal $i \neq j$ and column > row $i < j$

upper triangle

lower triangle

if any one is = 0 then

row > column and $i \neq j$, $i > j$

Diagonal Matrix? / Identity matrix

$a[i=j]$ and $a[i \neq j] = 0$

1	0	0
0	1	0
0	0	1

Transpose matrix?

1	2	10
2	7	8
3	8	11

→

1	2	3
2	7	8
10	8	11

transp[i][j] = a[i][j]

```
int arr[i][j];
int transp[i][j];
```

Task

Symmetric mat?

If mat = (mat)^T

1	2	3
2	7	8
3	8	5

Transpose →

1	2	3
2	7	8
3	8	5

- condition for mat addition/subtraction,

$m = n$

↑ row, ↑ column

mat add/subtract?

	0	1	2
0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$

+

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$

=

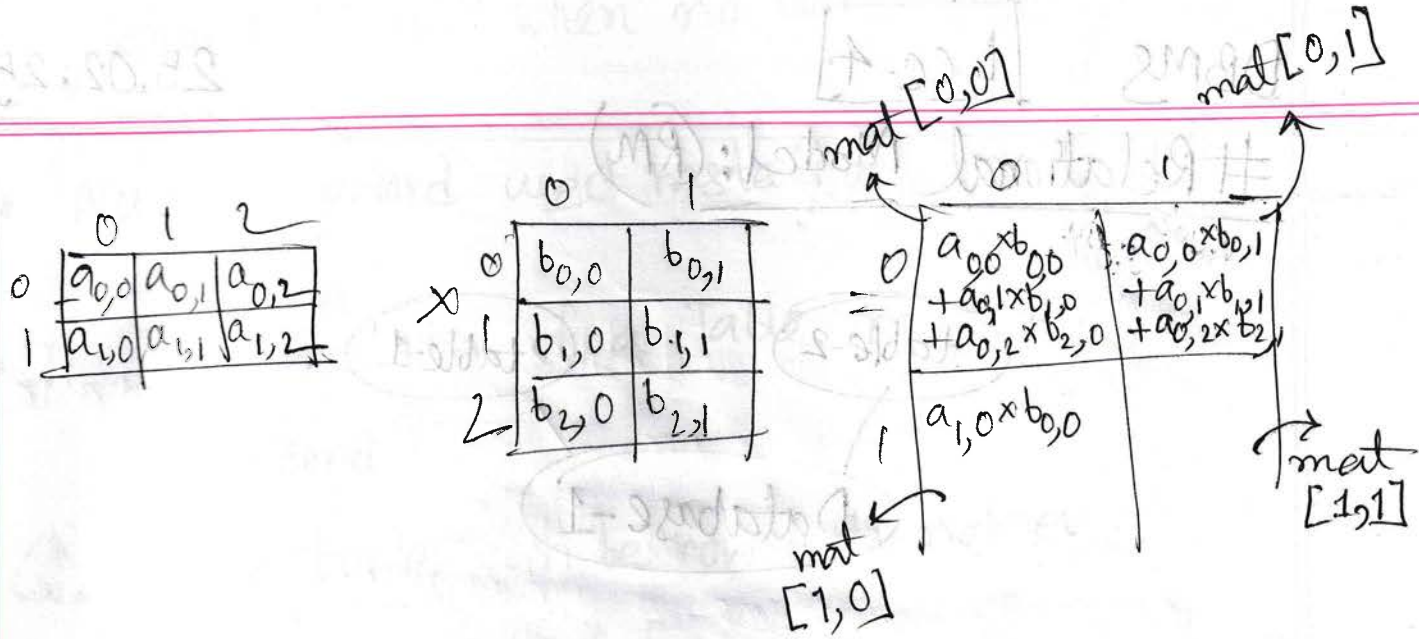
$a_{0,0}+b_{0,0}$	$a_{0,1}+b_{0,1}$	..
...	...	...
...	...	...

mat multiplication?

condition : $(m \times n) \times (n \times m)$

↑ = ↑

result mat : $(m \times m)$



3 loops;

loop-1 → row traversal

loop-2 → column traversal

loop-3 → changing index patterns when multiplying (for accessing indexes)

3rd mat → for multiplication

$n(mat1) = m(mat2)$

```
mat[i][j] += a[i][k] * b[k][j]
```

Ex: $\begin{matrix}1&2&3\\4&5&6\\7&8&9\end{matrix} \times \begin{matrix}9&8&7\\6&5&4\\3&2&1\end{matrix} = \begin{matrix}(9+12+3)&(8+10+2)&(\ )\\(\ )&(\ )&(\ )\\(\ )&(\ )&(\ )\end{matrix} = \begin{matrix}30&24\\\ &\ \end{matrix}$

(0+2+6)

sum = 0

```
for(int i=0; i<row; i++)
{
  for(int j=0; j<col; j++)
  {
    for(int k=0; k<row; k++)
    {
      m3[i][j] += m1[i][k] * m2[k][j];
    }
  }
}
```

→ or, col

DSA-1 (c4) 26.02.25

Linked List:

(c2)

Algorithms for 2D array:

How to declare diagonal matrix ?

for Primary diagonal

```
sum = 0;
for(i=1;i<6;i++){
  for(j=1;j<6;j++){
    if(i==j) sum += A[i][j]
  }}
```

for secondary diagonal

1+5
2+4
3+3
4+2
5+1
} = 6 = m+1 or; n+1

```
for(i=1; i<=m; i++){
  for(j=1;j<=n;j++){
    if(i+j==m+1) sum += A[i][j]
  }}
```

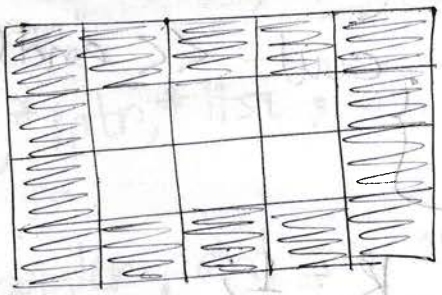
// sum for middle element overlaps
// we either need to reset sum (=0)
or show both in the same line

"contributions of others' behind a career"

for both diagonal:

```
for(i=1; i<=n; i++){
    for(j=1; j<=n; j++){
        if((i==j) || (i+j==n+1))
        { sum += A[i][j]; } } }
```

Boundary element: highest and lowest row and column index:



```
for(i=1; i<=n; i++){
    for(j=1; j<=n; j++){
        if(i==1 || i==5 || j==1 || j==6)
        { . } } }
```

look for more problems at [w3resources]

print output:

```
1
2  3
4  5  6
7  8  9  0
1  2  3  4  5
```

~~code~~ int k=1;

code : for(i=1; i<6; i++)

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

{ cout << k << " "; k++; }

cout << endl; if(k==10){k=0;}

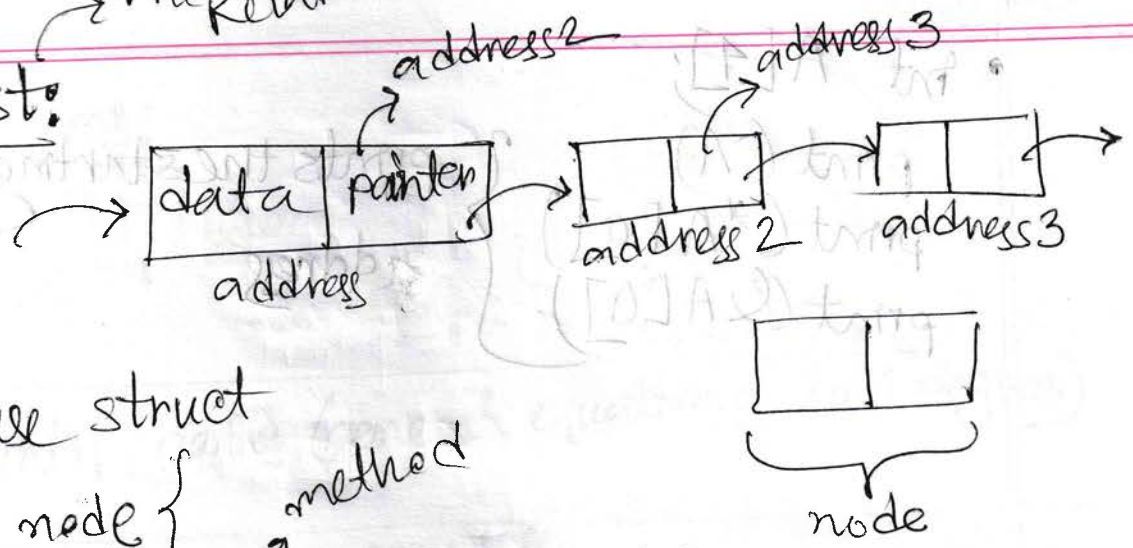
}

alt code: k=1;

```
for(j=1; j<=5; j++){
    for(i=1; i<=j; i++){
        if(k<=9){ print(k); k++; }
        else{ k=0;
              print(k);
              k++; } }
    print("\n"); }
```

Linked List:

→ kinda like "Relative Links" in WAD



- we will use struct

```
struct node {
    int data;
    node* next;
};
int i,n,item;
node *nptr, *ptr, *lst;
```

method

object-like

node type variables

- can one data type pointer point to another datatype(?)
- variable is a pointer

```
scanf("%d", &A);
```

- node data type keeps double size with the (int data + its mem address)
- compiler only remembers the starting address.

- int A[4];

  print(A)
  print(*A[0])
  print(&A[0])

  } prints the starting address

DSA-1 Lab 04

04.03.25

#Searching algorithm/technique:

(1) Linear search ⟶ O(n) → can be applied on unsorted array

#(2) Binary search ⟶ Divide and conquer

worst case: $O(\log_2 n)$

best case: O(1)

Prerequisite: Sorted array

(1) check if the target value is equal to the mid value or not

(2) for ascending order, the target value will be on the right of mid value if greater and vice versa.

(3) floor( ) values taken as mid point.

floor(7.5) = 7

ceil(7.5) = 8

(4) level by level division

(to avoid overflow)

alt:

$$mid = low + \left(\frac{high - low}{2}\right)$$

alt: can use floor() built in function

alt: can just mid = $\frac{high+low}{2}$

int variable take only int part of float

can cause overflow for large data

#Linear search: Avg/Worst O(n)

Best O(1)

(1) 1D array input

(2) flag (bool type) to check found/not

#Bool type variable

↳ True / False] only 2 values
1 0

LAB Test → Next week ~~(04.03.25)~~ (08.04.25)
(15 mark)

+

Quiz → (10 mark)

~~mapping~~

↓

Time complexity
Array ~~1D~~
Array 2D
Searching technique
Prefix array
Sorting Techniques
+ next class topic

Iterative method

#Binary Search:

$n = 13$; $arr[n] = 5, 7, \ldots, 65, 72$ (0 … 12)

(L) left = 0

(R) right = n − 1

$mid = \left\lfloor \frac{L+R}{2} \right\rfloor$

if

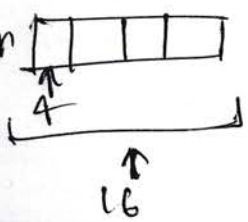
① Target value = mid
then
return mid

② if
Target value > mid
left = mid + 1
else if
Target value < mid
Right = mid - 1

③ if
Left == Right
then
return mid

#include <algorithm>
sort (array_name, array_name + array_size)

- sorting function:
- sizeof()
- to pass array as arguments: int n = sizeof(arr) / sizeof(arr[0]);
  sizeof(arr) → 16, sizeof(arr[0]) → 4

when arr 

n = 16/4

cin >> data; // to find

int n = sizeof(arr)/sizeof(arr[0]);

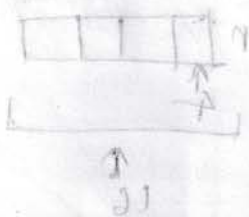
binary search(arr, data, n);

algorithm:

```
binarysearch(int arr[], int data, int n)
{
    int l = 0;
    int r = n-1;
    while(l <= r){
        int mid = (l + r)/2;
        if(data == arr[mid]) {return mid;}
        else if(data > arr[mid]){ l = mid + 1;}
        else { r = mid - 1;}
    }
    return -1;
}
```

Hw

- explore → recursive method (using recursion)
- Range sum problems,



#cumulative sum:

purpose: find ~~from~~ sum from the given array in range (,)

#optimization

- Range sum?

q ⟶ 10^7 (query)

↳ max operations

↓

how many times I want I range sum asks for L and R

```
while(q--)          O(n)
{
  cin >> l >> r
  for(l → r)        O(n)
  {                 }}
```

$O(n^2)$

How do we reduce the complexity?

- Prefix sum? (solving Range sum problems with time complexity > 1)

Cumulative sum / समयोजित मात्रा

doing sum before (instead of a nested loop)

Data ar[]	C. sum pre[]
2	2 → pre[0] = ar[0]
5	7
7	14
3	17

pre[i] = pre[i-1] + ar[i] (loop)

	0	1	2	3	4	5	6
int arr →	3	7	5	2	7	3	4
int pre[] →	3	10	15	17	24	27	31

pre[0] = arr[0]

pre[i] = pre[i-1] + arr[i]

L = 2

R = 5

to avoid garbage value

```
if(L==0)
  {cout << pre[R] }
else { pre[R] - pre[L-1];
     }
```

```
loop 1 (i=0; i<n)
  cin >> array
loop 2 (i=1; i<n)
  cin >> cumulative
         sum array
loop 3 (cin >> query
         /test case
  and cin >> range
         from pre[])
```

overall no nested loop (O(n²) cause) and a single loop making O(n)

DSA-1 05.03.25

Page-27 (Program to create a linked list)

- node type list
- "list" pointer and set as =NULL
- nptr = new(node) //new node creation

Keywords?

NULL → blank value → struct instance =

new → new instance creation

(X) next

- -> pointer operator? works like "this" keyword → references to instance variable
- list ← initial value
- (i++ and (++i) used for loop expression → does not matter
- malloc → dynamic memory allocation
- calloc ?

- using struct datatype we create

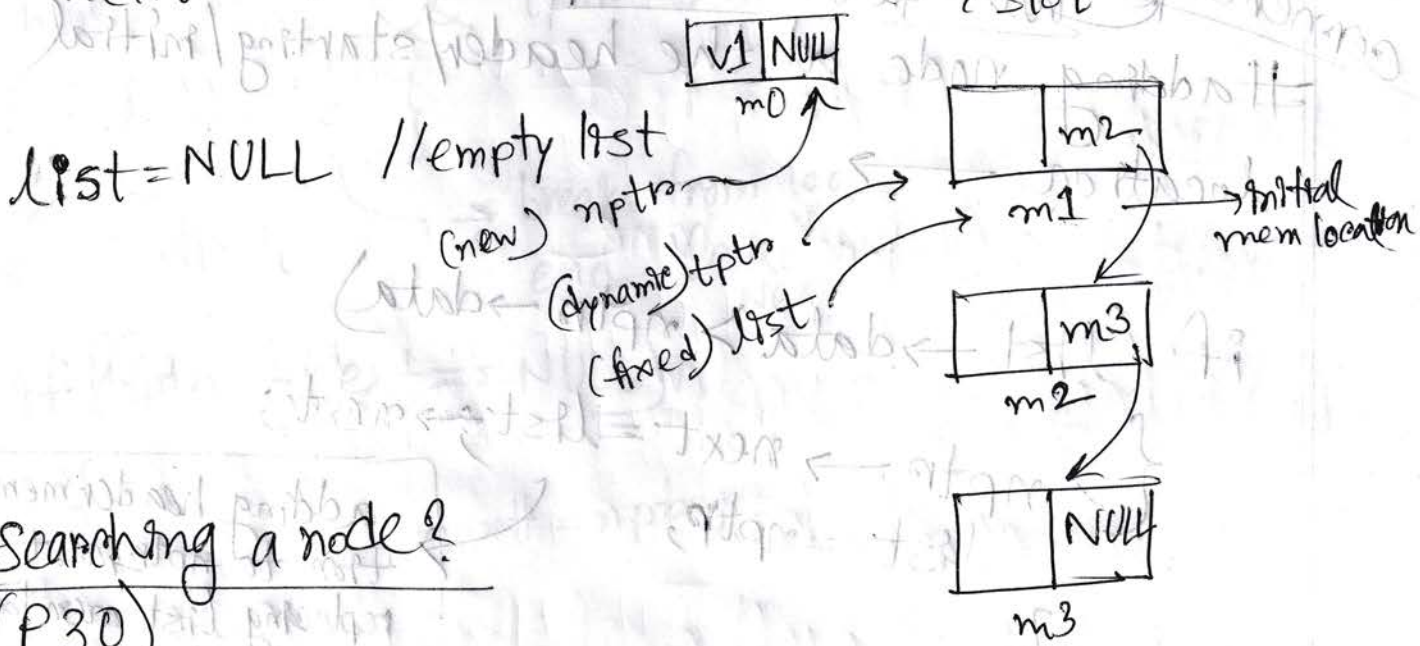
```
struct node
{ int data;
  node* next;
}
node *nptr, *tptr, *list;
```

→ creates 2 slot like structure

→ structure instance

Algorithm 4.2 (somewhere in the middle point of insert)

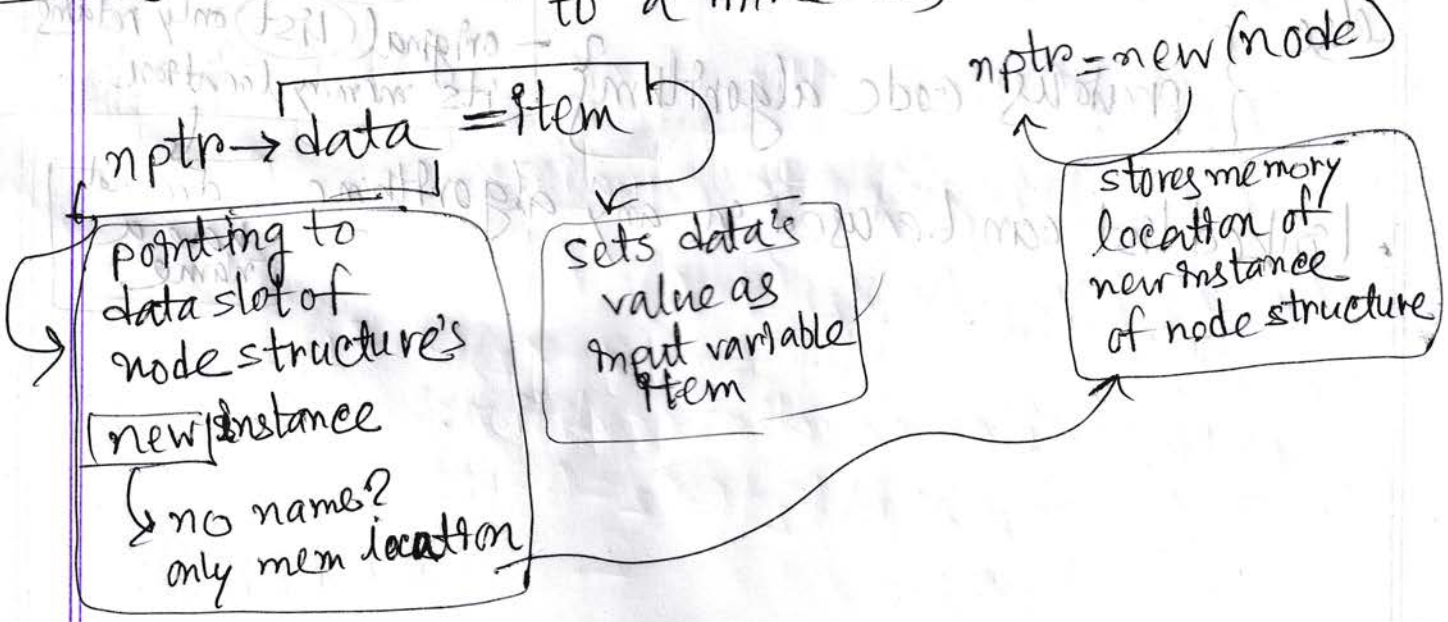
next → mem location of next list / slot



#Searching a node?

(P30)

Algorithm 4.3 (inserting a node to a linked list)



mid → until Linked list

corner case ← not address in the original code

#adding node at the header/starting/initial location →

```
if (lst -> data > nptr -> data)
  { nptr -> next = lst;
    lst = nptr;
  }
else { previous code algorithm }
```

- adding header mem location to nptr->next
- replacing lst mem location with nptr's
- now (nptr) → new header
- original (lst) only retains its memory location

- Linked list can be used in any algorithms → does not have a name

another corner case

#adding node at the end (next = NULL)

```
while( ... && tptr -> next != NULL)
{
}
```

→ breaks from loop when at the end (also Next = NULL)

```
if (tptr -> next == NULL)
{
    tptr -> next = nptr;
}
else
{
}
```

→ old code

DSA-1 10.03.25

Deletion of Linked List : similar 3 cases to consider

① Delete from first
② " middle
③ " Last

• PA.2

algorithm 4.4: for ② case

(item to be deleted)

```
tptr = lst;
while (tptr->data != item)
    { pptr = tptr;
      tptr = tptr->next; }
```

(deleting the node)

```
pptr->next = tptr->next;
next delete (tptr);
```

for (1) case

```
tptr = lst;
if (lst->data == item){
    lst = tptr->next;
    delete(tptr);
}
```

replacing header with next first node

frees the memory location (built in function in some languages) (Cpp)

(alt)

```
lst = lst->next
delete(lst)
```

deletes the lst node

Do Not Do It

for (3) case

```
while { }
if (tptr->next == NULL)
{ pptr->next = NULL;
  delete(tptr); }
```

making pptr/ previous pointer null

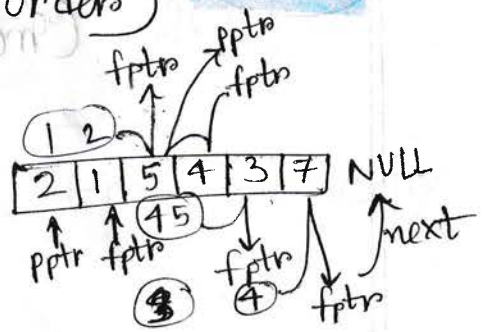
Sorting in Linked list?

Bubble sort

P4.3

algorithm 4.5: (Sorting in ascending order)

- fptr → latter pointer of pptr




```
fptr = pptr → next;
while (fptr != NULL)
  { if(pptr → data > fptr → data)
    { interchange (pptr → data, fptr → data);
    }
  }
```

Not a built-in function

algorithm notation for swapping

→ variable swapping

```
    fptr = fptr → next;
  } // end of while of step-5
pptr = pptr → next;
} // end of while of step-4
```

→ if breaks out from loop

***

Q. Sorting in descending order?

Next → doubly linked list
Circular linked list
(left before mid)

DSA-Lab (5) 11.03.25

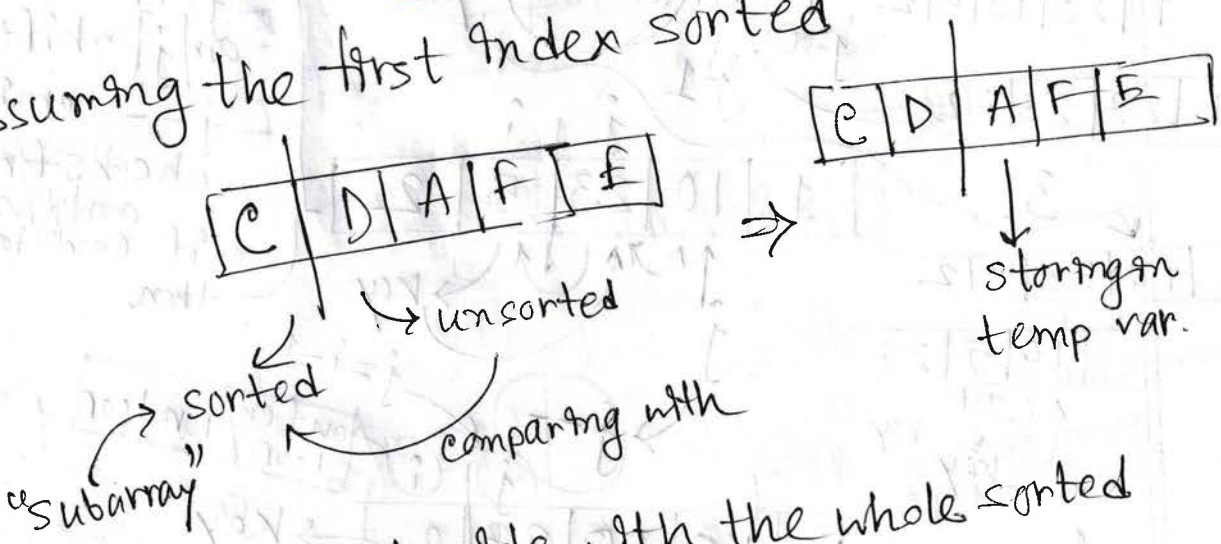
#Sorting Algorithm: (1) Insertion
(2) selection } $O(n^2)$
(3) bubble }

- multiple loop means polynomial time complexity

(1)

(1)

- assuming the first index sorted



- comparing unsorted side with the whole sorted side

skipping 1st step

```
if(ar[i] > ar[j]) { swap }
for(i=1; i<n; i++){ int key = arr[i];
                    int j = i-1;
while(j>=0 && arr[j]>key) {arr[j+1]=arr[j];
                           j--;}
arr[j+1]=key;}
```

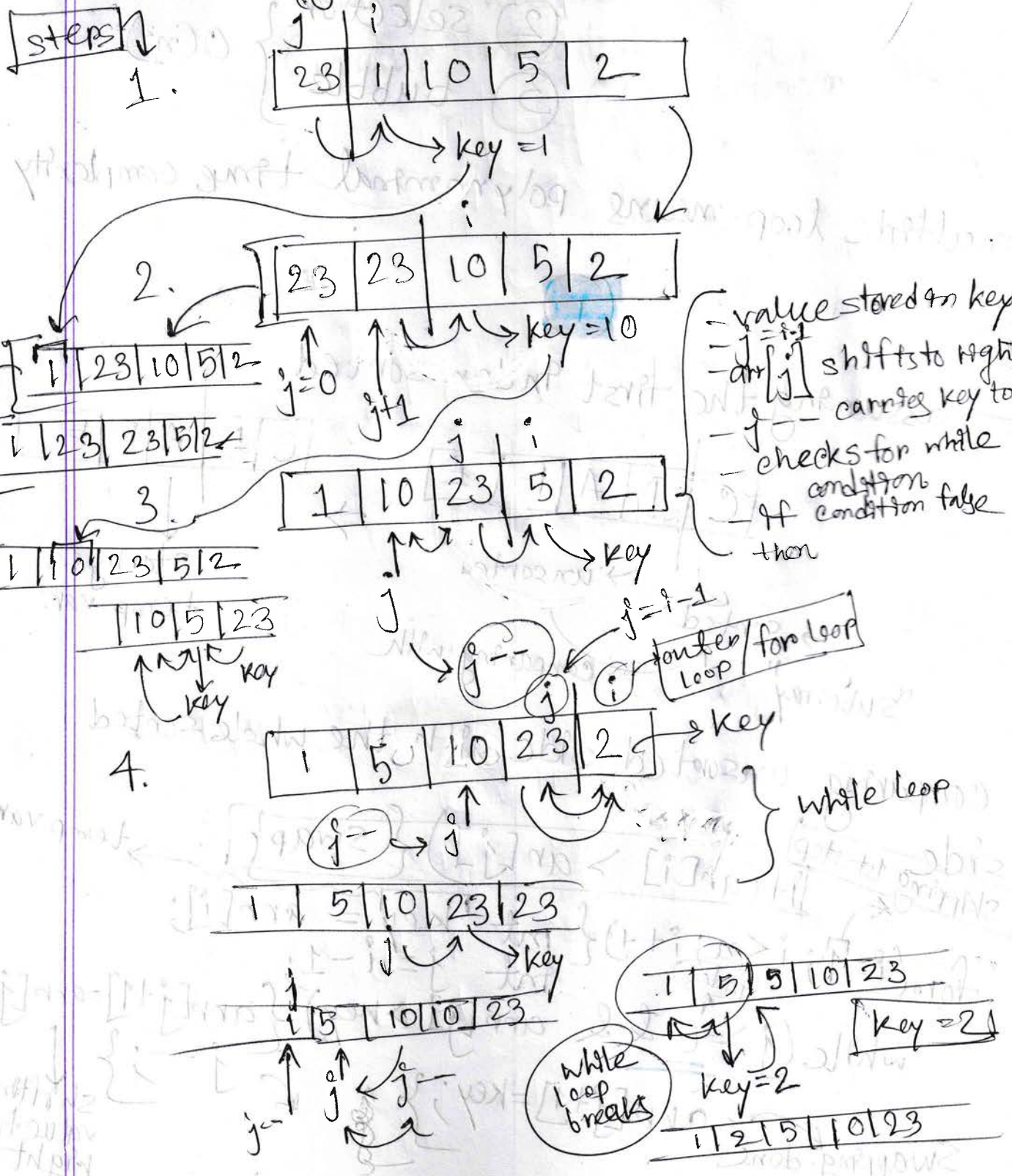
temp var.

index

shifting value to right

Swapping done

- If condition is true, shift the value to the right



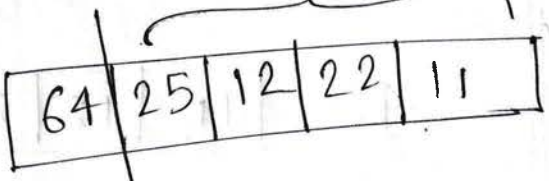
- we started with int i=0 instead of int i=1 as at i=0, j=-1 and in while loop condition (j>=0) is false so ar[j+1] = ar[0] = key meaning array element is kept where it was.

2 → select the min value among the unsorted subarray

outer loop i=0 → [n-1]

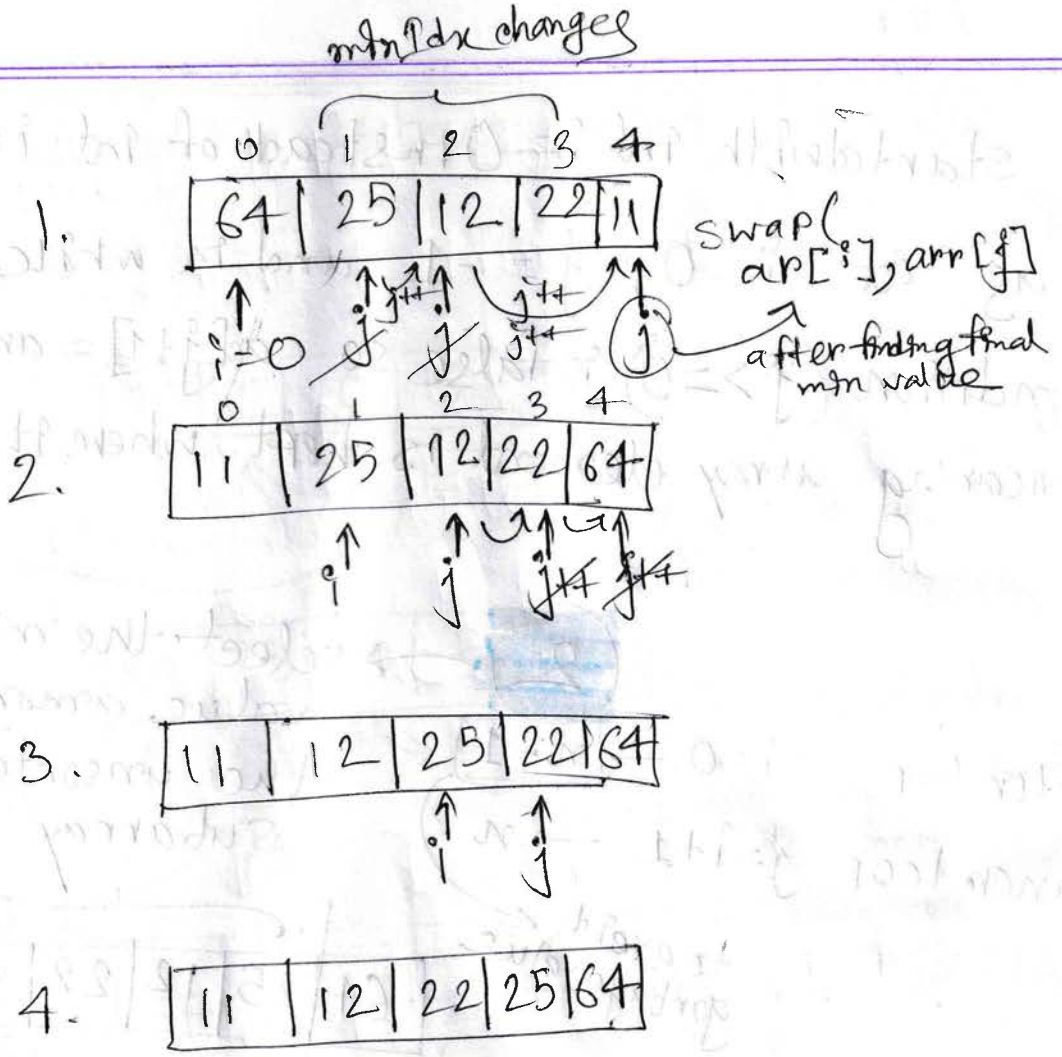
inner loop j=i+1 → n

to avoid garbage value



finding the index of the minimum value among unsorted side

```
for(i=0; i<n-1; i++)
{ int minIdx = i;
for(j=i+1; j<n; j++)
{ if(ar[j] < ar[minIdx])
{ minIdx = j; }
swap(ar[i], arr[minIdx]);
}
}
```

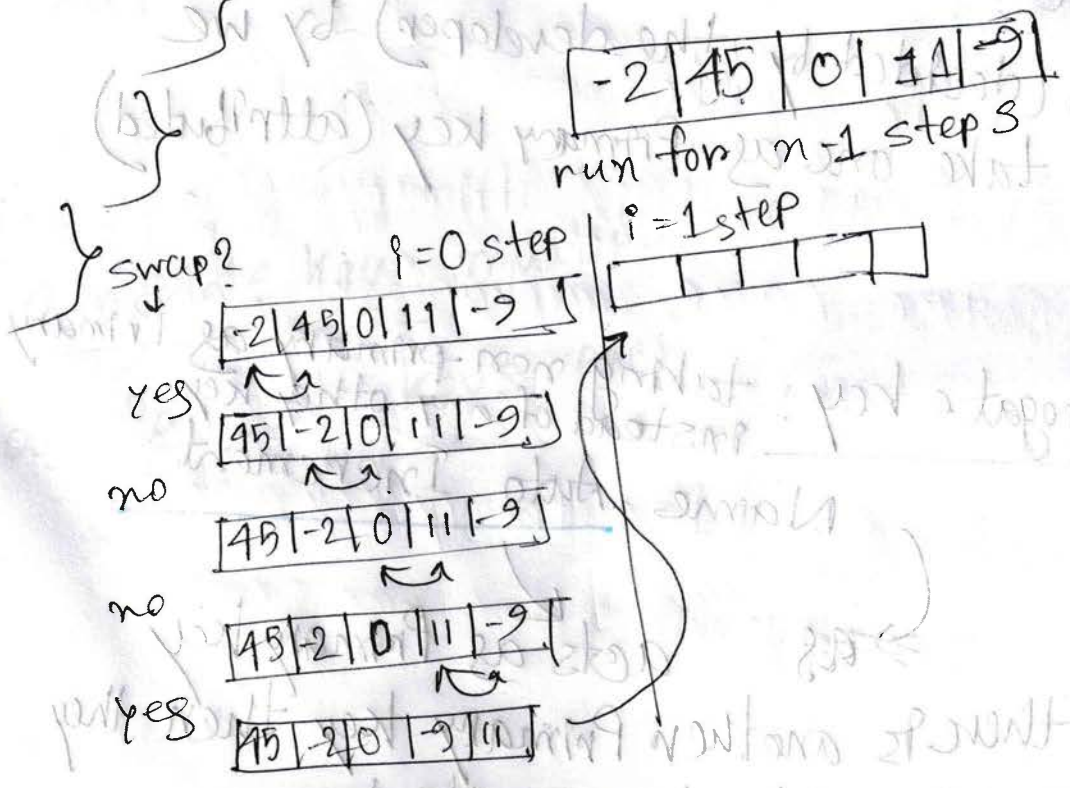


- Group project → Solo / group (max 4)
  - Idea (demo) will be shared can explore outside
  - add group details and github to excelsheet.

- **Assignment contest** in a week after mid (25 march)

3

```
for (int i=0; i<n-1; i++){          → full loop cycle
    for (int j=0; j<n-i-1; j++){    → j index shifts to left after each cycle
        if (arr[j] > arr[j+1]{
            swap (arr[j], arr[j+1]);
        }
    }
}
```

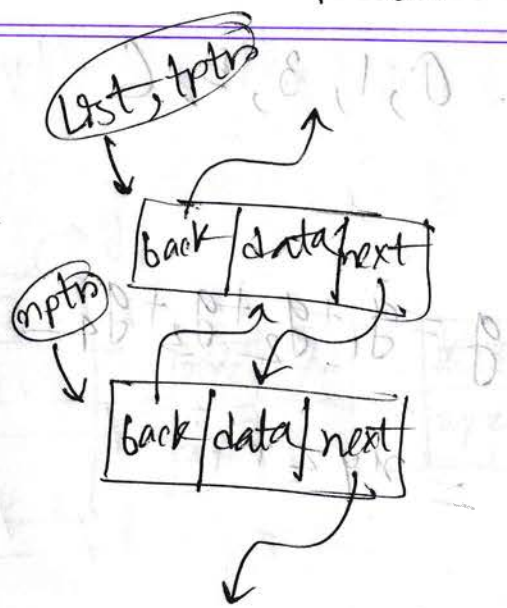


DSA-1 12.03.25

Doubly Linked List:

header and trailer
↓ ↓
back next

- both way traversal

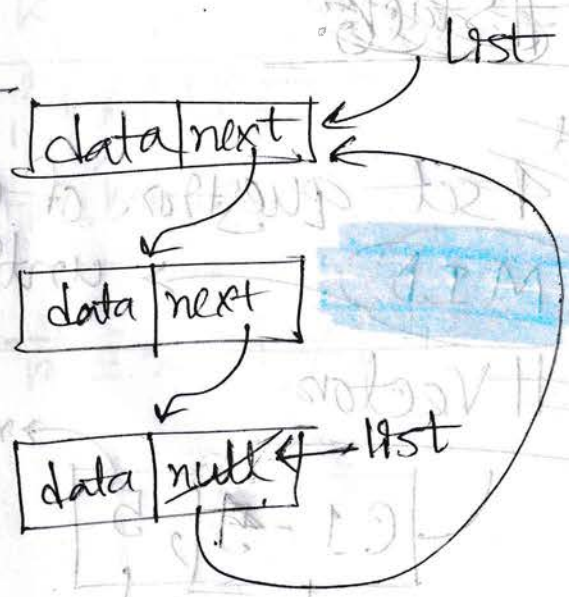


algorithm 4.6:

```
lst = NULL;
node *nptr;
nptr = new(node);
nptr → back = NULL;
nptr → data = Item;
nptr → next = NULL;
if(lst == NULL) {
    lst = nptr;
    tptr = nptr;
}
else {
    tptr → next = nptr;
    nptr → back = tptr;
    tptr = nptr; }
tail = tptr;
```

- can back propagate (if I need to search 997 of 1000 items, I can start from the end instead of top)

Circular Linked List:



algorithm 4.11:

```
last = NULL;
nptr = new (node);
nptr -> data = item;
nptr -> next = NULL;
if (last == NULL){
        last = nptr;
        nptr -> next = last;
        tptr = nptr;}
else{   tptr -> next = nptr;
        nptr -> next = last;   // differs from singly linked list
        tptr = nptr;
}
```

Mi dd → Schaum → introductory theory
Book highlighed parts.

Q. Array vs Linked list (P61)

DSA-1 08.04.25

Stack: LIFO = Last in (push) First out (pop)

- push operation : algorithm 5.1
- pop operation : algorithm 5.2
  - "underflow" or, "stack is empty"
    - when we want to operate even when there is no value to operate on?
    - not overflow?
    - underflow happens when we try to pop an item from an empty stack
- checking validity of arithmatic expression (using parentheses with pop and push)
  - Calculator's "Syntax Error"
- after "push" and "pop" operation, a stack must be empty, otherwise if there are elements left out, there must be parentheses remaining to be closed and therefore invalid.
- if "underflow" then invalid

- "infix" ? concern of precedence

operand - operator - operand

A + B

1 + 2 = 3

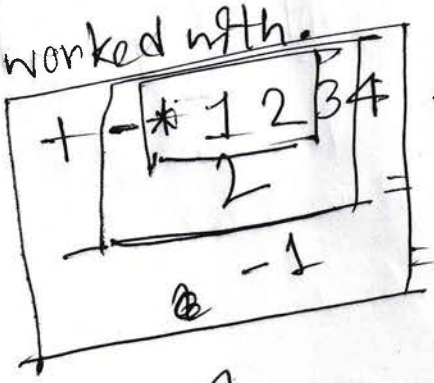
- postfix: (searches for operators from left to Right, and does operation on previous two)

AB+ = 1 2 + = 3

- prefix: (Right to left)

+AB = +12 = 3

after finding an operator, we expect to find an operand. Otherwise if there is another operator instead, then the final operator is worked with.



= + - (1 * 2) 3 4 = + - 2 3 4

+ (2 - 3) 4 = + (-1) 4

(-1) + 4 = 3

3 -1 + 4

Sign and operators are different

(-) (-)

DSA-1 09.04.25

Conversion to postfix form:

Page 77 (Table)

step-8:

- If operand → add to postfix (pop)
- If open parenthesis → add to stack (push)
- If closing parenthesis → add all (until open parenthesis) to postfix (pop)
- If operator → check if higher precedence then poped otherwise the higher precedence one is poped and current one added to stack instead.

3 algorithms, (we consider a math equation)

1. checking brackets (checking validity)
2. converting to postfix
3. evaluating postfix

- need to write algorithm + table process

Queue: can be implemented using

array and linkedlist

- push = enqueue (add)
  pop = dequeue (remove/delete)
- here FIFO is used

if (front == 0) //then we can push in the queue

{ front = 1; rear = 1;
queue [front] = 12; }

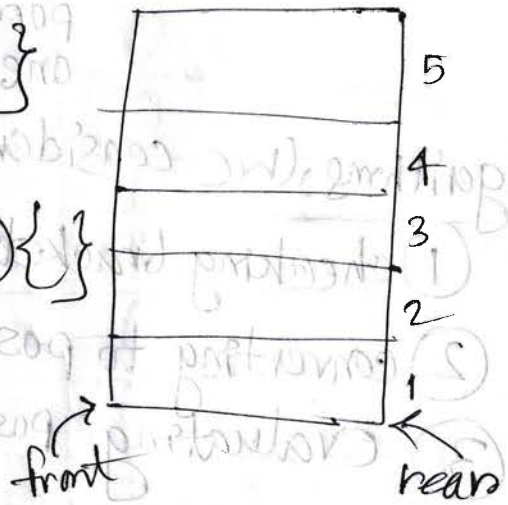
else if(

else if{ front = rear+1;

else { rear++; }

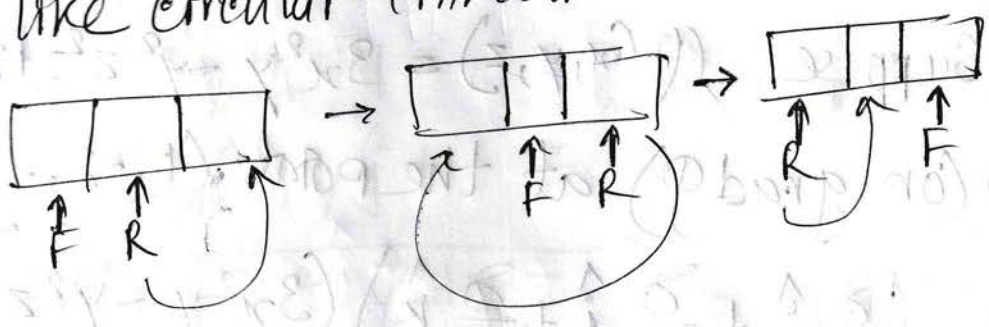
↳ queue [rear] = element

rear++;



- queue is usually taught as an infinite array (not circular)

- works like circular linkedlist



Algorithm: (for enqueue)

if (front = 1 && rear = N or front = rear + 1)
print (overflow ; return)
if (front = NULL) { front = 1; rear = 1; }
else if (rear = n) { rear = 1; }
else if (rear = n) { rear = 1; } → after dequeue, resets
{ rear = rear + 1; }
else
queue [rear] = item;

Next → dequeue

DSA-1 → FIFO 15.04.25

[PDF] Queue by ASM

#Deletion of an element from a queue:

- when front = 0, queue is empty
  or, "underflow"

Q. Static and Dynamic Programming?
list <list>

Q. type?

#enque = addition

#Deque: (double ended queue)
↓
accessible from both ends
(generalized version of queue)

① input restricted queue → input from only one end, deletion from both
② output " " → deletion from one end, insertion from both ends

- alt approach (not efficient): to Null the target slot and shift the others

#Priority Queue:

(1) Ascending order priority queue
(2) Descending order priority queue

lower priority number
and higher priority number } language?
in the priority

Q. (type)

→ Q. addition, deletion
Q. priority queue.

must show step by step

↓

Mid solve?

DSA-1 Lab 6 15.04.25

#Stack : LIFO

Stack is a container
linear data structure → static
can be implemented using primitive data structure

push(), pop(), peek()/top(), empty()
checking highest value

if top = -1 then stack is empty
arbitrary

- for each push():
  top++; //top = -1+1 = 0
  arr[top] = value //arr[0] = value

check if array has reached highest capacity:
top = size - 1; //stack is full

- for each pop():
  top--;

check if stack is empty (stop pop()):
if (top == -1);

(structure and class are similar)

using class for data structure;

- Access modifiers: Public, Private, Protected

↳ when nothing mentioned class is private

- constructor = method named as the class

- "public:" =

- preprocessors:

```
#define size 100
```

(no ;) — similar to constant keyword

- "return" works like "break" in loops, for functions

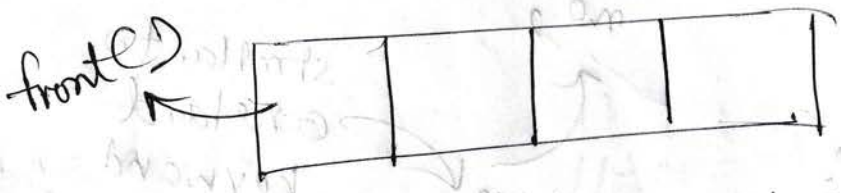
- we access stack elements from top to bottom

Ctrl+shift+↓ = copy above line in VScode

- object creation:

```
Stack S;
S.push(10);
S.push(20);
:
:
```

Queue: FIFO



variables
- front will update once, points to the front index
- back/rear updates per operation

methods:
- enque()
- dequeue()
- peek()/front()
- isEmpty()
- display()

front = -1; } queue is
back = -1; } empty

- when enqueue() used:

if(front==-1){front++;} // increments only when queue is empty /underflow
back++;

- check overflow:

if(back == size-1)
{queue is full.}

- when dequeue() used:

front++; // previous element ignored

- we cannot delete an element so we ignore the element (increment index)

- check underflow;

if(front > back)
{front = -1;
back = -1;}

- check top: arr[front]
- check if queue is empty : same as stack
- Output: print 0-(n-1) unlike stack

চলতি মাস

- Variable naming / function:
  1. isEmpty → camle case
  2. is_Empty → using underscore
  3. SIZE → preprocessor name standard
  4. is-Empty → using hyphen

- replace if block in isEmpty( );

```
return (front == -1); // when front = -1 it means empty and function returns true value
```

Lab Project

→ Details will be given at classroom

DSA-1 22.04.25

Mid review:

1(c) I was supposed to find time complexity n^2. I concluded at $80n^3$

2(b) Instead of algorithm, we can write the code. But code must be ~~core~~ correct. (A few wrote "c#" but the pattern did not require input.)

2(a) Traversing, Inserting, Deleting, and?

2(c) while( ) {

Pptr = fptr → next;

:

(Need to revisit algorithms, especially sorting algorithms)

PDF [sorting]

Sorting : (1) Internal Sorting
(2) External Sorting

(1) done in main memory or RAM

(2) stored in auxiliary memory such as hard disk, floppy disk...

- Stable and unstable sorting? → logic issue
- Adaptive and non-adaptive sorting? → runtime issue

Order : (1) Increasing
(2) Decreasing
(3) non-increasing → next element must be equal or smaller than the previous one

Algorithm : (1) Bubble Sorting
(2) Selection Sorting
(3) Insertion Sorting

- runtime and compile time?

① Input array A[1...n]

→ means array elements 1st to nth and not indexes as convention

we can write A[n].

② We need to "select" an element and then "sort" it by putting it somewhere.

```
for(i=1; i<=n; i++){                    → moves/changes
    small_index = i;
    for(j=i+1; j<=n; j++){              → "<" for descending order
        if(A[j] < A[small_index])
            {small_index = j;}          } finding the smallest element of the unsorted elements
    temp = A[i];
    A[i] = A[small_index];              } swapping the smallest of unsorted with the first element.
    A[small_index] = temp;
}
```

only index changing

DSA-1 Lab (7) 22-0425

#Types of Queue: (1) Simple Queue
(2) Circular Queue
(3) Priority Queue
(4) DEqueue (Double ended queue)

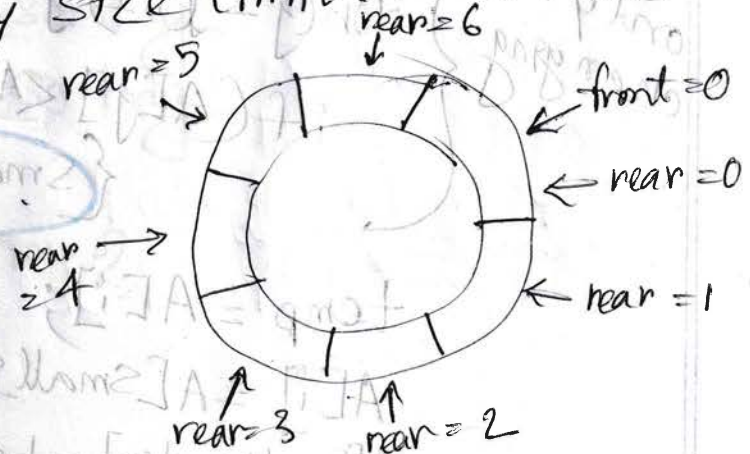
(1)

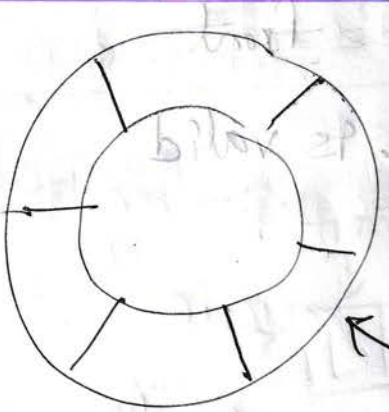
"dequeue" only shifts the index instead of actually deleting them. We cannot "deallocate" the memory.

(2)

overflow cases reset the index so the traversal circles within the array size limit. Better use of limited memory.

- front changes only once.





- Modulus theorem: to check if input is valid

(rear+1)%N

value returns in a circular manner.

value of the empty index

in circular queues, → has limitation

rear = (rear+1)%N

when, rear = 6 for N = 7

(6+1)%7 = 7%7 = 0 = ~~front~~ rear, → valid for enqueue

Why not else if?

```
if (front == 0 && rear == SIZE-1)
if (front == (rear+1)%SIZE)
```

=> to ensure that both conditions are checked? but return in the first if would skip the other returns?

overflow logic: when rear == front

no further enqueue is valid

new function: bool isFull()

→ outputs correct

false negative? false positive? => logic issue/ corner case

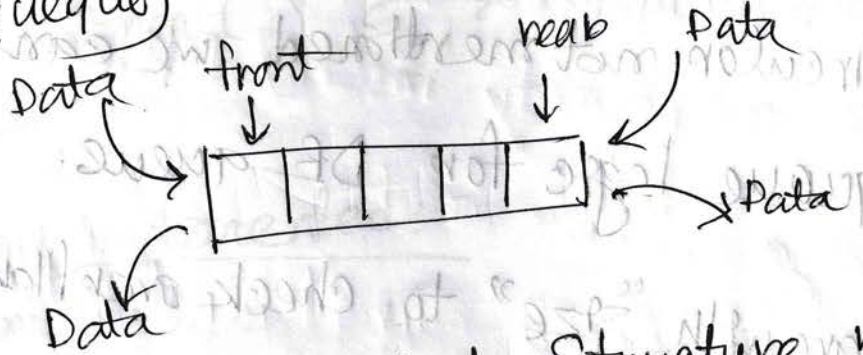
- return only returns single time and breaks out of the function.
- if (front == rear) // deQueue()
  {front = -1;
  rear = -1;}
  } means that queue has only one element and indexes need to be reset
- peek() displays the value of the front

HW → alternative to outputting last index of while loop inside inEmpty()

(4)

Double Ended Queue: Does not follow FIFO rule.

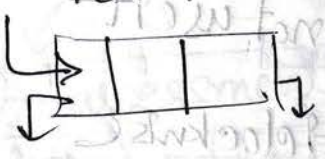
(DEqueque)



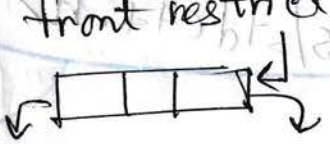
- It is a combined Data Structure with properties of both stack and Queue and follows the logic of Circular Queue.

Types:

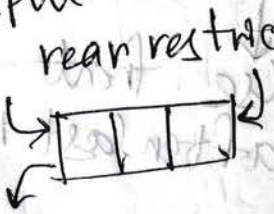
(1) Input restricted Deque:



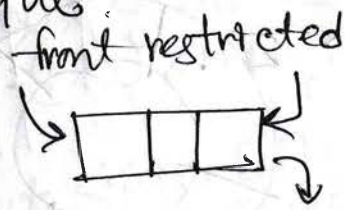
or;



(2) Output Restricted Deque:



or;



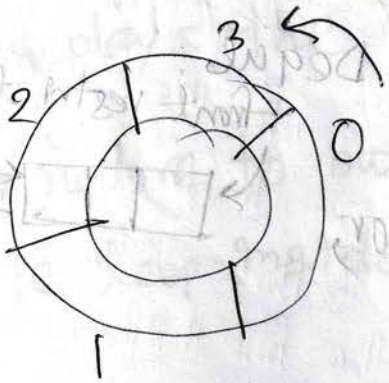
- ~~Capacity?~~ another variable "size" to track capacity
  ↳ might be instructed

Q. If circular not mentioned we can follow simple queue logic for DE queue:

- we compare with "size" to check overflow.

```
if (size == MAX) then
    Queue is full
```

- must use normal/standard header file for lab works. (using non-standard <bits/stdc++.h> will go against marks)
  → do not use it



anticlockwise insertion

↓

else front--;
(after last index)

- rear will not be used when calling "insertFront()" → Property of DEqueue.

functions implemented:

1. insertfront
2. insertRear
3. deleteFront
4. deleteRear

HW → Optimize the functions used in today's lab (using Modulas theorem)

for assessment

code:

```
int i = front;
while(i != rear){ cout << arr[i] << " ";
    i = (i+1) % MAX;
}
cout << arr[i];
cout << endl << "Rear->
```

#Priority Queue:

enque, deque used based on priority

- based on priority insert/delete elements

(1) Ascending order

↳ comparing current input value with previous value

```
if(ar[i] <ar[i+1])
        {ar[i+1] = ar[i];}
```

- constant time complexity

- input + sort happens for insertion

for deletion:

Priority number?

value → 7 2 9 5

priority → 1 3 2 4

Question type

value + priority pair used in structure (node type data structure)

compare + insert → while inserting / after inserting

DSA-1 theory

#Assignment: (3 codes)

1. Priority Queue (for ① ascending, ② descending order using priority number) ③

10 marks

2. Optimize the DEqueue

3. Making tutorial (screen record on mobile, zoom app)

3 minutes each code, total 3 codes.

under 10 minutes total.

→ file name: name Id section

→ Deadline: next class

Project Discussion:

Algorithms? (what we learnt was data structure)

Idea:

1. Management
2. Game → Guessing game, multiple player, scores
   - word completion
3. file system → Read-write
4. etc.

time limitation → timer function

Project:

1. Presentation on last class
2. Each group must submit a project report

→ elaborations on next class

"Don't make fun of other faculties"
"especially in front of other faculties"

DSA-1

23.04.25

#(3) Insertion Sorting:

- In "selection sort" i stays, j moves for each i loop. j moves/changes as "small_index" and the smallest value gets swapped with i.
- i will move until n-1, ∵ j = i+1
- here time complexity was: $O(n^2)$

algorithm: array A[0...n-1]

```
for(i=1; i<=n; i++)
{ j=i;
while(j>0 && a[j-1]>a[j])
{temp=a[j];
a[j]=a[j-1];
a[j-1]=temp;
j--;}
}
```

"<" for descending order

Q. Why Insertion sort?

- "Insertion sort" works by taking smallest value with j and bringing them at the beginning. So we are "inserting" values in a sorted manner in our theoretical "sorted list" that is being expanded by i++.

#Divide and Conquer Method:

- এগুলো যে algorithmগুলো ব্যবহার করলে worst case and not the most efficient.

(1) Merge Sort

(1) → algorithm? (wasn't in the syllabus before)

25 operations

Merge sort time complexity $O(n \log_2 n)$

Q. Given the characteristics, identify the sorting algorithm?

DSA-1 29.04.25

Tree: Top-down arrangement of root-node

(unlike ~~actual~~ real life tree (bottom-up))

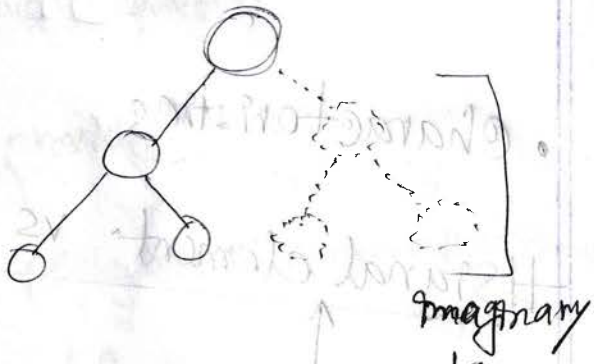
Binary tree: child node can be 0, 1, 2

"grand parent" and "grand child"

"Left child" and "right child"

2×n 2×n+1

Data structure:



level (k) = k

node per level = 2^n ← for binary tree

$2^0 = 1$, at k=1
$2^1 = 2$, at k=2
$2^2 = 4$, at k=3

height?

max no. of nodes = $2^k - 1 = n$

Q. given 31 nodes of a binary tree, find k?

$k = \log_2 32 = 5$

Q. what if 33?

$k = \log_2 34 = \lceil 5.08 \rceil = 6$

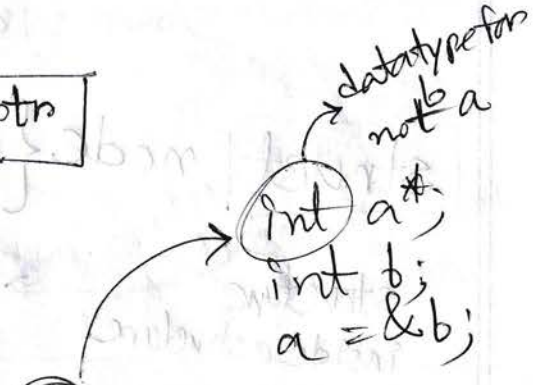
↖ we use ceil function

∴ for node n, level k,

$$k = \lceil \log_2 n + 1 \rceil$$

Data structure:


```
struct node{ int data;
             node *lptr;
             node *rptr;
           }
```



- array can hold similar type of data arr[int, int, int]
- structure can hold different type of data struct[int, float, char]

CT → Searching, Sorting, tree

06/07 May

DSA-1

30.04.25

CT → Searching, Sorting, Stack

Date 6.06.04.25

Full binary tree:

"If a binary tree contains nodes in such a way that every node has at least 2 children."

Complete binary tree: → tree-build

"Every node except the deepest level are full and deepest level is as left as possible."

"All full binary trees are complete binary trees".

Traversing = visiting every element of a data structure (most fundamental dsa operation)

- Shafayat's planet → blog for recursion → valid way to learn

#LLM

- Ad algorithm → knowledge graph
- Federate learning → next word prediction, Google vector embedding
  Adversal learning

query generator

- ecwid → website builder
- software firm: dev. team and ? (gets user feedback and suggests the dev. team for update)

- ChatGPT uses Transformers
ChatGPT-4 has added knowledge graph

Algorithm 7.1 : Preorder traversal method

Print → lchild → rchild

since printing happens before left child and Right child check

Algorithm 7.2: In-order traversal method

lchild → print → rchild

printing happens after left child check and befor Right child check

Algorithm 7.3: Post-order traversal method

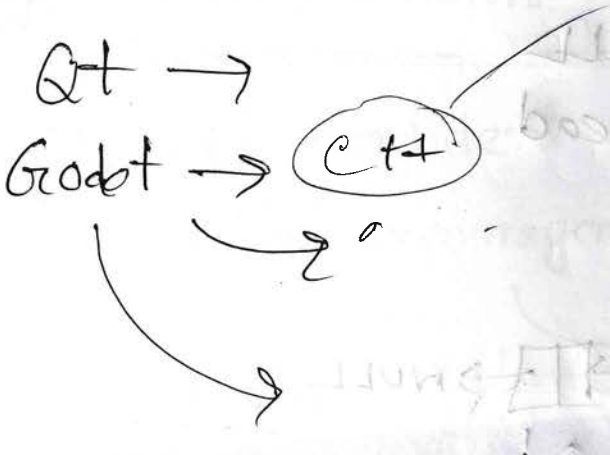
lchild → rchild → print

printing happens after left child and Right child check is done

Traversing? accessing node elements. printing is evidence of traversal.

DSA Lab (9)

06.05.25



#Linked List Insert operation:

- Insert at ~~beginning~~:

- InsertAtPosition: position does not mean index Linked list does not rely on indeces.

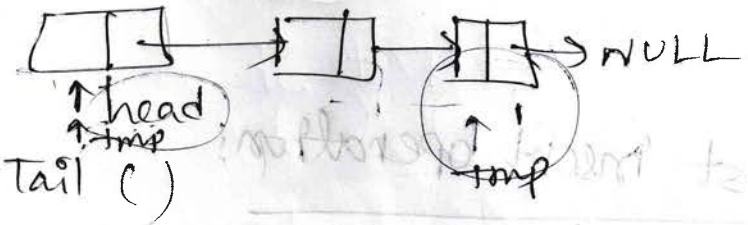
"new" keyword is used in dynamic memory allocation. We ~~dynamically~~ allocate memory when creating instances and object.

- for graph presentation we use "malloc"
- position

checking if any node inserted :

head == NULL
newNode = head

- insert



insertAtTail ( )

{

Node *tmp = head;

while(tmp->next != NULL)

{ tmp = tmp->next;

}

tmp -> next = newNode

- function for finding size of Linked list

video submission review:

- be more confident
- prize for encouragement
  - first

Presentation (Project):

Scoring Rubrics:

Project Report (5)
- marksan format
- 1 Report per Group

Presentation (15)
- Viva (5)
  - Q&A faculty + students
  - understanding
- Implementation (5)
  - must be able to explain every part
- Contribution (5)
  - be honest about it

Project Report

Cover Page:
- Title: "final project is about..."
- Student name and ID
- Section
- Session
- faculty information (name + designation) → both faculties
- Submission Date

Body:

1. Objective: 1-2 lines
2. Tools: mention whatever softwares used
3. Methodology:
   - Code
   - Code Explanation (of important parts)
4. Result: Output Screenshot (1)
5. Conclusion: fully implemented/completed or not
   - future features
   - Reasons of incompletion
6. Reference: Sources of Idea
7. GitHub Link: Repository Link.

sub-details can be added.

Jay: multiuser support; mmo games