

DSA(1)

29.01.25

using C / C++

C# .net framework

"The less source, the more demanding the work"

Egypt* framework (free)

"Competitive Programming is the most reliable"

not language

learn the concept and apply it

Documentation?

syntax

specific

better than Book?

language

concept

general

"5:5 rule" or "5% 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

"fatherly doubt"

"you're not smart enough"

pep talk

DSA book: IIT writer's Book

Sr will provide

Diploma or BSC?

classroom: no bg sks

DSA-1

04.02.25

Course outline:

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

Book: Schaum series + IIT এর লেখা গ্রন্থ

(containing mistakes)

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

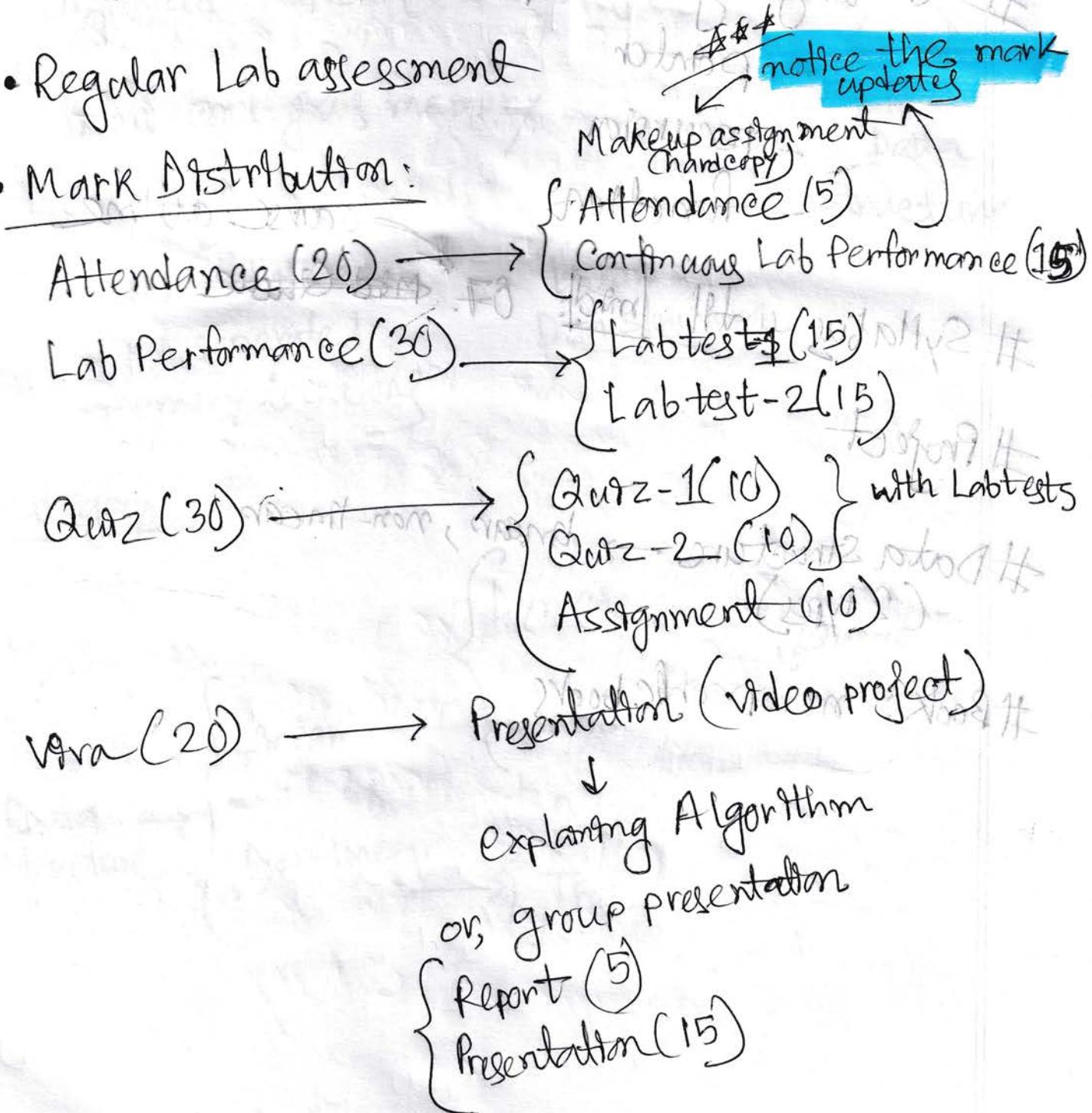
DSA-1 [Lab] 01

02.02.25

Introduction to Data Structure:

- Regular Lab assessment

- Mark Distribution:



C Language Concepts: (Important for DSA)

→ pointer

recursion

function

course outline

Syllabus until

mid:

07.

mid Queue

Project

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

Book: no specific books

maths I/A
maths II/B
maths III/C
Oxford
(C) Unit test

Organized sequential contiguous memory

non-contiguous memory

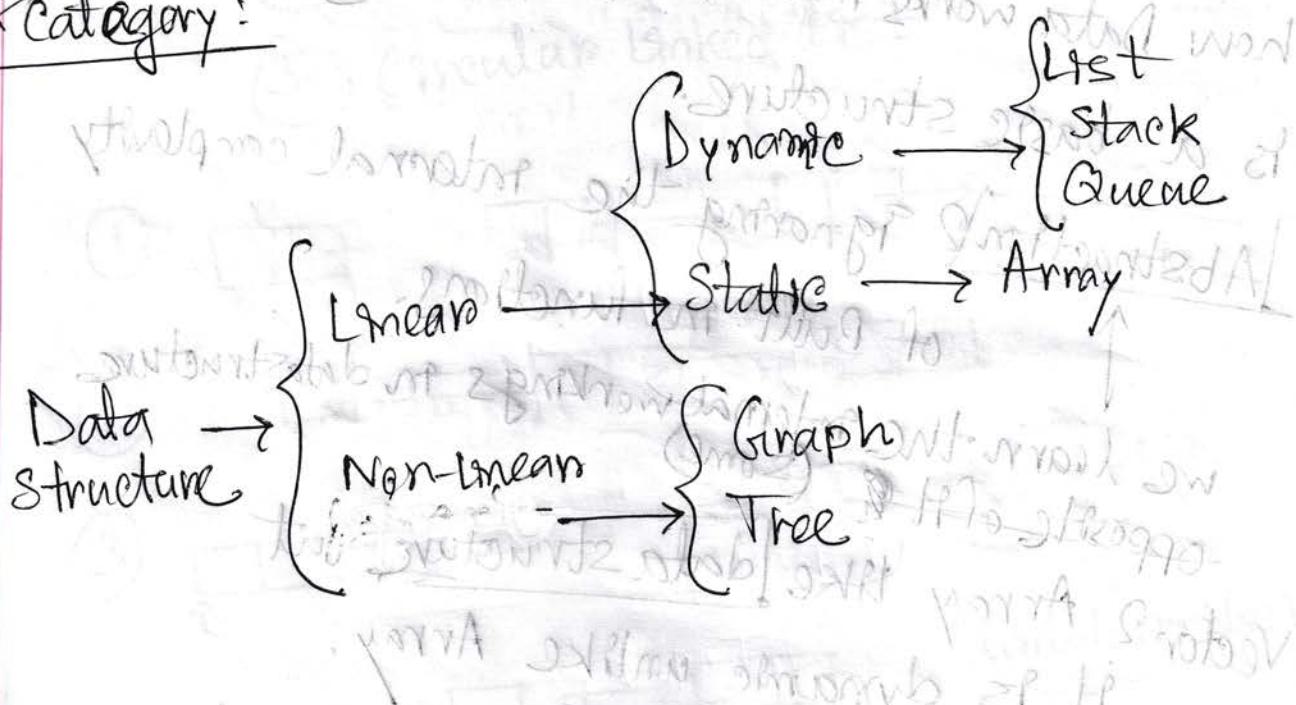
Linear Data Structure

Linked between two separated memory locations

non linear Data structure

Structure

Category:



Q. why learn Data Structures?

↳ Q. why learn Programming Language?

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

→ Applications of Data structures, learning
how Data works in computer operations, function

is a basic structure.

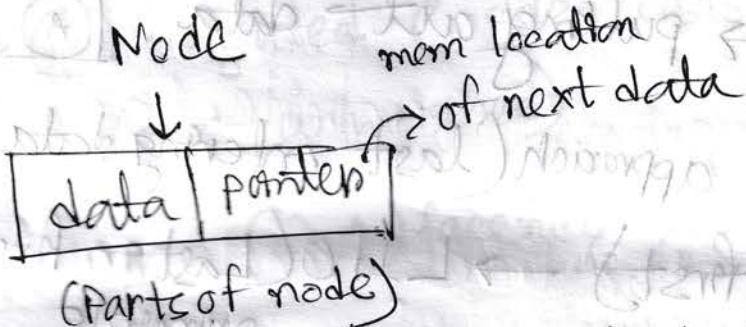
Abstraction? Ignoring the internal complexity

↑ of Built-in functions.
we learn the internal workings in datastructure
(some)

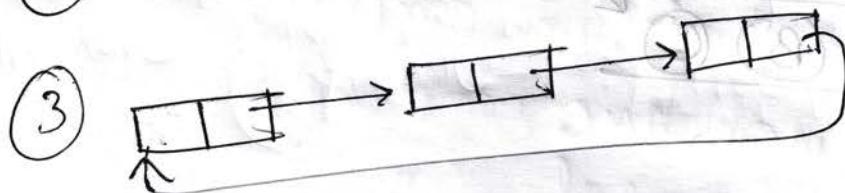
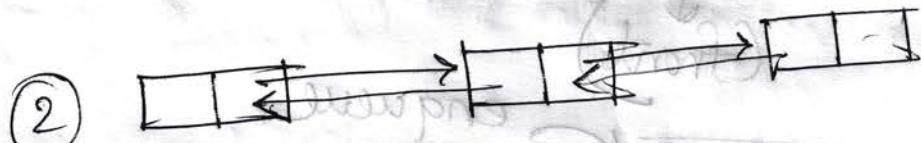
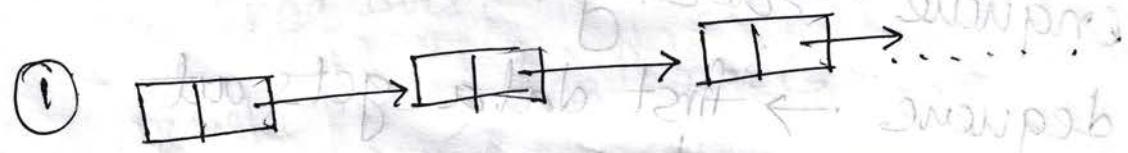
Vector² Array like Data Structure but
it is dynamic unlike Array.

Linked list 2 - first topic of non-linear data

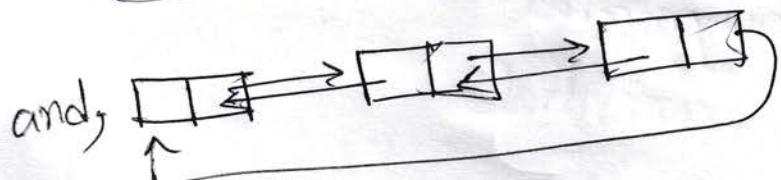
Structure



- types:
- ① singly connected Linked list
 - ② Doubly Linked list
 - ③ Circular Linked list



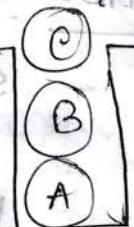
(Circularity)



Stack: (2nd)

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

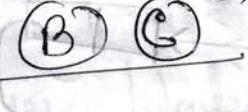
dequeue → first data gets out

↓
(front)

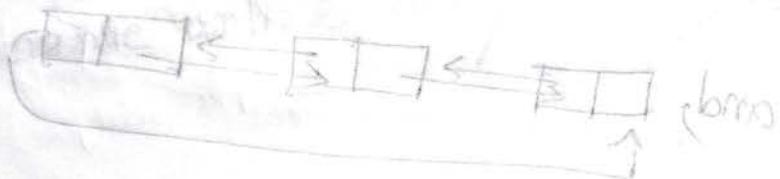
dequeue



enqueue



(first in first out)



Graph theory → Graph traversal

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

Representation: ① Adjacency list

② Adjacency matrix

Tree: graph does not maintain a hierarchy.

Tree does. Root node, Parent node

Child node, leaf node

(0)

Parent node and child = (1) 0

Root node and child = (n) 0

Isomorphic = (E_r) 0 (E_n) 0

Runtime

Time Complexity Analysis:

algorithm → set of instructions

↓
finite program (not infinite loop)

Definition: the time taken to execute the set of code, and not the total time

mathematical concepts

Asymptotic notation / relation:

DM book:
and most efficient

used to represent time complexity

- Big O (upper bound / worst case)
- Big Omega ($\Omega(n)$) (lower bound / best case)
- Big Theta ($\Theta(n)$) (tight bound) average case

Big O notation
the 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

~~Time complexity~~

$O(n \times m)$ notation indicates nested loops of n, m
 If $n=m$ then $O(n \times n) = O(n^2)$

code:

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

↑
outer
↓
inner

Polynomial
time
complexity

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

Linear search $\rightarrow O(n)$

Binary search $\rightarrow O(\log n)$

the most
("Not efficient")
as said in
Discrete mathematics

notive

$n \times n \rightarrow n \times n$

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 $O(V \times E)$

"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:

(2)

- Our concern is to eliminate the cause of

Big O

- Computer can explore/traverse per second.

$$1 \text{ s} = 10^7 \text{ steps}$$

$10^7 \approx 10^8$ step

↓ we consider

- Time limit depends on loops.

(loop & Time complexity)

lower upper limit

on Runtime

(2)

- ✓ Runtime and time limit?

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 (?)

[PPB11, 2010]

for($i=1; i \leq n; i++$) \rightarrow checks $(n+1)$ times

{
if statements \rightarrow checks (n) times
}

time complexity = $O(n+1+n)$

= $O(2n+1)$

$\approx O(n)$

↓

variable assignment

we ignore operation times. (ignores constants).

machines only concern with "the value of n ".

$$1s = 10^7$$

$$O(n)$$

$$\xrightarrow{10^7}$$

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

that is not how it is

about different constant

will be different

(1) O will always work

in terms of next function or function

programmed to do something else

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

$$O(n+m+2)$$

$$= O(n+m)$$

$$= O(n); \text{ if } (n > m)$$

for ($i = 1$; $i \leq n$;

{ }

for ($j = 1$; $j \leq m$;

{ }

$\therefore m$ cycles are counted
within n cycles

(does not separately check for m)
(does not operate

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

for ($i = 1$; $i \leq \sqrt{n}$; $i++$)

$$O(\sqrt{n})$$

{ }

for ($i = 1$; $i * i \leq n$; $i++$)

$$O(\sqrt{n})$$

{ }

• Tasks with time limit (CP problems)

• Online judge \rightarrow Problem solving sites like

Codeforces.

$O(\log n)$

for($i=1$; $i \leq n$; $i = i * 2$)

$O(\log n)$

for($i = 1$; $i \geq 1$; $i = i / 2$)

$$\frac{N}{2^k} \rightarrow 1$$

পুরোক ক্ষেত্রে দাপ্ত/ক্ষেত্র

means logarithmic

time complexity

$$\Rightarrow N \Rightarrow 2^k$$

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

$$\Rightarrow 1 \cdot k = \boxed{\log_2 N}$$

• logarithmic operations $\rightarrow 1s = 10^{-18}$

• \sqrt{N} $\rightarrow 1s = 10^{-14}$

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

Basic CPP Syntax:

C headers → `#include < stdio.h >`

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

`scanf() → cin >>`

`printf() → cout <<`

shifting operator (?)

HW

short explanation on (+ code)

$O(1)$

$O(N)$

$O(\sqrt{N})$

$O(N^2)$

$O(\log N)$

$O(n \log N)$

hard COPY

C1.3 (Schaum)

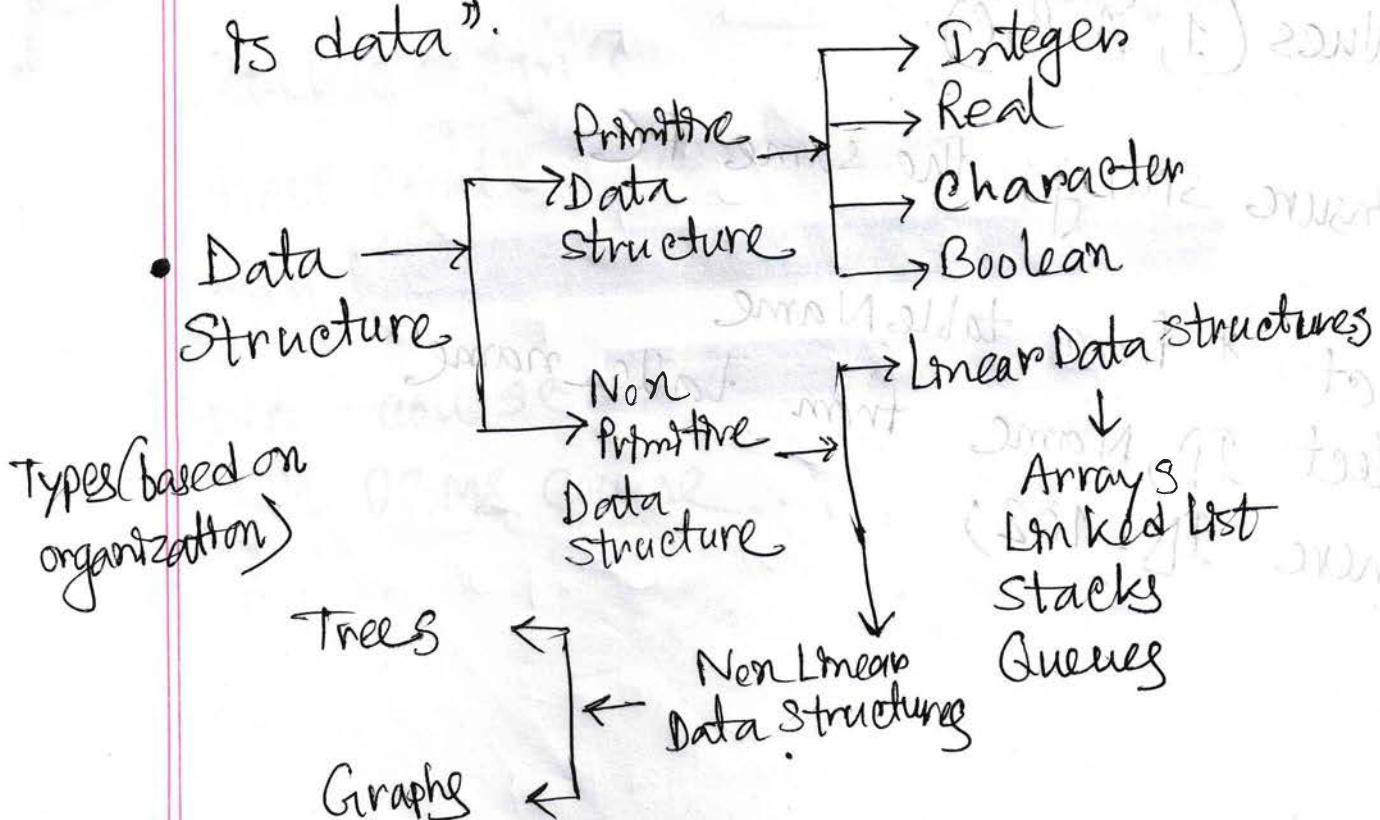
"The logical or mathematical model of a particular organization of data is called a data structure"

"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".



28.8.11

Data ADT

- 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/next
 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

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++)
```

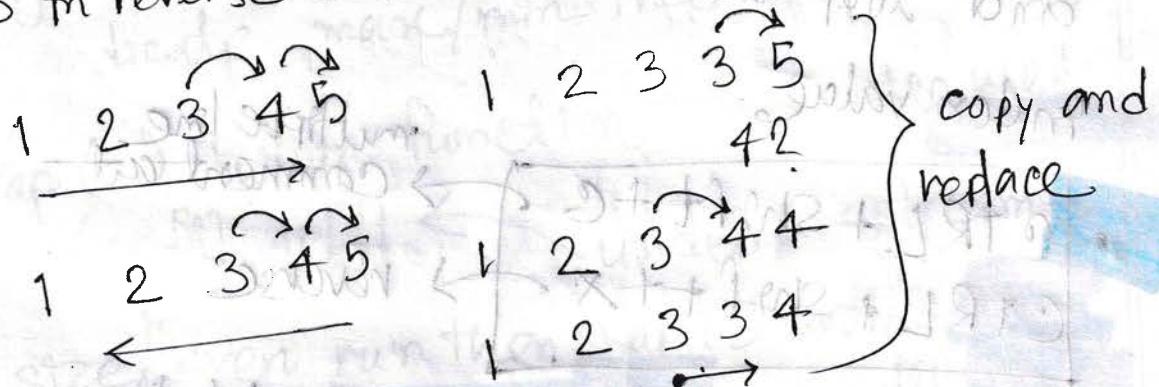
for loop scope

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

- Scope? access level?

- local
- global

Right shift ↗ 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 operators? eliminates the possibility of going out of bounds in array?

→ CPP doesn't show error

~~int f(i)(m>i) = { true if > pos > val };~~
 int pos, val; ~~can > 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] << " "; }~~

~~✓~~
~~Ctrl + Shift + C~~
~~Shift + X~~

Q3 Delete an element

• Shift to the deleted elements position

• Shift to the deleted elements position
and loop until the without the last

index value.

~~multiple line~~
~~Comment out~~
~~CTRL + Shift + C~~
~~CTRL + Shift + X~~ → reverse

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

(...)

cn > 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 structures)

- Using "Two pointers technique"

we indicate max/min index and replace

also used in
Binary Search

- swap() built-in function

swap(ar[i], ar[j]) ← under <i, ostream>

- 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 ($i : n - \rightarrow$)

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

reverse_arr reverse_array.epp

Passing parameters
in main function

i++ -> 0

reverse_

runs

array.0

int main (int argc, char *argv[])

Q. How to run C/CPP

from terminal/
with prompt from terminal?

(Main function with
argument)

$\checkmark O(n)$

~~#~~ O(log n)

exponential way (ଏ କୋଣ ଖେଳ କମଳେ) / କାହାଲେ

for (i=10; i<n; i=i*2)
 {
 : i = 10
 : i = 20
 : i = 40
 : i = 80
 : i = 160
 : i = 320
 : i = 640
 : i = 1280
 : i = 2560
 : i = 5120
 : i = 10240
 : i = 20480
 : i = 40960
 : i = 81920
 : i = 163840
 : i = 327680
 : i = 655360
 : i = 1310720
 : i = 2621440
 : i = 5242880
 : i = 10485760
 : i = 20971520
 : i = 41943040
 : i = 83886080
 : i = 167772160
 : i = 335544320
 : i = 671088640
 : i = 1342177280
 : i = 2684354560
 : i = 5368709120
 : i = 10737418240
 : i = 21474836480
 : i = 42949672960
 : i = 85899345920
 : i = 171798691840
 : i = 343597383680
 : i = 687194767360
 : i = 1374389534720
 : i = 2748779069440
 : i = 5497558138880
 : i = 10995116277760
 : i = 21990232555520
 : i = 43980465111040
 : i = 87960930222080
 : i = 175921860444160
 : i = 351843720888320
 : i = 703687441776640
 : i = 1407374883553280
 : i = 2814749767106560
 : i = 5629499534213120
 : i = 11258999068426240
 : i = 22517998136852480
 : i = 45035996273704960
 : i = 90071992547409920
 : i = 180143985094819840
 : i = 360287970189639680
 : i = 720575940379279360
 : i = 1441151880758558720
 : i = 2882303761517117440
 : i = 5764607523034234880
 : i = 11529215046068469760
 : i = 23058430092136939520
 : i = 46116860184273879040
 : i = 92233720368547758080
 : i = 184467407437095516160
 : i = 368934814874191032320
 : i = 737869629748382064640
 : i = 1475739259496764129280
 : i = 2951478518993528258560
 : i = 5902957037987056517120
 : i = 11805914075974113034240
 : i = 23611828151948226068480
 : i = 47223656303896452136960
 : i = 94447312607792904273920
 : i = 188894625215585808547840
 : i = 377789250431171617095680
 : i = 755578500862343234191360
 : i = 1511157001724686468382720
 : i = 3022314003449372936765440
 : i = 6044628006898745873530880
 : i = 12089256013797491747061760
 : i = 24178512027594983494123520
 : i = 48357024055189966988247040
 : i = 96714048110379933976494080
 : i = 19342809622075986795298160
 : i = 38685619244151973590596320
 : i = 77371238488303947181192640
 : i = 154742476976607894362385280
 : i = 309484953953215788724770560
 : i = 618969907906431577449541120
 : i = 123793981581286355489882240
 : i = 247587963162572710979764480
 : i = 495175926325145421959528960
 : i = 990351852650290843919057920
 : i = 1980703705300581687838155840
 : i = 3961407410601163375676311680
 : i = 7922814821202326751352623360
 : i = 1584562854240465350270526720
 : i = 3169125708480930700541053440
 : i = 6338251416961861401082106880
 : i = 1267650283392372280216413760
 : i = 2535300566784744560432827520
 : i = 5070601133569489120865655040
 : i = 1014120226713898241731310080
 : i = 2028240453427796483462620160
 : i = 4056480906855592966925240320
 : i = 8112961813711185933850480640
 : i = 16225923627422371867600961280
 : i = 32451847254844743735201922560
 : i = 64903694509689477470403845120
 : i = 129807389019378954940807690240
 : i = 259614778038757909881615380480
 : i = 519229556077515819763230760960
 : i = 103845911215503163952646153120
 : i = 207691822430006327905292306240
 : i = 415383644860012655810584612480
 : i = 830767289720025311621169224960
 : i = 166153457944005062324233844960
 : i = 33230691588801012464846768960
 : i = 66461383177602024929693537920
 : i = 13292276635204049848938715840
 : i = 26584553270408099697877431680
 : i = 53169106540816199395754863360
 : i = 106338213081632398791509326720
 : i = 212676426163264797583018653440
 : i = 425352852326529595166037306880
 : i = 850705704653059190332074613760
 : i = 1701411409306118380664149227520
 : i = 3402822818612236761328289455040
 : i = 6805645637224473522656578910080
 : i = 1361129127444894704531357782160
 : i = 2722258254889789409062715564320
 : i = 5444516509779578818125431128640
 : i = 1088903301955915763625862257280
 : i = 2177806603911831527251724514560
 : i = 4355613207823663054503449029120
 : i = 8711226415647326109006898058240
 : i = 17422452831294652218013796116480
 : i = 34844905662589304436027592232960
 : i = 69689811325178608872055184465920
 : i = 13937962265337721774110296891840
 : i = 27875924530675443548220593783680
 : i = 55751849061350887096441187567360
 : i = 111503698122701774192882375334720
 : i = 223007396245403548385764750669440
 : i = 446014792490807096771529501338880
 : i = 892029584981614193543059002677760
 : i = 178405916976322838708611800535520
 : i = 356811833952645677417223600107040
 : i = 713623667905291354834447200214080
 : i = 142724733581058270966894400428160
 : i = 285449467162116541933788800856320
 : i = 570898934324233083867577601712640
 : i = 1141797868648466167735155203425280
 : i = 2283595737296932335470310406850560
 : i = 4567191474593864670940620813701120
 : i = 9134382949187729341881241627402240
 : i = 1826876589837545868362483245404480
 : i = 3653753179675091736724966490808960
 : i = 7307506359350183473449932981617920
 : i = 1461501278670236694699865596323840
 : i = 2923002557340473389399731192647680
 : i = 5846005114680946778799462385295360
 : i = 11692010229361893557598924770590720
 : i = 23384020458723787115197849541181440
 : i = 46768040917447574230395698982362880
 : i = 93536081834895148460791397964725760
 : i = 187072163669790296921582795929451520
 : i = 374144327339580593843165591858903040
 : i = 748288654679161187686331183717806080
 : i = 1496577309358322375372662367435712160
 : i = 2993154618716644750745324734871424320
 : i = 5986309237433289501490649468742848640
 : i = 11972618474666578022891298937485697280
 : i = 23945236949333156045782577874971394560
 : i = 47890473898666312091565555749942789120
 : i = 95780947797332624183131111499885578240
 : i = 19156189559466524836626222299977115680
 : i = 38312379118933049673252444599954231360
 : i = 76624758237866099346504889199908462720
 : i = 153249516475732198693009778398176925440
 : i = 306498032951464397386019556796353850880
 : i = 612996065902928794772039113592707701760
 : i = 1225992131805857895540782227185415403520
 : i = 2451984263611715791081564454370830807040
 : i = 4903968527223431582163128908741661614080
 : i = 9807937054446863164326257817483323228160
 : i = 19615874108893726328652555634966666456320
 : i = 39231748217787452657305111269933332912640
 : i = 78463496435574905314610222539866665825280
 : i = 156926992871149810629204445078333317655680
 : i = 313853985742299621258408890156666635311360
 : i = 627707971484599242516817780313333310622720
 : i = 1255415942969198485334035560626666621255440
 : i = 2510831885938396970668071121253333342509840
 : i = 5021663771876793941336142242506666668519680
 : i = 10043327543733978826672844485033333337099360
 : i = 20086655087467957653345688970066666674998720
 : i = 40173310174935915306691377940133333349997440
 : i = 80346620349871830613382755880266666699994880
 : i = 160693240699743661226765511760533333399989760
 : i = 321386481399487322453531023521066666799979520
 : i = 642772962798974644907062047042133333599959040
 : i = 1285545925597949289814124094084266666719918080
 : i = 2571091851195898579628248188168533333439836160
 : i = 5142183702391797159256496376337066666879672320
 : i = 10284367404783594318512932752674133333759344640
 : i = 20568734809567188637025865505348266667518689280
 : i = 41137469619134377274051731010696533335037378560
 : i = 82274939238268754548103462021393133330745757120
 : i = 16454987847653750909620692404278666667551515440
 : i = 32909975695307501819241384808557333335103030880
 : i = 65819951390615003638482736017114666667056061760
 : i = 13163990278123007327691472034229333335101515520
 : i = 26327980556246014655382944068458666667023030400
 : i = 52655961112492029310765888136917333335046060800
 : i = 105311922224840586621531776273836666670115116000
 : i = 210623844449681173243063552547673333350230232000
 : i = 421247688899362346486127105095346666670460464000
 : i = 842495377798724692972254210190693333350920928000
 : i = 168499075559744938594458842038138666667184176000
 : i = 336998151119489877188917684076273333350983532000
 : i = 673996302238979754377835368152546666671967664000
 : i = 1347992604477959508755671216301093333350995332000
 : i = 2695985208955919017511342432602186666671990664000
 : i = 5391970417911838035022684865204373333351981332000
 : i = 10783940835823676070045369730408746666673962664000
 : i = 2156788167164735214009073946081746666677931332000
 : i = 4313576334329470428018147892163493333355862664000
 : i = 862715266865894085603629578432693333351725332000
 : i = 1725430533731788171207251556865386666673450664000
 : i = 3450861067463576342414503113730773333350901332000
 : i = 6901722134927152684829006227461546666670902664000
 : i = 13803444269854305369658012454923093333350905332000
 : i = 27606888539708610739316024909856186666671805332000
 : i = 55213777079417221478632049819712373333350910664000
 : i = 11042755415883444295726409643942573333350921332000
 : i = 2208551083176688859145281928788546666670826664000
 : i = 4417102166353377718290563857577093333350941332000
 : i = 8834204332706755436581127715154186666670826664000
 : i = 17668408665413510871162554430308373333350982664000
 : i = 3533681733082702174232510886061674666667165332000
 : i = 70673634661654043484650217721233493333350991332000
 : i = 14134726932330808696930423444246673333351982664000
 : i = 28269453864661617393860846888493373333350995332000
 : i = 565389077293232347877216937769866666671990664000
 : i = 1130778154586464695754433875337733333509995332000
 : i = 2261556309172929391508867750675466666673990664000
 : i = 452311261834585878301773550135093333351995332000
 : i = 904622523669171756603547100270186666670990664000
 : i = 1809245047338343513207094200540373333350995332000
 : i = 3618490094676687026414188401080746666670990664000
 : i = 7236980189353374052828376802161493333350995332000
 : i = 1447396037870674810565675360432296666670990664000
 : i = 2894792075741349621131350720864586666670990664000
 : i = 5789584151482699242262701441729173333350995332000
 : i = 11579168302965398484535402883458373333350995332000
 : i = 2315833660593079696907080576691673333350995332000
 : i = 4631667321186159393814161153383373333350995332000
 : i = 926333464237231878762832230676673333350995332000
 : i = 18526669284744637575256644613533493333350995332000
 : i = 370533385694892751505132892267673333350995332000
 : i = 7410667713897855030102657845353373333350995332000
 : i = 1482133542779571006020531569070673333350995332000
 : i = 29642670855591420120410631381413493333350995332000
 : i = 5928534171118284024082126276282673333350995332000
 : i = 11857068342236568048164252552565466666670990664000
 : i = 2371413668447313609632850510513093333350995332000
 : i = 47428273368946272192657010210261866666670990664000
 : i = 9485654673789254438531402042052373333350995332000
 : i = 189713093475785087706620408441047466666670990664000
 : i = 37942618695157017541324081688209493333350995332000
 : i = 7588523739031403508264816337641893333350995332000
 : i = 15177047478062807016529632673237866666670990664000
 : i = 3035409495612561403305926534647573333350995332000
 : i = 60708189912251228066118530692951466666670990664000
 : i = 12141637982450245613237060338590293333350995332000
 : i = 242832759649004912264741206771805866666670990664000
 : i = 48566551929800982452948241354361173333350995332000
 : i = 971331038596019649058964827087223466666670990664000
 : i = 194266207719203929811732965417444673333350995332000
 : i = 3885324154384078596234659308348893333350995332000
 : i = 77706483087681571924693186166977866666670990664000
 : i = 15541296617536353849338637233955573333350995332000
 : i = 31082593235072707698677274467911493333350995332000
 : i = 621651864701454153973545489358229866666670990664000
 : i = 12433037294029083079470909787165873333350995332000
 : i = 248660745880581661589418195743317466666670990664000
 : i = 49732149176116332317883639148663493333350995332000
 : i = 994642983522326646357672782973269866666670990664000
 : i = 198928596704465329271534556594653866666670990664000
 : i = 39785719340893065854306851318926773333350995332000
 : i = 795714386817861317086137026378535466666670990664000
 : i = 159142877363572263417274052476107093333350995332000
 : i = 3182857547271445268345481049522141866666670990664000
 : i = 6365715094542890536690962098544283466666670990664000
 : i = 127314301890857810733819241970885666666670990664000
 : i = 2546286037817156214676384839417713466666670990664000
 : i = 509257207563431242935276967883542666666670990664000
 : i = 1018514415126862845870553937767093466666670990664000
 : i = 203702883025372569174110787553418666666670990664000
 : i = 4074057660507451383482215751068373333350995332000
 : i = 814811532101490276696443150213674666666670990664000
 : i = 16296230642029805533928863004273493333350995332000
 : i = 3259246128405961106785772600854693333350995332000
 : i

for (int i = n; i >= 1; i--) {

$$\begin{array}{r} \overset{i}{\cancel{1}} \\ \overset{i}{\cancel{1}} - 1 = 0 \\ 1 \times 2 = \frac{1}{2} \\ 2 \times 2 = \underline{\underline{2^2}} \end{array}$$

$\overset{k-1}{\cancel{2}}$

nesting cases
↳ nested O (log n)

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

$$\Rightarrow k = \log_2 n + 1$$

~~#~~ $O(n \log n)$

for (i=1; i<n; i++) ← O(n)

$$\frac{\#O(\sqrt{n})}{\text{assume}}$$

for ($i=1$: $i < n$; $i = i \times 2$) $\leq O(\log n)$

$$i \times i = n$$

$$k \times k = n$$

$$\sqrt{n^2} = n$$

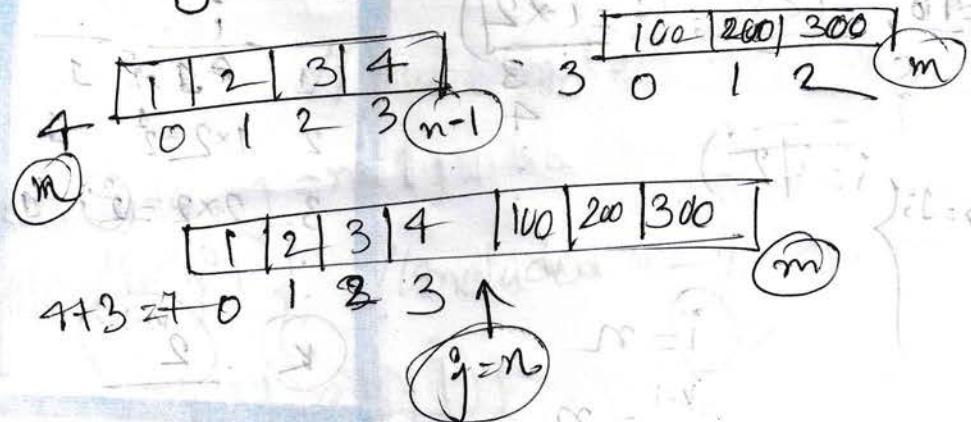
$n = \sqrt{n}$

for(i=1 ; $i \times i \leq n$, 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]; }  
for (int i = 0; i < m; i++) { cin >> b[i]; }  
int c[n+m]; for (int j = 0; j < n; j++) {  
    c[j] = a[j]; } int j = n;  
    c[j] = a[n]; } for (int i = 0; i < m; i++) { c[j] = b[i]; j++; }  
for (int i = 0; i < n+m; i++) { cout << c[i] << " "; }  
for (int i = 0; i < n+m; i++) { cout << c[i] << " "; }  
return 0;
```

CS, SO, PP

DBL

next

String as array

prefix array

multidimensional array

vector (dynamic)

Practice Problem → Online judge

case searching takes

O²n²

"boob" of boy pat

C2.3 Algorithmic Notations:

inspired from Pascal language

formal notation:

- (assign operator) $\leftarrow =$
 - (ending block) Exit $\leftarrow \}$
 - (Procedure / module) \leftarrow user defined function code
- matn function code
is called "Algorithm"

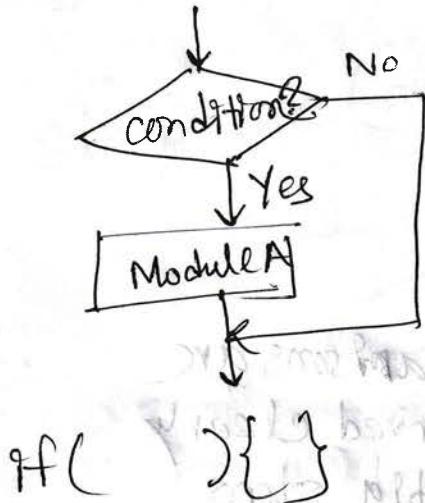
- (Input) Scanf
- (Output) Read \leftarrow Scanf
- Write \leftarrow printf

control structure (sequential flow, conditional flow,
iteration/ repetitive logic/flow)

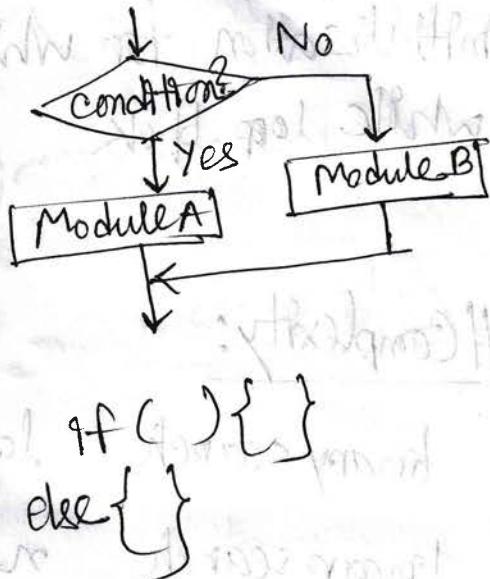
Top-down approach

if block / scope
python does not use {} for defining scope but indentation

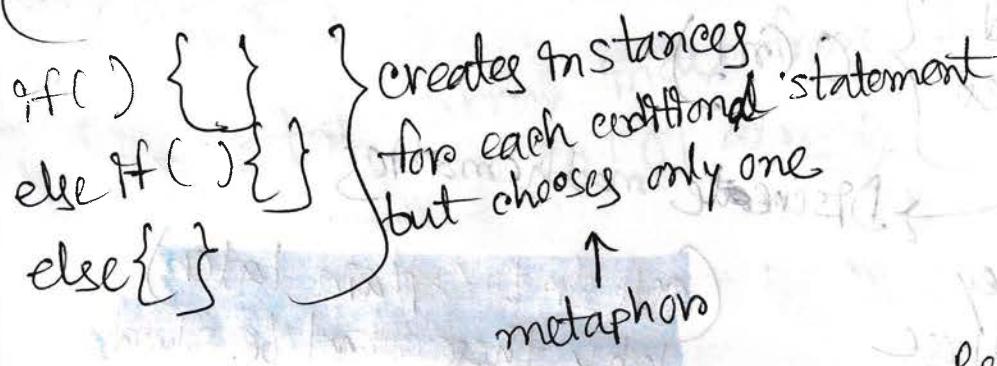
(single alternative)



(Double alternative)



(multiple alternative)



- Repeat for structure
- Repeat while structure

for loop → initialization

↓ condition check

↓ statement execute

↓ increment/decrement

loop -> condition check -> statement execute -> increment/decrement -> loop

Initialization for while loop happens before the
while loop block

#Complexity:

binary search $\log_2 n$
linear search n

big-O

comparisons are
observed clearly
for big steps
(no. of data)

NP-hard
NP-complex

$O(n!)$

Bounty money
on solving these
problems

Movie: Back to the future

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

1900s sci-fi stories

• Jimmy Fallon show \rightarrow lenovo

□ Assignment \rightarrow complexity graph plot
 $\log n, n, \text{in}(\text{lg} n), n^2, n^3, 2^n$ (Page 49) Book

#4 and #9

DSA-1 Lab

2D array, tag, element

string type (array problems) → from Task - 2

5	6	2	3	2
---	---	---	---	---

Tips: if

row → ar[i]

(flag not
needed as
 $a[0]$
man
skips that)

start array index from 1 since
 $a[0]$ already

faculty changed to? Oriti Ghosh
ma'am

63

DLD

19.02.25

multiplexer

multiplexer with logic II



19.02.25

DSA-1

entering

$$f(x) = y$$

$$f(2) = y$$

$$x^2 + 2 = y$$

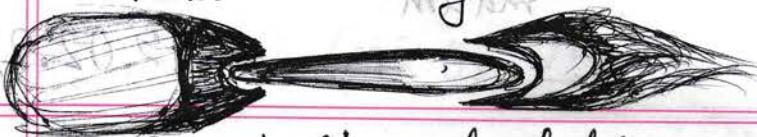
$$f(n) = 1$$

for every input the
output is 1
hence $O(1)$
each step takes
the constant
amount of time

$n \setminus g(n)$	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10 ³	10 ³
100	7	100	700	10000	10 ⁶	10 ³⁰
1000	10	10 ³	10 ⁴	10 ⁶	10 ⁹	10 ³⁰⁰

fig. 2.6 - Rate of Growth
of Standard functions

"wait in thought" ~~of big words about~~



Complexity calculations

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

positive value

high bound

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

f(n)

gen)

- 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

000	001	010	011	100	101	110	111
000	001	010	011	100	101	110	111
000	001	010	011	100	101	110	111
000	001	010	011	100	101	110	111

Network to start - 2.8 off
2nd best branch to

C 2.6 - Best case scenario / lower bound :

Omega notation

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

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

$$18n + 9 \geq 18n$$

\downarrow $\rightarrow g(n)$

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

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

\downarrow \downarrow
 $M_1 g(n_1)$ $M_2 g(n_2)$
 $= M_1 g(n)$ $\geq M_2 g(n)$

Time complexity is basically the time required for an algorithm to complete executing all steps. Big(O) means highest required time for the corresponding input value. Big(Omega) means lowest required time. 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 "less" efficient and can only be towards Omega and Theta (can only improve).

IIT writer's book → C-2

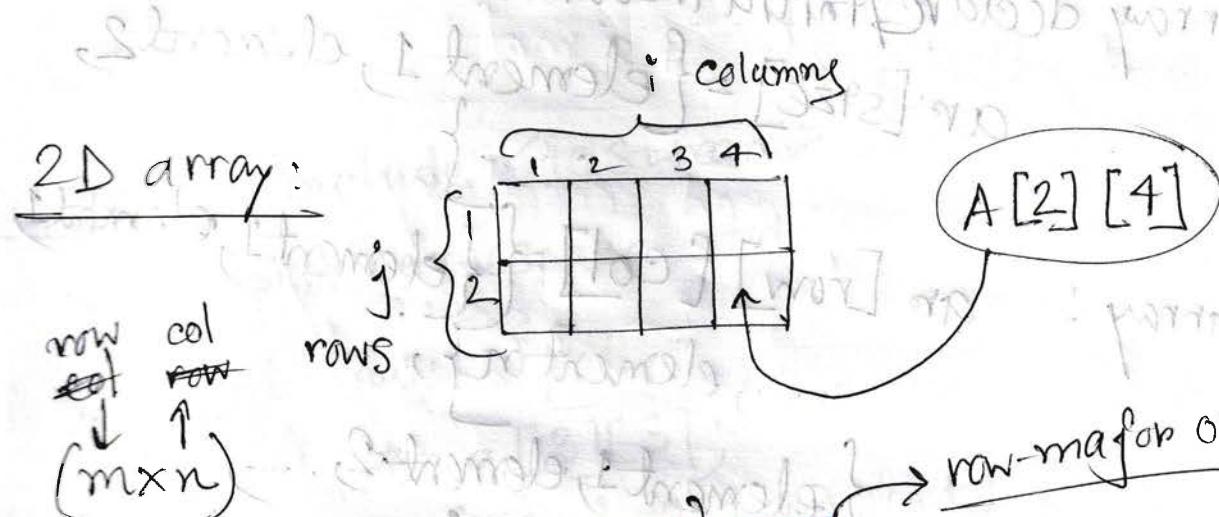
Array:

- computer remembers only the starting index.
- each increasing index increasing size by 4 bytes.
- to find a memory location of a specific index, let, $A[997]$; $1000 + (997-1) \times 4$.
- formula: $= 4984$ → mem address
- in 1D array, each location takes 4 byte space

28.02.22

DATA STRUCTURE

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



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

starting value

Base(A)

why (-1)?

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

column-major order

datatype size (4 for int)

Task:

watch yt video (previous class, time complexity)

Multidimensional Array: (2D array)

1D array declare/initialization:

$$\text{arr}[\text{size}] = \{\text{element}_1, \text{element}_2, \dots\}$$

2D array: $\text{arr}[\text{row}][\text{col}] = \{\{\text{element}_1, \text{element}_2, \dots, \text{element}_n\}, \{\text{element}_1, \text{element}_2, \dots, \text{element}_n\}, \dots\}$

The diagram illustrates the mapping of a 3x3 2D array to a 3x3 grid of indices. On the left, a 3x3 grid of numbers (2, 3, 4; 5, 6, 7; 8, 9, 11) is shown with 'row' labeled vertically and 'column' labeled horizontally. An arrow points from this grid to a 3x3 grid of indices on the right, where each index is represented as a pair of row and column values: (0,0), (0,1), (0,2); (1,0), (1,1), (1,2); (2,0), (2,1), (2,2).

2	3	4
5	6	7
8	9	11

3x3

(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++)

 { for (int j=0; j<col; j++)

 { arr[i][j]; }

 }

3D array: $\text{arr}[][][]$

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++)
        {
            cout >> 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

	0	1	2	
0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	secondary diagonal
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$i+j+1 = \text{row/col}$
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	= for square mat

primary diagonal
↓
 $i=j$

to understand the pattern of secondary diagonal:

0,0	0,1	0,2	0,3	row/col = 4
1,0	1,1	1,2	1,3	$2+1=3 = \text{row}-1 = \text{col}-1$
2,0	2,1	2,2	2,3	$i+j = \text{row}-1$
3,0	3,1	3,2	3,3	$i+j+1 = \text{row}$

4x4

Triangular Matrix?

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

not diagonal if i 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

$\text{transp}[i][j] = a[i][j]$

Transpose matrix?

1	2	10
2	7	8
3	8	11



1	2	3
2	7	8
10	8	11

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

Task

Symmetric mat?

If $\text{mat} = (\text{mat})^T$

$$\begin{array}{|c|c|c|c|} \hline & 1 & 2 & 3 \\ \hline 1 & 2 & 7 & 8 \\ \hline 2 & 3 & 8 & 5 \\ \hline \end{array} \xrightarrow{\substack{\text{Transpose} \\ \text{pose}}} \begin{array}{|c|c|c|c|} \hline & 1 & 2 & 3 \\ \hline 1 & 2 & 7 & 8 \\ \hline 2 & 3 & 8 & 5 \\ \hline \end{array}$$

Condition for mat addition/subtraction,

$$\begin{matrix} m = n \\ \uparrow \quad \uparrow \\ \text{row} \quad \text{column} \end{matrix}$$

mat add/subtract?

$$\begin{array}{|c|c|c|} \hline & 0 & 1 & 2 \\ \hline 0 & a_{0,0} & a_{0,1} & a_{0,2} \\ \hline 1 & a_{1,0} & a_{1,1} & a_{1,2} \\ \hline 2 & a_{2,0} & a_{2,1} & a_{2,2} \\ \hline \end{array}$$

$$+ \begin{array}{|c|c|c|} \hline & 0 & 1 & 2 \\ \hline 0 & b_{0,0} & b_{0,1} & b_{0,2} \\ \hline 1 & b_{1,0} & b_{1,1} & b_{1,2} \\ \hline 2 & b_{2,0} & b_{2,1} & b_{2,2} \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|} \hline & 0 & 1 & 2 \\ \hline 0 & a_{0,0} + b_{0,0} & a_{0,1} + b_{0,1} & a_{0,2} + b_{0,2} \\ \hline 1 & \dots & \dots & \dots \\ \hline 2 & \dots & \dots & \dots \\ \hline \end{array}$$

mat multiplication?

Condition: $(m \times n) \times (n \times m)$
 result: $(m \times m)$

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

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

	$a_{0,0} x^{b_{0,0}}$	$a_{0,0} x^{b_{0,1}}$
=	$+ a_{0,1} x^{b_{1,0}}$	$+ a_{0,1} x^{b_{1,1}}$
	$+ a_{0,2} x^{b_{2,0}}$	$+ a_{0,2} x^{b_{2,1}}$
1	$a_{1,0} x^{b_{0,0}}$	

$\text{at}[0,1]$

3 loops;

→ loop-1 → row traversal
loop-2 → column traversal

loop-1 → row traversal
loop-2 → column traversal
loop-3 → index pair

loop-3 → changing index pattern when multiplying

$$n_{\text{mat 1}} = m_{\text{mat 2}}$$

for multiplication <

$\text{mat}[i][j] += a[i][k] \times b[k][j]$

$\text{mat}[i][j] = \text{un}$

$$\textcircled{6} \quad \text{Ex. } \begin{array}{r} 123 \\ \times 654 \\ \hline 789 \end{array}$$

$$(9+12+9) \quad (8+10+6) \quad ()^3 = 30,24$$

for (int i = 0; i < row; i++)

```

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
}

```

```

    { for( int j=0; j < col; j++)
    {   for( int k=0; k < row; k++)
        {
            ...
        }
    }
}

```

```

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

```

DSA-1

④

26.02.25

Linked List:



2 pointers forward & backward

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~~

$\begin{matrix} 1+5 \\ 2+4 \\ 3+3 \\ 4+2 \\ 5+1 \end{matrix} = 6 = m+1 \text{ or } n+1$

```
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

88

"contributions' of others' behind a career"

for both diagonal:

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

look for
more
problems
at w3resources

print output:

1

2 3

4 5 6

7 8 9 0

1 2 3 4 5

code

code

/Algorithm

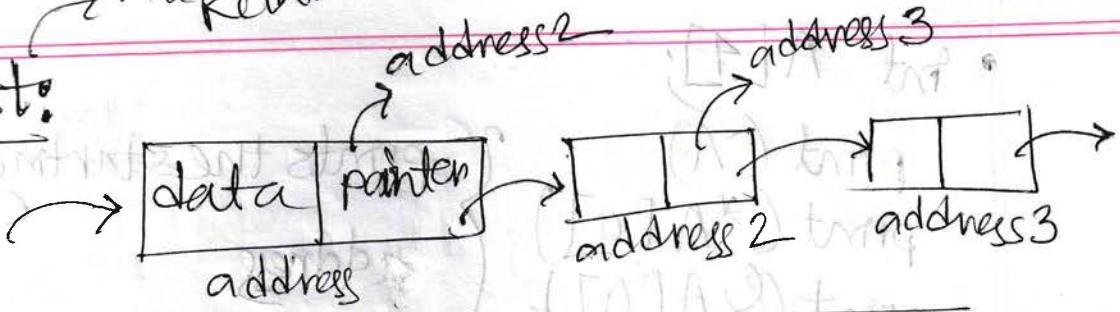
```
int K=1;
for (i=1; i<6; i++)
{
    for(j=1; j<=i; j++)
    {
        cout << K << " ";
        K++;
    }
    cout << endl;
    if (K==10) { K=0; }
}
```

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");
    }
}
```

→ kinda like
Relative links in WAD

#LinkedList:



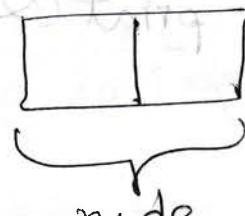
- we will use struct

```
struct node { method
```

```
    int data,
    node* next; } object-like
```

```
; int i, n, item;
```

```
node *nptr, *pptr, *first;
```



- can one datatype pointer point to another datatype(?)

- variable is a pointer

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

- **node 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

Searching algorithm/technique:

① Linear search $\rightarrow O(n) \rightarrow$ can be applied on unsorted array

② Binary search \rightarrow Divide and conquer

worst case: $O(\log n)$

best case: $O(1)$

$$\text{floor}(7.5) = 7$$

$$\lceil \text{ceil}(7.5) = 8 \rceil$$

Prerequisite: Sorted array
 ① check if the target value is equal to the mid value or not

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

③ floor(.) values taken as mid point.

④ level by level division

(to avoid overflow)

alt:

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

(alt) can use $\text{floor}()$
 built-in function
 (alt) can just divide $\frac{\text{high} - \text{low}}{2}$

Linear search: Avg/Worst $O(n)$
 Best $O(1)$

① 1D array input

② flag (bool type) to check found/not

[int range will take only int part if float]

can cause overflow for large data

Bool type variable

True / False] only 2 values
1 0

LAB Test

Next week (15 mark)

04.03.25
(08.04.25)

Quiz

(10 mark)

mark

Iterative method

Binary Search:

$n=13; \text{arr}[n] = 0, 5, 7, \dots, 12, 15, 17, 20, 22, 25, 27, 30, 32, 35, 37, 40, 42, 45, 47, 50, 52, 55, 57, 60, 62, 65, 67, 70, 72, 75, 77, 80, 82, 85, 87, 90, 92, 95, 97, 100$

(1) left = 0

(2) right = $n - 1$

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

Time complexity

Array 1D

Array 2D

Searching technique

Prefix array

String Techniques

+ next class topic

if

but still $a[0] < n$

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)

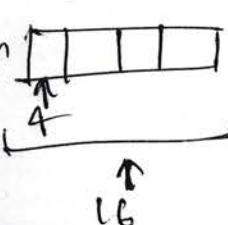
• sorting function:

• sizeof()

• to pass array as : arguments

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

↓
16↓
4

when arr [] 

$n = \frac{16}{4}$

arr > data; / To find
 algorithm:
 int n = sizeof(arr)/sizeof(arr[0]);
 binary search (arr, data, n);
 binary search (int arr[], int data, int n)
 {
 while ($l \leq r$) int l = 0;
 int r = n - 1;
 while ($l \leq r$) {
 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 problem



int arr →

0	1	2	3	4	5	6
3	7	5	2	7	13	14

int pre[] →

3	10	15	17	24	27	31
---	----	----	----	----	----	----

$$pre[0] = arr[0]$$

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

$$l = 2$$

$$R = 5$$

$$\text{if } (L == 0)$$

{ cout << pre[R] }

else {

$$pre[R] - pre[L-1]$$

to avoid
garbage value

loop 1 ($i=0; i < n$)

cn >> array

loop 2 ($i=1; i < n$)

cn >> cumulative
sum array

loop 3 (cn >> query
/ test case

and cn >> range
from pre[1])

overall nested
loop ($O(n^2)$ cause)
and a single loop makes
 $O(n)$

Page - 27 (Program to create a linked list)

- node type list

- "list" pointer and set as =NULL

- nptn = new(node) // new node creation

Keywords

NULL → blank value → struct instance

new → new [instance] creation

(*) next

• → pointer operator ? works like this
 ↳ references to instance variable

• list ← initial value

• (i++ and (i++) instead of loop expression) → does not matter

• malloc → dynamic memory allocation

• calloc ?

• using struct datatype we create

struct node

{ int data; }

node* next;

creates 2 slot like structure



}

node *nptr, *lptr, *list;

structure instance

Algorithm 4.2

somewhere in the
middle point
(meant)

next → mem location of next list
(slot)

list = NULL //empty list

(new) nptr →
(dynamic) tptr
(fixed) list

v1 | NULL

m0

m2

m1

m3

NULL

initial mem location

Searching a node?

(P30)

Algorithm 4.3 (inserting a node to a linked list)

nptr → data = item

→ pointing to
data slot of
node structure's
new instance

→ no name?
only mem location

sets data's
value as
input variable
item

nptr = new (node)

stores memory
location of
new instance
of node structure

Md

→ until Linked list

corner case → not address in the original code

#adding node at the header/startng/middle
location

if ($l\text{st} \rightarrow \text{data} > n\text{ptr} \rightarrow \text{data}$)

{
 $n\text{ptr} \rightarrow \text{next} = l\text{st};$
 $l\text{st} = n\text{ptr};$
}

else { previous code algorithm }

• Linked list can be used in any algorithm

→ adding header mem location to $n\text{ptr} \rightarrow \text{next}$
→ replacing $l\text{st}$ mem location with $n\text{ptr}$'s
→ now $n\text{ptr}$ → new header
→ original $l\text{st}$ only retains its memory location

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 \downarrow $\text{Next}=\text{NULL}$)

if (tptr->next==NULL)

{

 tptr->next=nptr;

*you will simply want to make a new node but do it
 by (first) having (1, 2, 3) string and
 then adding 4 to the end of it*

else {

} *old code*

$(x-s)(x-p)(x-r)(x-q)$

$= (x-s)(x-p)(x-r)(x-q)$

$O = s + p + r + q$

$b = -O$

$c = \frac{b^2}{4}$

$a = \frac{1}{4}$

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 = list;

while ($\text{tptr} \rightarrow \text{data} \neq \text{item}$)

{ pptr = tptr;

tptr = tptr \rightarrow next; }

(deleting the node)

$\text{pptr} \rightarrow \text{next} = \text{tptr} \rightarrow \text{next};$

~~next~~ delete (tptr);

for ① cage

1-121.

~~lptr = list;~~

if (~~list~~ → data == item) {

list = tptrn \rightarrow next;

Delete(tptr);

3

→ replacing header
with next-first
node

(att)

$lst = lst \rightarrow nest$

delete(lst)

\uparrow (math = false) delete($\mathbf{a[3]}$)

~~deletes~~
the 1st node

\ frees the
memory location
(but in function
in some languages)
(C++)

Do NOT
Do it

for (3) case

while { }

```
    while {  
        if (tptr->next == NULL)
```

{ pptr → next = NULL; }

```
{ pptr -> item =  
    delete (tptr); }
```

making pptr/
previous pointer
null

Sorting in Linked list?

P4.3

Bubble Sort

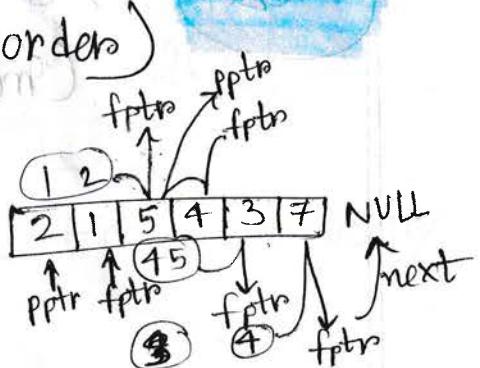
algorithm 4.5: (Sorting in ascending orders)

- $fptr \rightarrow$ latter pointer of $pptr$

$fptr = pptr \rightarrow next;$
while ($fptr \neq \text{NULL}$)

{ if ($pptr \rightarrow data > fptr \rightarrow data$)

{ interchange ($pptr \rightarrow data, fptr \rightarrow data$);
variable swapping



Not a
built-in
function

algorithm
notation
for swapping

$fptr = fptr \rightarrow next;$
} // end of while of step-5 \rightarrow if breaks out
from loop

$pptr = pptr \rightarrow next;$
} // end of while of step-4

Q. Sorting in descending orders?

Next

doubly linked list

Circular linked list

(left before mid)

E.Aq

Sorting Algorithm: ① insertion
② selection, -

- ① insertion
 - ② selection } $O(n^2)$
 - ③ bubble }

② selection } $O(n^2)$

③ bubble

- multiple loop means polynomial time complexity

1

assuming the first index sorted



→ unsorted

sorted
"subarray"

comparing with



Strong
temp var.

- comparing unsorted side with the whole sorted "subarray"

- Comparing side skipping 1st step
if($\text{arr}[i] > \text{arr}[j]$) { swap }
 $i < n, i++$ } int key = arr[i-1];

```

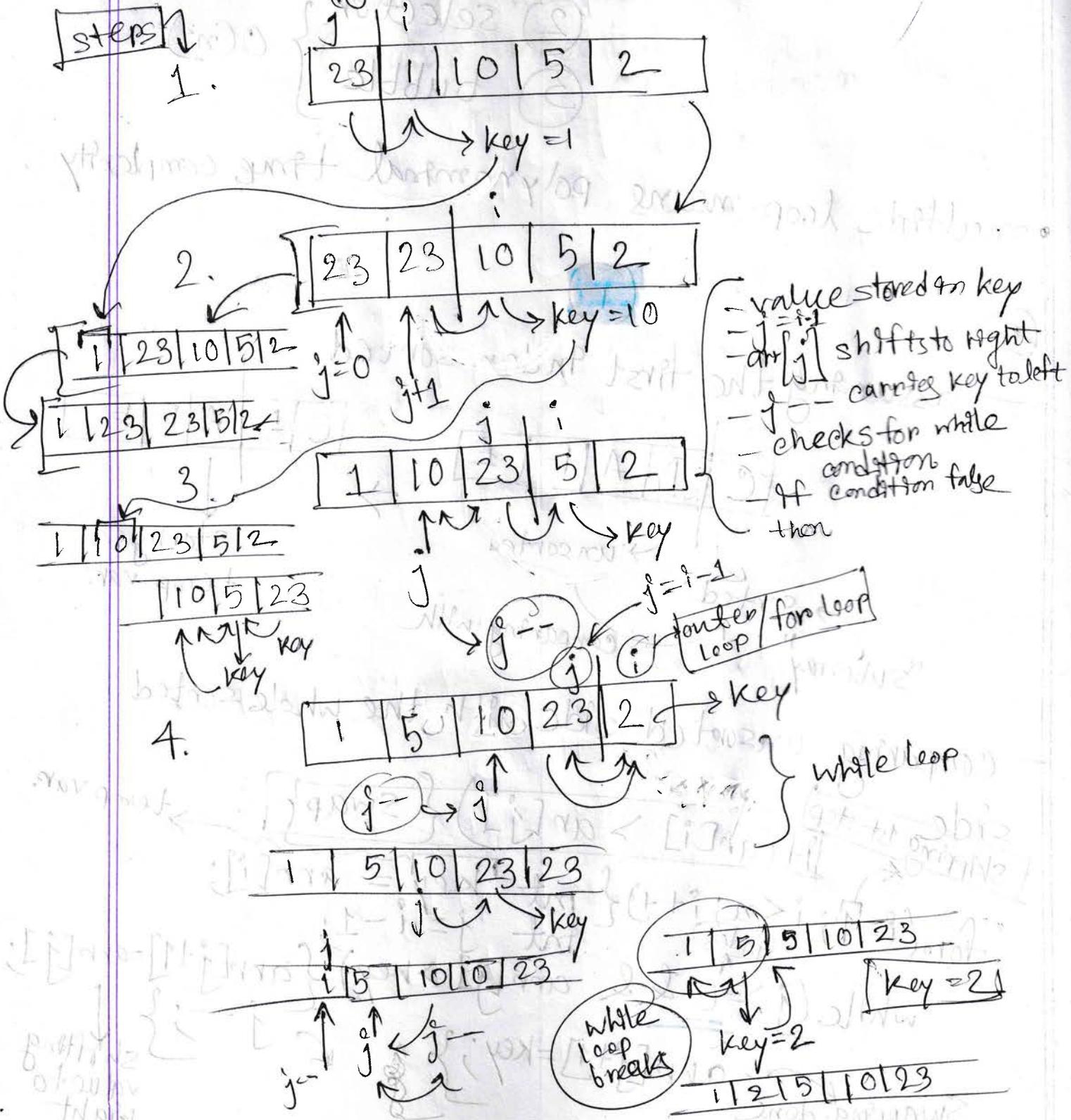
for(i=1; i<n; i++) {
    if(arr[i] < key) {
        arr[i] = arr[i+1];
        i++;
    }
}

```

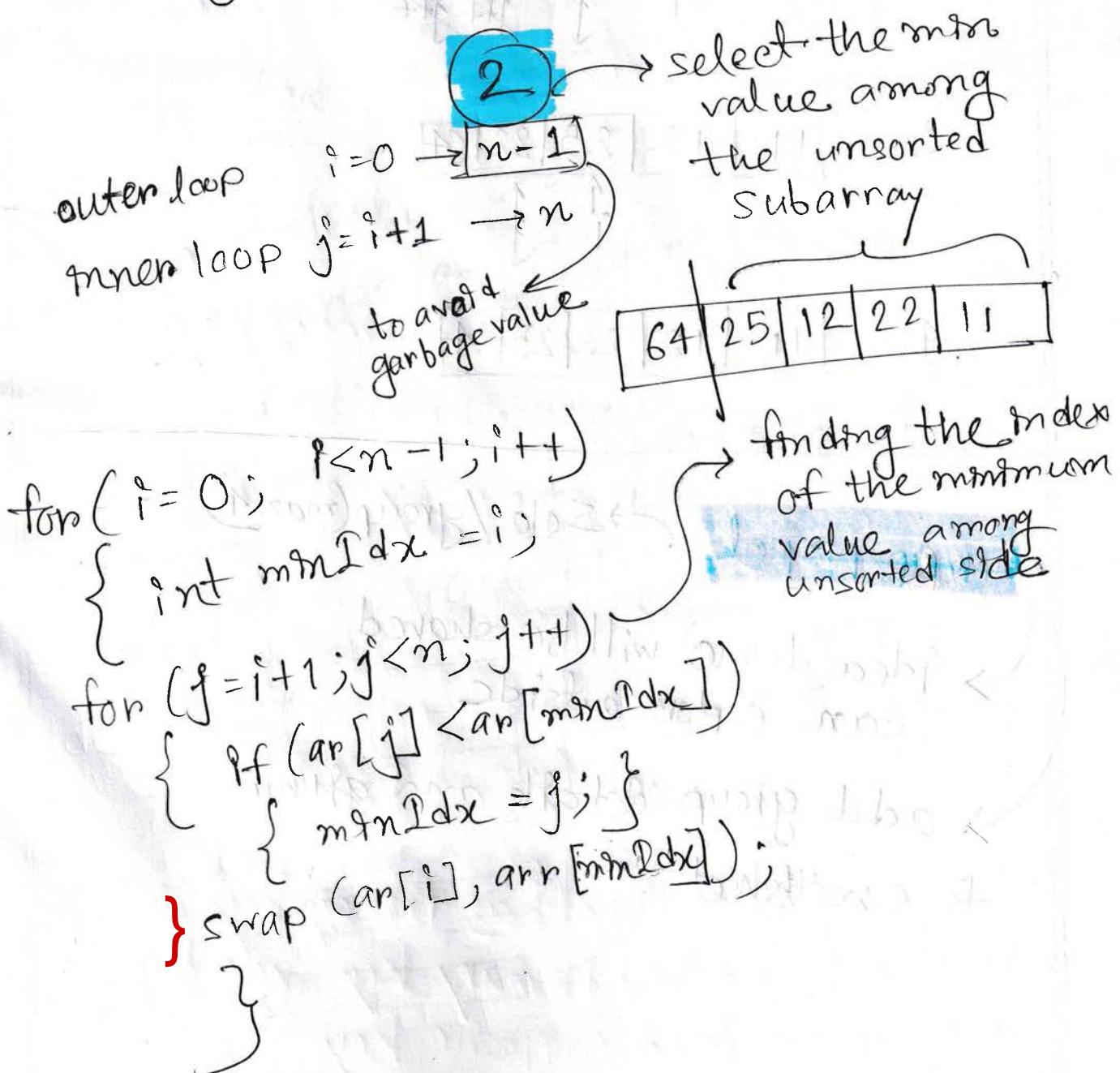
```
for(i=1; i<n; i++) { int j = i-1;
```

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 ($j \geq 0$) is false so $ar[j+1] = ar[0] = \text{key}$
 meaning array element is kept where it was.



minIdx changes

1.

64	25	12	22	11
----	----	----	----	----

 swap(
arr[i], arr[j])
 $i=0$ j $j++$ j after finding final
min value
2.

11	25	12	22	64
----	----	----	----	----

 i j $j+1$ $j+1$
3.

11	12	25	22	64
----	----	----	----	----

 i j
4.

11	12	22	25	64
----	----	----	----	----

→ Solo / group (max 4)
• **Group project**

→ Idea (demo) will be shared
can explore outside

→ add group details and github
to excel sheet

- 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^{th} index shifts to left after each cycle

```

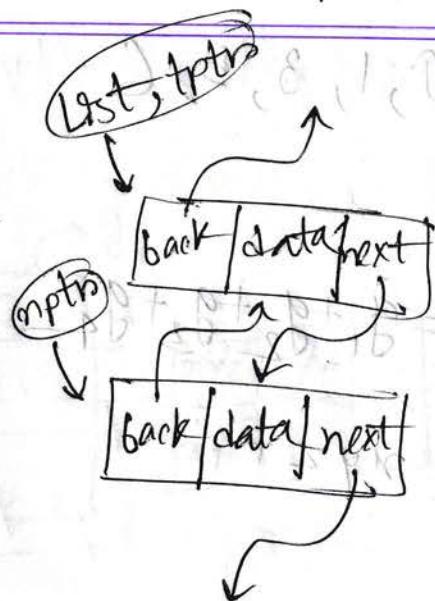
    if (arr[j] > arr[j + 1]) {
        swap(arr[j], arr[j + 1]);
    }
}
    
```

-2 | 4 | 5 | 0 | 1 | 1 | -9 | run for $n - 1$ steps

swap?
yes $i = 0$ step $i = 1$ step
 $\begin{array}{|c|c|c|c|c|c|c|} \hline -2 & 4 & 5 & 0 & 1 & 1 & -9 \\ \hline \end{array}$ $\begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline \end{array}$
 $\begin{array}{|c|c|c|c|c|c|c|} \hline 9 & 5 & -2 & 0 & 1 & 1 & -9 \\ \hline \end{array}$
no $\begin{array}{|c|c|c|c|c|c|c|} \hline 9 & 5 & -2 & 0 & 1 & 1 & -9 \\ \hline \end{array}$
no $\begin{array}{|c|c|c|c|c|c|c|} \hline 4 & 5 & -2 & 0 & 1 & 1 & -9 \\ \hline \end{array}$
yes $\begin{array}{|c|c|c|c|c|c|c|} \hline 4 & 5 & -2 & 0 & 1 & 1 & -9 \\ \hline \end{array}$
 $\begin{array}{|c|c|c|c|c|c|c|} \hline 4 & 5 & -2 & 0 & -9 & 1 & 1 \\ \hline \end{array}$

Doubly Linked List:

- headers and trailers
- ↓
- 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)

CL.8.1

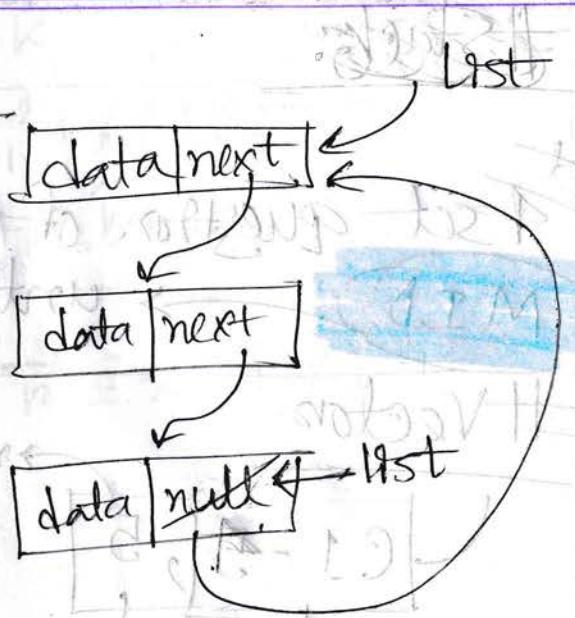
HOME

Circular Linked List:

algorithm 4.11:

```

l1st = NULL;
nptr = new (node);
nptr->data = item;
nptr->next = NULL;
if (l1st == NULL)
    l1st = nptr;
    nptr->next = l1st;
    tptr = nptr;
else {
    tptr->next = nptr;
    nptr->next = l1st; // differs from
    tptr = nptr;
}
    
```



Md → Schaum → introductory theory
Book highlighted parts.

Q. Array vs linked list (P61)

Stack: LIFO = Last in First out

push pop

- push operation : algorithm 5.1

- pop operation : algorithm 5.2

→ "underflow" or "stack is empty"

→ where 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 arithmetic expression (using parenthesis with pop and push)

→ calculator's "Syntax Error"

→ after "push" and "pop" operation, a stack

- 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

M

- "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)

$$+AB = +1 2 = 3$$

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

$$\begin{aligned}
 & + - (1 * 2) 3 4 = + - (1 * 2) 3 4 \\
 & + (-1) 4 = -1 + 4 \\
 & -1 + 4 = 3
 \end{aligned}$$

Sign and operators are different
 $(-)$

DSA-1

09.04.25

Conversion to post fix 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 than
 - 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

(calculator type)

#Queue: can be implemented using array and linkedlist

- push = enqueue (add)

- pop = dequeue (remove/delete)

- here FIFO is used

if ($\text{front} == 0$) // then we can push in the queue

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

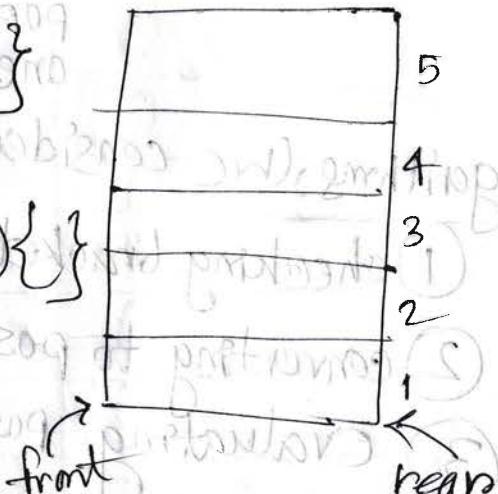
else if

else { $\text{front} = \text{rear} + 1$;

else { $\text{rear}++$;

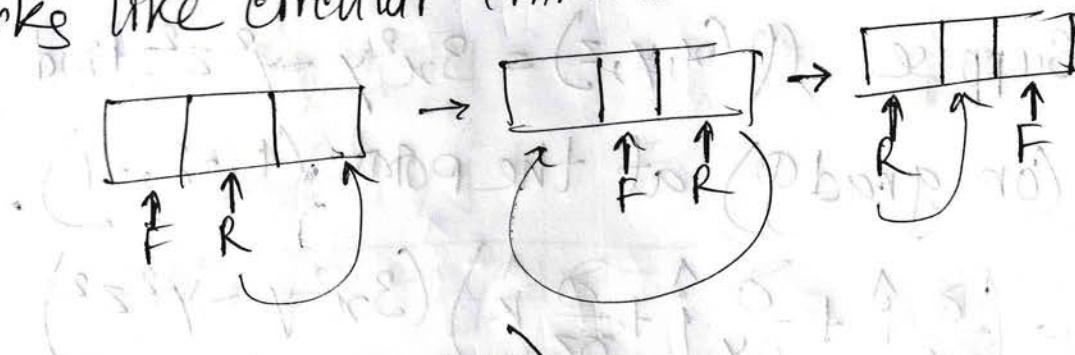
$\hookrightarrow \text{queue}[\text{rear}] = \text{element}$

$\text{rear}++$



- queues usually taught as an infinite array (not circular)

- works like circular linked list



Algorithm : (for enqueue)

```

if (front = 1 && rear = N or front = rear + 1)
print (overflow ; return)
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 $\text{front} = 0$, queue is empty
or "underflow"

Q. State and Dynamic Programming?
<last>

list

Q. type?

Enqueue = addition

Dequeue: (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

AS-FO-21

Q. do 1-A2a

#Priority Queue:

① Ascending order priority queue

② Descending order priority queue

lower priority number
and higher priority numbers
in the priority

language?

Q. (type)

→ Q. addition, deletion

Q. priority queue.

must show
step by step

Mid solve?

#Stack, LIFO

stack is a container
linear data structure
can be implemented using primitive data structure
push(), pop(), peek() / top(), empty()
[checking highest value]

If $\text{top} = -1$, 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:
 $\text{top} = \text{size} - 1; // \text{stack is full}$
- for each pop():


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

→ basic and ends with ↓ → ~~functions~~

(structure and class are similar)

using class for data structures

• 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

I **Ctrl+shift+↓** = copy above line in VScode

object creation:

Stack S;

S.push(10);

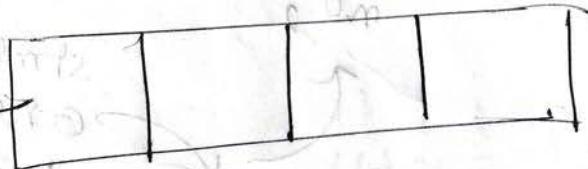
S.push(20);

Extracting:

#Queue:

FIFO

front



vars } front will update once, points to the front index
back } back/rear updates per operation

methods: enqueue()

dequeue()

peek() / front()

isEmpty()

display()

$\{ \text{front} = -1; \}$ } queue is empty
 $\{ \text{back} = -1; \}$

when $\text{enqueue}()$ used:

```

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

```

check overflow:

```

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

```

when $\text{dequeue}()$ 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: $\text{arr}[front]$

if queue is empty: same as stack

check if queue is empty: unlike stack

Output: front 0 - (n-1)

- ২৪ মে ২০২৩
- variable naming :
 - (1) ISEmpty → camel 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 returning true value

Lab Project

→ Details will be given at classroom

[6] 7/9

186
X.Home & 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 "on" 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]

classmate

LA2A

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 complexity?

ISPO-88

F. D. HAZA

Input array $A[1 \dots n]$
 → means array elements
 we can write $A[n]$ 1st to nth
 and not indexes as convention

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

for ($i=1; i \leq n; i++$) {
 moves changes
 small_index = i ; }
 " > " for
 descending order

only index changing } for ($j=i+1; j \leq n; j++$) {
 finding the
 smallest
 element
 of the
 unsorted
 elements

if ($A[j] < A[\text{small index}]$)
 { $\text{small index} = j;$ }

temp = $A[i];$
 $A[i] = A[\text{small index}];$ } swapping the smallest
 $A[\text{small index}] = \text{temp};$ } of unsorted with the
 first element.

Types of Queue: ① Simple Queue

② Circular Queue

③ Priority Queue

④ Dequeue (Double ended queue)

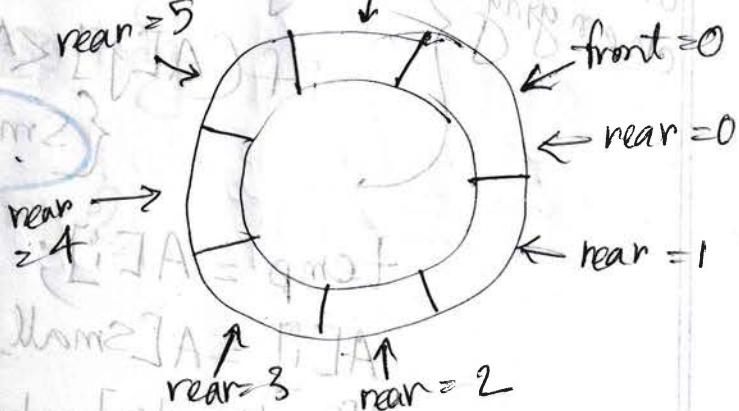
①

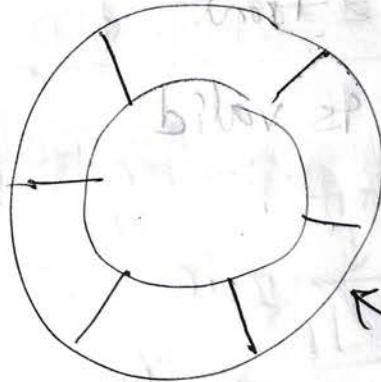
"dequeue" only shifts the index instead of actually deleting them. We cannot "deallocate" the memory.

②

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

- front changes only once.





- Modular theorem: to check if input is valid

$$(rear + 1) \% N$$

value returns in a circular manner.
value of the empty index

In circular queues,

$$\boxed{\text{rear} = (\text{rear} + 1) \% N}$$

when, $\text{rear} = 6$ for $N = 7$

$$(6 + 1) \% 7 = 7 \% 7 = 0 = \cancel{\text{front}} \text{ rear}$$

} {
if ($\text{front} \geq 0$ & $\text{rear} = \text{SIZE} - 1$)
if ($\text{front} = (\text{rear} + 1) \% \text{SIZE}$)

Why not
else if?

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

overflow logic: when $\text{rear} == \text{front}$

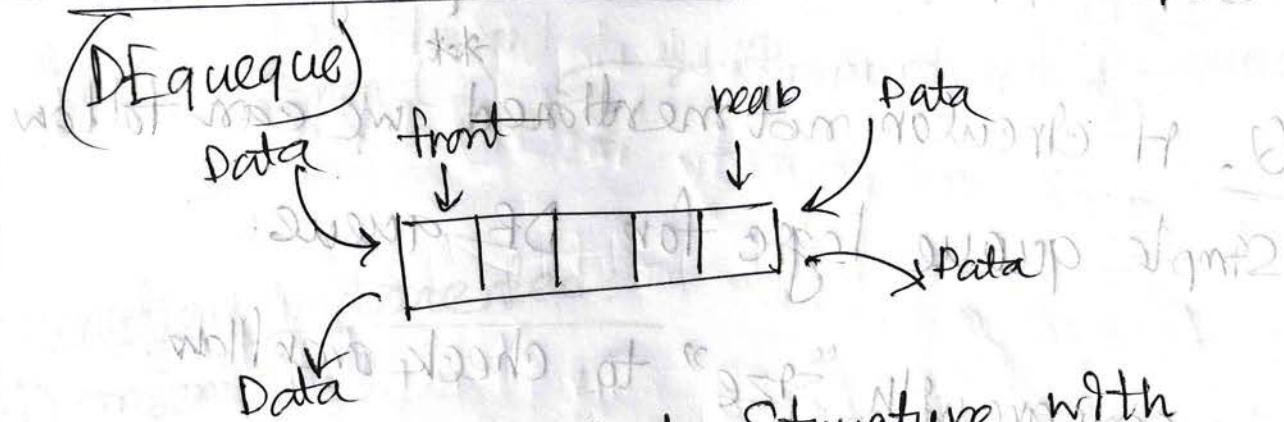
no further enqueue is valid

new function: bool $\text{isFull}()$

- outputs correct
- false negative? false positive? \Rightarrow logic issue / corner case
- return only returns single time and breaks out of the function.
- if ($\text{front} == \text{rear}$)
 $\{ \text{front} = -1; \}$ // deQueue()
 $\{ \text{rear} = -1; \}$ 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 $\text{mEmpty}()$

(4)

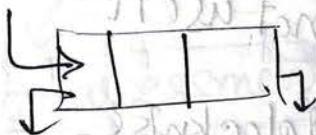
Double Ended Queue! Does not follow FIFO rule.



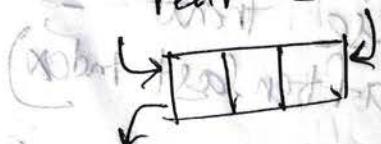
- It's a combined Data Structure with properties of both stack and Queue and follows the logic of Circular Queue.

Types:

① Input restricted
 rear restricted

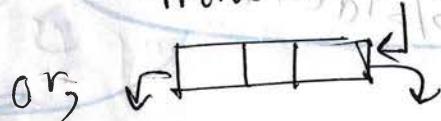


② Output Restricted
 rear restricted



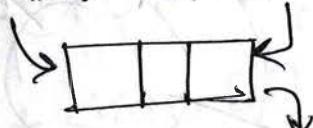
Deque:

front restricted



Deque:

front restricted



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

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

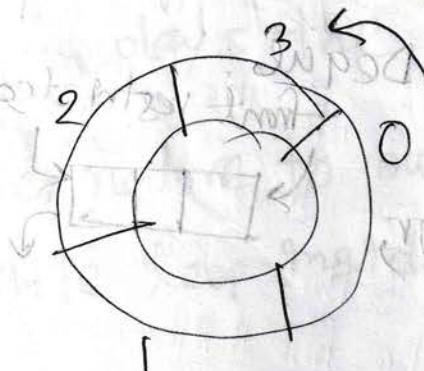
- we compare with "size" to check overflow.
if ($size == MAX$) then
Queue is full

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 "insert Front()"
- Property of DE queue

functions implemented:

- ① insertFront
- ② insertRear
- ③ deleteFront
- ④ deleteRear

HW

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

for assessment

code:

```

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

```

(3)

#Priority Queue:

- enqueue, dequeue used based on priority
- insert/ delete elements based on priority

① Ascending order

comparing current input value
with previous value

$$\text{if } (\text{ar}[i] < \text{ar}[i+1]) \\ \{ \text{ar}[i+1] = \text{ar}[i]; \}$$

constant time complexity

input + sort happening for insertion.

for deletion:

$i \leftarrow \text{front} \rightarrow \text{two} \quad \{ \text{front} = i, \text{two} = \text{MAX}(i+1) \}$

$\leftarrow \text{front} \rightarrow \text{two} \gg \text{two} \quad \{ \text{front} = i, \text{two} = \text{MAX}(i+1) \}$

Priority number?

value →	7	2	9	5
priority →	1	3	2	4

Question type

value + priority pairs used in
structures (node type data structure)

compare + insert → while inserting/
after inserting

DSA - 1 theory

Assignment: (3 codes)

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

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 2 (what we learned w/ data structure)

Idea:

(1) Management

(2) Game → Guessing game, multiple players, scores

(3) file system

• Read-write

• word completion

(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"

199

DSA-1

23.04.25

#③ Insertion Sorting:

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

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

for ($i=1$; $i \leq n$; $i++$)

{ $j=i$ }

while ($j > 0$ && $a[j-1] > a[j]$)

{ $temp = a[j]$ }

$a[j] = a[j-1]$ }

$a[j-1] = temp$ }

$j--$ }

}

→ " $<$ " for
descending
order

"different parts to sort when flood"
"different parts to sort when flood"

CS PA 25

L-A2D

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:

- ~~Quicksort~~ is algorithm of ~~cell~~ ~~sort~~ ~~array~~ worst case and not the most efficient

① Merge Sort

Ex. No. 8

(Q) [10 marks] - NB.M

- ① algorithm? (wasn't in the syllabus before)
- 25 operations
- Merge Sort Time complexity $O(n \log n)$

Q. Given the characteristics, identify the sorting algorithm?

1. Works to divide the input into subproblems of size $\frac{n}{2}$ each.

2. Merge subproblems.

3. No break or loop.

Algorithm 2:

This is merge sort.

Time complexity: $O(n \log n)$

$O = \text{Nb. of steps} \times \text{Time per step}$

$O = \frac{15}{4} \times 15 = 56.25$

$O < \text{Insertion}$

#Tree: Top-down arrangement of root-node

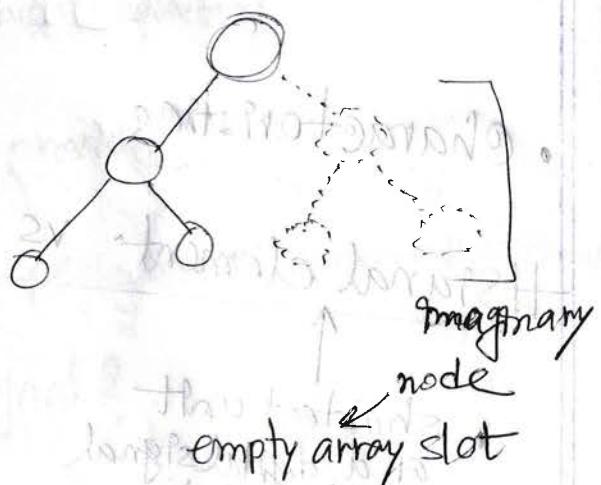
(unlike ~~actual~~ real life tree (bottom-up))

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

"grand parent" and "grandchild"

"Left child" and "right child"
 $2 \times n =$ $2 \times n + 1$

Data Structure:



level (K) = ∞

node per level = 2^n ← for binary tree

height?

$$\text{max no. of nodes} = 2^K - 1 = n$$

$$\begin{aligned} 2^0 &= 1, \text{ at } K=1 \\ 2^1 &= 2, \text{ at } K=2 \\ 2^2 &= 4, \text{ at } K=3 \end{aligned}$$

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

$$K = \log_2 32 = 5$$

Q. what if 33?

$$K = \log_2 34 = [5.08] = 6$$

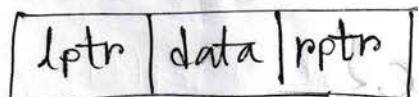
we use ceiling function

33 May

③ Data struc

for node n, level K,

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

Data structure!

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

datatype for
not a
int a;
int b;
a = &b;

- 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



Full binary tree:

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

Complete binary tree:

"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

- ECMid → 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
before 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.

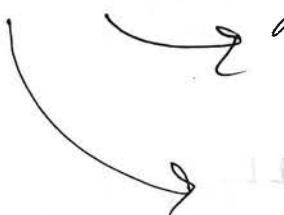
219

DSA# Lab 9

06.05.25

Qt →

Godot → C++



UN = base

base = shared by

genera

freeing.



Linked List Insert operation:

- insert at beginning:

- insert At Position: position does not mean index
Linked list does not rely on indices.

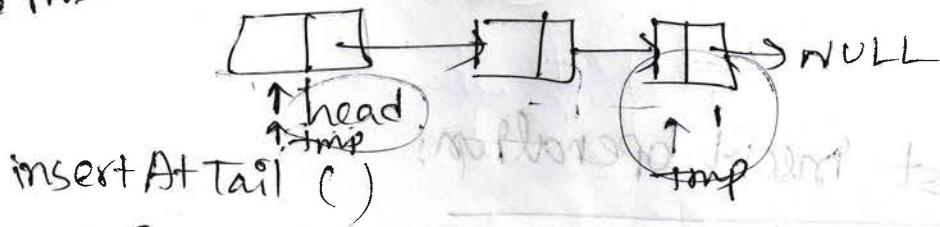
"new" keyword is used in dynamic memory allocation. We dynamically allocate memory when creating instances and object.

- for graph presentation we use "malloc"
- position

if any node inserted:

head == NULL
newNode = head

• insert



insert At Tail ()

{

Node $\ast \text{tmp} = \text{head}$

while ($\text{tmp} \rightarrow \text{next} \neq \text{NULL}$)

$\text{tmp} = \text{tmp} \rightarrow \text{next}$

$\text{tmp} = \text{tmp} \rightarrow \text{next}$

$\text{tmp} \rightarrow \text{next} = \text{newnode}$

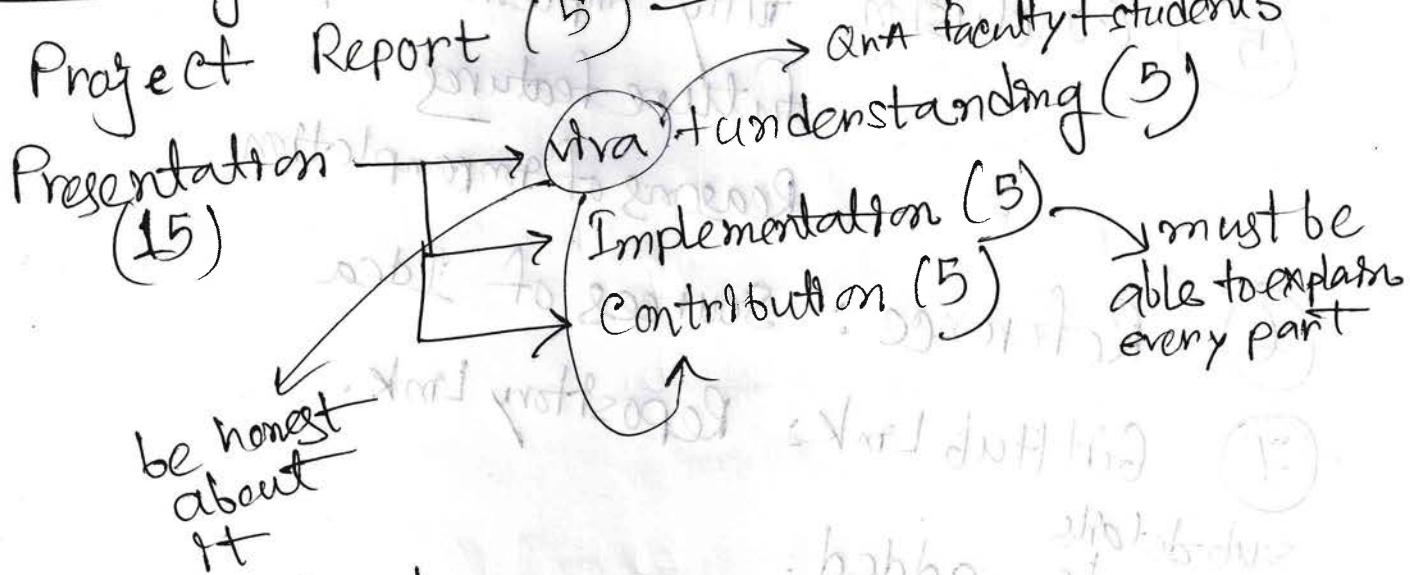
function for finding size of Linked List

Video submission review:

- be more confident
- prize for encouragement

Presentation (Project):

Scoring Rubrics:



Project Report

Cover Page:

Title: "final project is about..."

student names and ID

Section

Session

faculty information (named)
+ designation → both faculties

Submission Date

Booby

- ① Objective: 1-2 brief
 - 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 non completion
 - ⑥ Reference: Sources of Idea
 - ⑦ GitHub Link: Repository Link.
 - sub-details can be added.

Jay: ~~multihuser support; many games~~

2nd ad

Sarafit

Grodot

Q

I want to contribute

DO 2 friend

X IT.

ITAKU

Mithibra in Shabu

Stoßfunktionen, wenn ich mit reagiere,

(not - MoT)

mit dem am Flugzeug gehen auf ITAKU

Comment? = workA T12

zurück zu mithibra MoT ITAKU

zusammenführen? = workB T12

Wiederholung, Formular, Formular ITAKU P. E. T. O. U. I. R.

trifft sich zu einem

work A MoT MoT

zurück zu mithibra

zurück zu mithibra

work B MoT

work C MoT