Exam-2021

PART-A

1 (a)

By: Mohan Kumar

Binary search has the following benefits over linear search.

1. Efficiency: Binary search is faster than linear search, especially for larger lists.

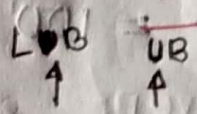
2. Reduced comparisons: Binary search requires fewer comparisons to find the target element, because it narrows down the search space by half in each step.

3. Divide and conquer approach: Binary search uses a divide and conquer strategy by repeatedly dividing the search space, leading to faster convergence.

4. Sorted list requirement: It needs a sorted list as input, but if the

list is already sorted or can be sorted once, binary search provides significant time savings.

1 (b)



Given: Maze(1:6, -4:1, 5:10)

Base = 100

w = 4

Address of Maze[3, -2, 8] :

~~in row maj~~ $L_1 = 6 - 1 + 1 = 6$

$L_2 = 1 - (-4) + 1 = 6$

$L_3 = 10 - 5 + 1 = 6$

$E_1 = 3 - 1 = 2$

$E_2 = -2 + 4 = 2$

$E_3 = 8 - 5 = +3$

For Row major:

Base + W [(E1 L2 + E2) L3 + E3]

= 100 + 4 [(2 × 6 + 2) 6 + 3]

= ~~100 + 4 [(18+2) 6 − 3]~~ = 100 + 4 [84 + 3]

= ~~296~~ = ~~4848~~ 448

For column major:

Base + W [(E3 L2 + E2) L1 + E1]

= 100 + 4 [(3 × 6 + 2) 6 + 2]

= ~~100 + 4 (16 × 6 + 2)~~

= ~~296~~ ~~276~~ 488

For Row major:

Base + W [(E1 L2 + E2) L3 + E3]

= 100 + 4 [(2 × 6 + 2) 6 + 3]

= 448

For Column major:

Base + W [(E3 L2 + E2) L1 + E1

= 100 + 4 [(3 × 6 + 2) 6 + 2]

= 488

1(c)

Matrices with a high propotion of zero entries are called Sparse Matrices. we often store sparse matrix in a 1D array to save spaces. Here some reason is as follows:

1. we can save almost half the memory requirement of a n square sparse matrix A

2. So, we can use an one dimensional array (say B). That is, we let $B[1] = a_{1,1}$, $B[2] = a_{2,1}$ and so on.

3. In this way the one dimentional array will contain : $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$ elements rather than n^2.

4. we need a formula that gives us the integer L in terms of j and k where $B[L] = a_{j,k}$.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{pmatrix}$$

Triangular matrix

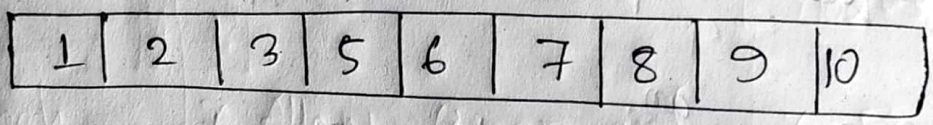
2D

$$\begin{pmatrix} 4 & -3 & 0 & 0 & 0 \\ -2 & 3 & 6 & 0 & 0 \\ 0 & 4 & -5 & 7 & 0 \\ 0 & 0 & -2 & -5 & 5 \\ 0 & 0 & 0 & 8 & 9 \end{pmatrix}$$

Tridiagonal Matrix

2D

convert

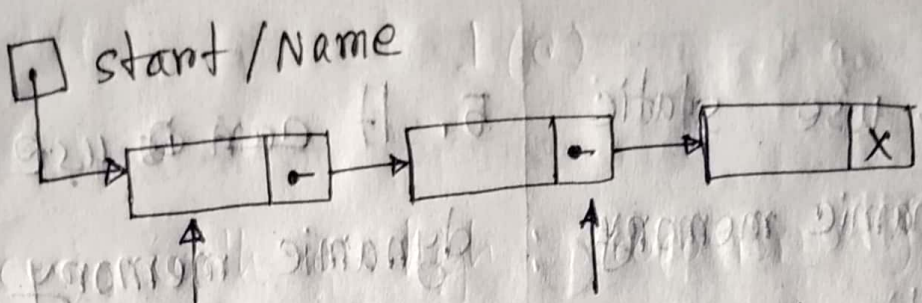


represent in 1D

2 (a)

Linked list:

A linked list, or one way list, is a linear collection of data elements called nodes, where the linear order is given by means of Pointers.



here the main differences between a linked list and linear array:

	Linear array	Linked list
Structure	1. Arrays are configuous blocks of memory	2. Linked list consists of nodes
Inserction & Deletion	2. Arrays can be costly for inserction and deletion.	2. It use dynamic memory so, perform these operations more efficiently
Size	3. Arrays have a fixed size	3. It can dynamically grow or shrink.
Random access	4. It allows for direct access to elements	4. It require traversal from the be gining

Memory allocation	5. Arrays use static or dynamic memory allocation	5. It can use dynamic memory allocation through pointers
Iteration	6. Arrays allow for faster iteration using index	6. It requires traversal through nodes
Search technique	7. Allow Linear and Binary search	7. Binary search not allow.

2 (b)

Garbage collection:

The technique to collect the unused space is called Garbage collection.

When a node is deleted from a list, the deleted node should be put onto a list of

available space or free storage list or free pool so that another can use that. The OS often periodically collect all the deleted space onto free-storage list.

Overflow:

Data insertion to a data structure in situation when there exist no available space to the free-storage list (AVAIL = NULL) generates a error. This error is called Overflow.

The programmers may handle overflow by printing the message OVERFLOW.

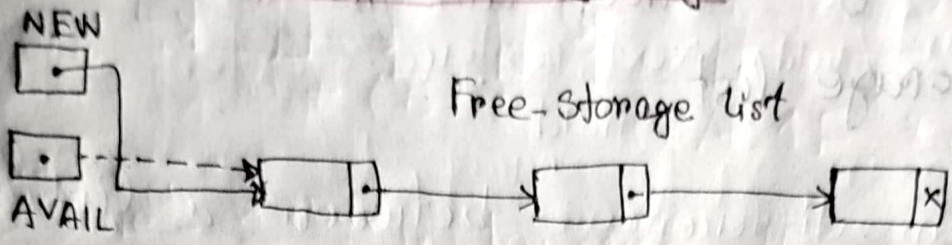
Underflow:

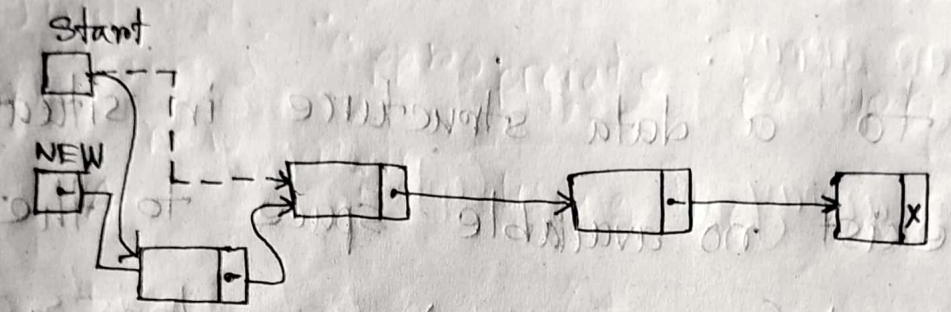
Data deletion from a data structure in situation when there is no elements to the data structure (START = NULL) generates a error, this error is called Underflow.

The programmer may handle underflow by printing the message UNDERFLOW.

2 (c)

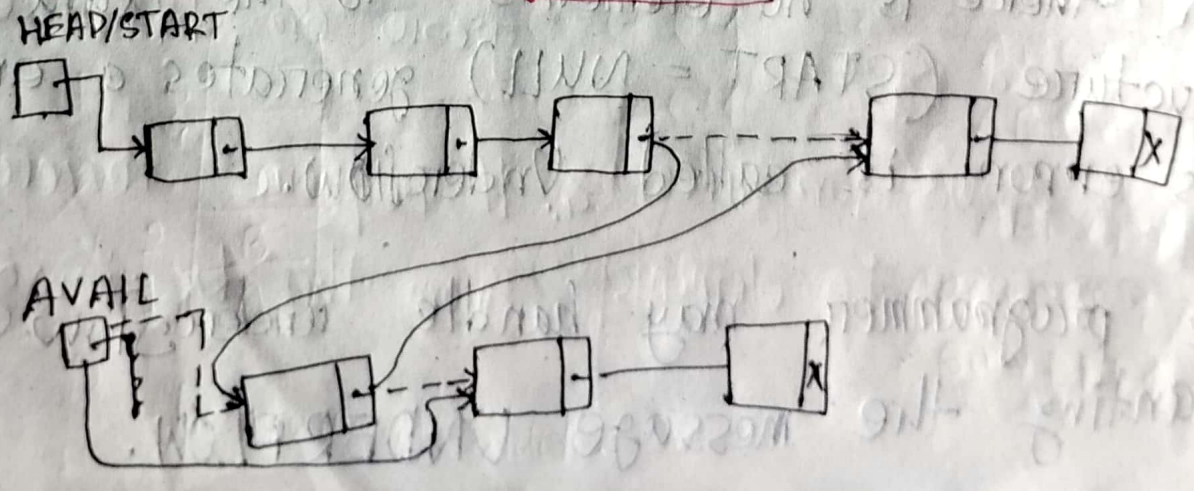
Insert data at the begining:





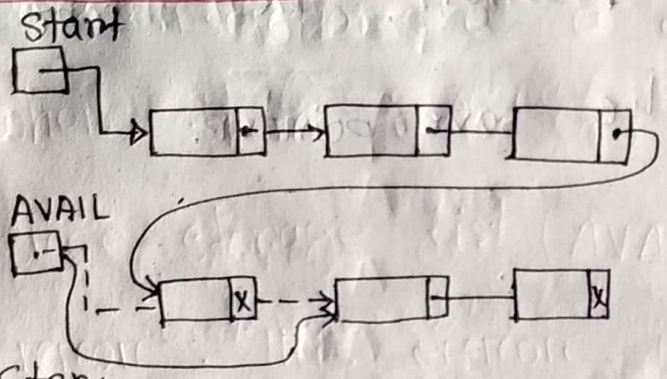
At first create a new node and assign the data that we want to insert. Then make the new node point to the current head of the linked list. Update the head pointer to point to the new node.

Insert data after a given node:



1. At first create a new node with desired data. Then
2. Find the given node in the linked list.
3. Set the new node's next pointer, point to the next node of the given node.
4. Finally, update the 'next' pointer of the given node to point to the new node.

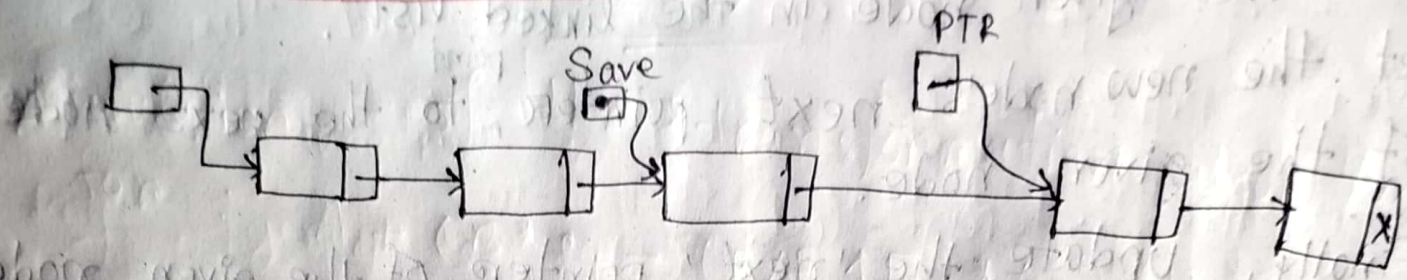
Inserting at the end:



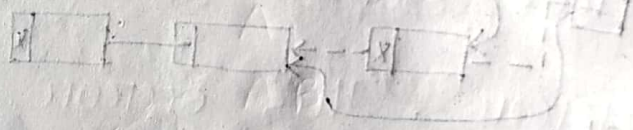
Step:

1. Create a new node with desired data.
2. If the linked list is empty, make it head and finish.
3. Traverse until the end of the linked list,
4. Update the next pointer of the last node to point to the new node.
5. Set the new node as the new last node by updating its next pointer to NULL.

Inserting into a sorted list:



Create a node, traverse the linked list to find its appropriate position, then insert the new node using the save pointer.



3 (a)

Stacks: A stack is a linear list of elements in which an element may be inserted or deleted only at one end. ~~called the~~ It is also called LIFO - Last in Fast out.

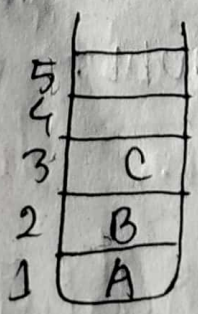


Diagram of stack.

the usage of stack in recursive algorithm implementation:

i) Stack tracks function calls and execution contexts in recursive algorithms.

ii) Stores parameters, return addresses and local variables

iii) Remembers the state of recursive calls and returns to the correct point.

iv) Handles multiple recursive calls, backtracks and determines termination.

v) Maintains operation order and handles nested calls effectively.

B) b)

3(b)

Simulation of the transformation algorithm for : A + (B * C - (D / E ↑ F) * G) * H)

Ans: ⊛ Slide - 2 [infix-postfix-Queue 2nd]

Scanned	Stack	Expression
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
↑	(+(-(/↑	ABC*DE
F	(+(-(/↑	ABC*DEF
)	(+(-	ABC*DEF↑/
*	(+(-*	ABC*DEF↑/
G	(+(-*	ABC*DEF↑/G

) C+ ABC*DEF↑/G*-

* C+* ABC*DEF↑/G*-

H C+* ABC*DEF↑/G*-H

) C ABC*DEF↑/G*-H*+

3(c)

INSERT (queue, n, front, rear, item)

1. If front = 1 and rear = n, or if front = rear+1, then, write Overflow and return.
2. If front = NULL, then set front := 1 and rear := 1
3. Else if rear := n, then Set rear := 1.
4. Else rear := rear+1
5. Set queue [rear] := item.
6. Return.

4(a)

A hash function is function which given a key, generates an address in the table. Hash function are used to speed up searching. example: a book call number.

Hash function has the following properties:

i) H always returns a number for an object

ii) two equal objects will always have the same number

iii) two unequal objects not always have different numbers.

4(b)

Given, the table length = 10

Data: 14, 12, 18, 13, 2, 3, 23, 5, 15

Here, K % 10 → hash function

14 % 10 = 4

12 % 10 = 2

18 % 10 = 8

13 % 10 = 3

2 % 10 = 2

3 % 10 = 3

23 % 10 = 3

5 % 10 = 5

15 % 10 = 5

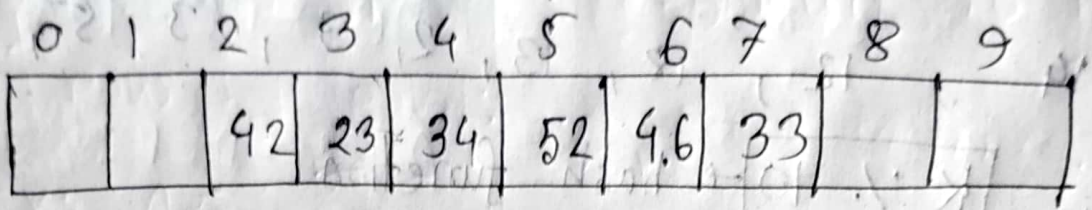
From this hash value we can see, collision is arising so, we can resolve it by plus three probing.

0	1	2	3	4	5	6	7	8	9
	5	12	13	14	2	3	15	18	23

using plus three probing

4(c)

Give the hash table:



for,

A) 46, 42, 34, 52, 23, 33

here,

46 % 10 = 6

42 % 10 = 2

34 % 10 = 4

52 % 10 = 2 → rehash → 3

from the table 52 is not place its apporoprite position, it is not right key values order.

B) 34, 42, 23, 52, 33, 46

34 % 10 = 4

42 % 10 = 2

23 % 10 = 3

52 % 10 = 2 → rehash → 5

33 % 10 = 3 → rehash → 6

it also wrong order.

46, 34, 42, 23, 52, 33

46 % 10 = 6

34 % 10 = 4

42 % 10 = 2

23 % 10 = 3

52 % 10 = 2 → rehash → 5

33 % 10 = 3 → rehash → 7

o, we can see that order is appropriate.

2020-

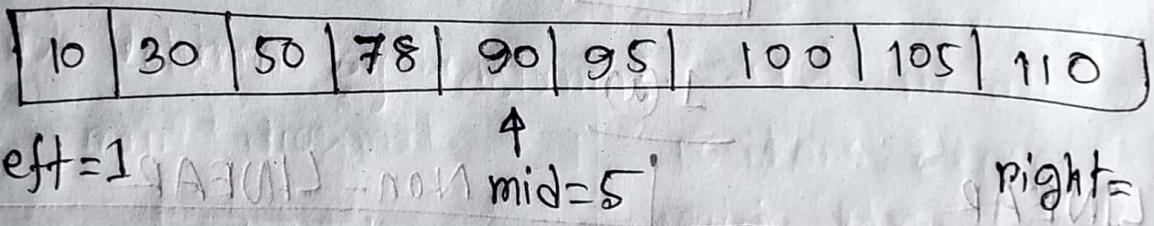
1(a)

LINEAR	Non-LINEAR
1) The data items are arranged in a sequential manner, where the elements are attached adjacently.	2) The data elements are attached in hierarchically manner.
2) Memory utilization is inefficient	2) Memory utilization is efficient

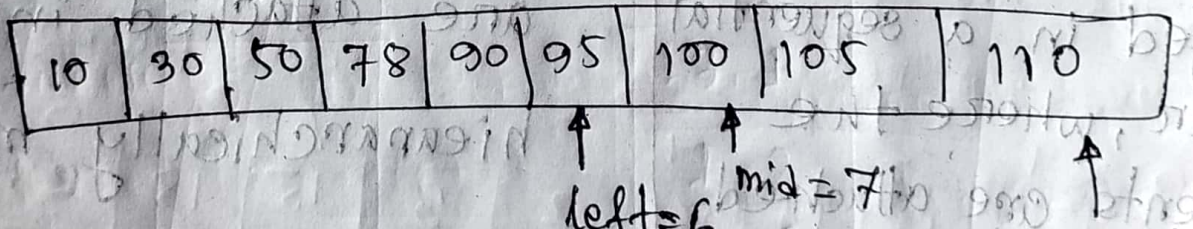
3) Single-level	3) Multi-level
4) Easier to implement	4) Difficult to implement
5) Ex: Array, linked list, stack, queue.	5) Tree, graph.

1(b)

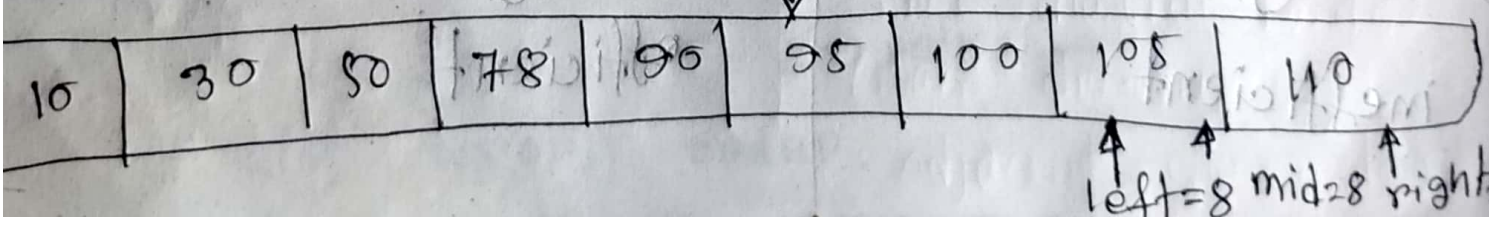
Here, search item = 110

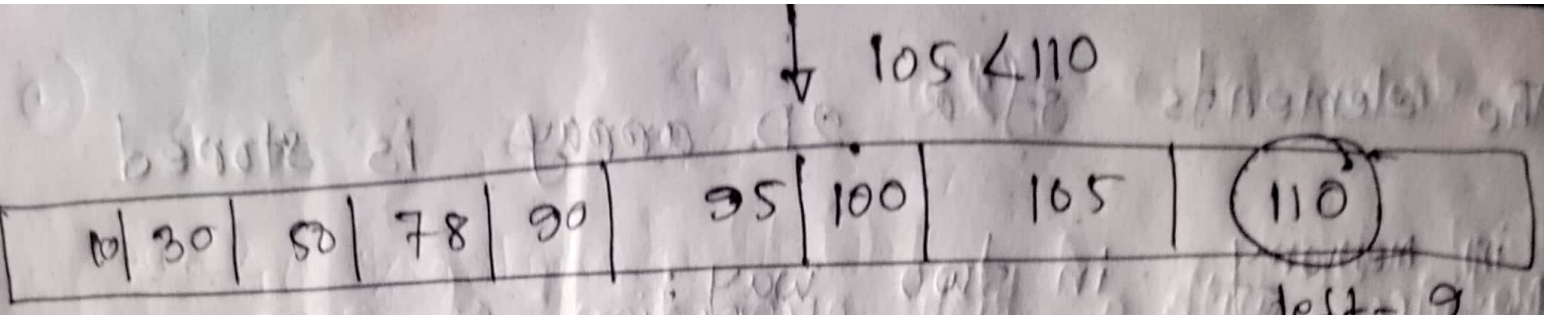


90 < 110



100 < 110





Data[mid] = item

Search complete with: successful.

1 (c)

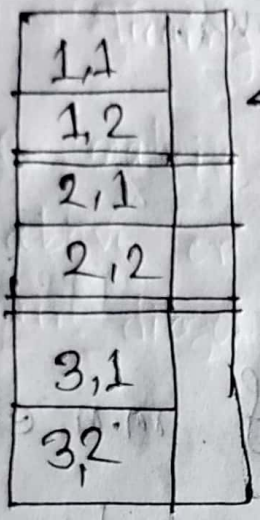
2D Array: It is a collection of m.n elements, where each element is specified by a pair of integers (such as j, k) called the subscripts.

The elements of a 2D array with m rows and n columns, looks like as below

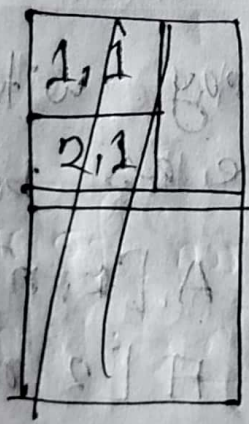
A[1,1]	A[1,2]	...	A[1,n]
A[2,1]	A[2,2]	...	A[2,n]
...	...	...	...
A[m,1]	A[m,2]	...	A[m,n]

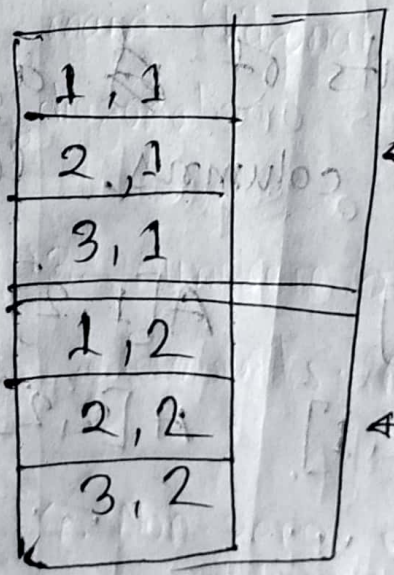
The elements of a 2D array is stored in memory in two way:

i) row by row:



ii) Column by column:





2(a)

Here,
Size $m = 20$
$n = 5$
Base = 200
$w = 4$

Address Score [10, 2]:

In row major:

$$\text{Base} + w[n(i-1) + (j-1)]$$

$$= 200 + 4[5(10-1) + (2-1)]$$

$$= 200 + 4 \times 46$$

$$= 384$$

In column major:

$$\text{Base} + w[m(j-1) + (i-1)]$$

$$= 200 + 4[20(2-1) + (10-1)]$$

$$= 200 + 116$$

$$= 316$$

2(b)

Matrices with a high proportion of zero entries are called Sparse Matrices.

Traingular	Tridiagonal
1. Square matrix with zeros above or below the main diagonal	1. Square matrix with non-zero entries only on the main diagonal and adjacent ~~media~~ above or blow.
2. zeros above or below the main diagonal	2. Zeros in all other positions except the main diagonal and adjacent diagonals
3. Number of non zero elements: $\frac{n(n+1)}{2}$	3. number of non zero elements: $(n-2)*3+4$
4. Forms a triangular shape	4. Non-zero entries form three diagonal bands (main diagonal and adjacent diagonals)

Example:

$$\begin{bmatrix} a & b & c \\ 0 & d & f \\ 0 & 0 & e \end{bmatrix}$$

5. example:

$$\begin{bmatrix} a & e & 0 & 0 \\ f & b & g & 0 \\ 0 & h & c & i \\ 0 & 0 & j & d \end{bmatrix}$$

2(c)

We can save almost half the memory requirement of a n square sparse matrix A if we use an one dimensional array B.

In this way B will contain,

$B[1] = a_{1,1}$

$B[2] = a_{2,1}$

$B[3] = a_{2,2}$

$B[k] = a_{j,k}$

$B\left[\frac{n(n+1)}{2}\right] = a_{n,n}$

let, we can insert $a_{j,k}$ in 1D array:

For triangular matrices:

location: $L = \emptyset \quad \frac{j(j-1)}{2} + k$

For tridiagonal matrices:

~~$L = 2j + k \quad 2i + j - 2$~~

location: $L = 2j + k - 2$

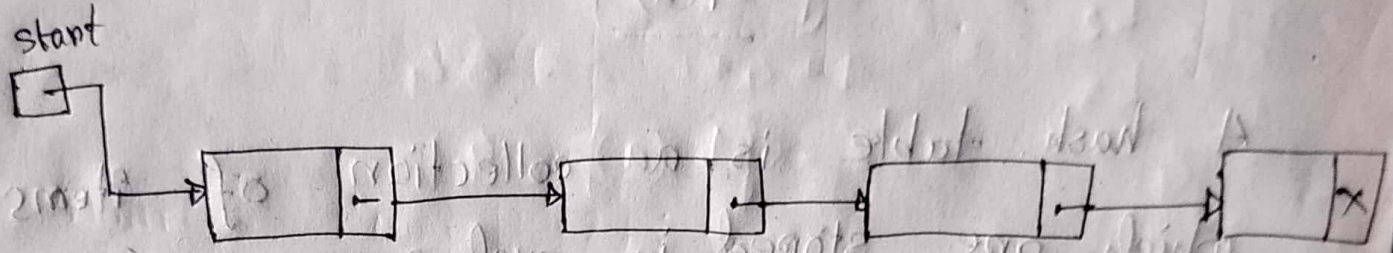
3 J a

same as 2021 : 2,a

3 (b)

The binary search algorithm cannot be directly used on data stored in a linked list because it requires random access to elements, while linked lists only provide sequential access.

Binary search divides the search space in half but accessing the middle element in a linked list would require traversing the list from the beginning, resulting in linear time complexity.



To search for an element in a linked list, linear search is more suitable, where we iterate through the list sequentially, until the desired element is found or end of the list.

So we can conclude, we can not employ Binary search on linked list as is done on array because there is no way of indexing the middle element of the list directly.

3(c)

same as: 2021 - 2(c)

4(a)

A hash table is a collection of items which are stored in such a way as to make it easy to find them later.

When two items have a same hash value then the situation is called collision.

One way to avoid collision is to increase the size of the hash table so that each possible value in the item range can be accommodated. But it is not practical for a large data set.

There are some ways to extend the simple remainder method:

i) Folding method

ii) Mid square method

iii) character based method.

3 (b)

Here, table size = 9

Hash function = "K % 9"

Data: 5, 29, 20, 0, 27, 26, 17, 18

Now,

5 % 9 = 5

29 % 9 = 2

20 % 9 = 2

0 % 9 = 0

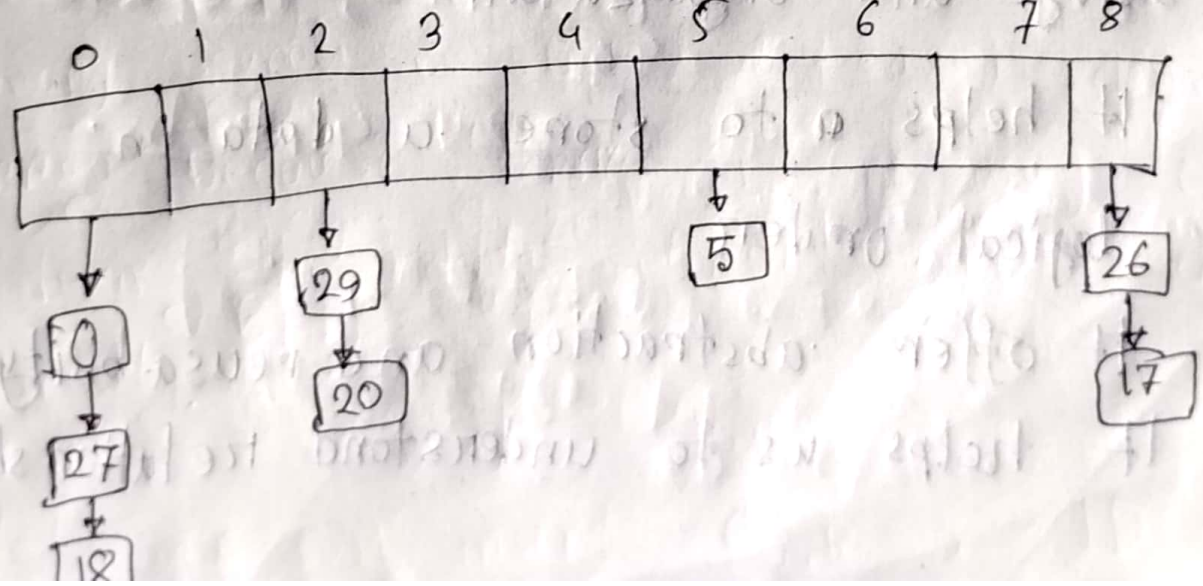
27 % 9 = 0

26 % 9 = 8

17 % 9 = 8

18 % 9 = 0

Hash table using chainig (collision resolution):



4(c)

Same as : 2021 → 4(c)

2019

1 (a)

Data structure is storing and organization of the data in such a way so that it can be used efficiently.

We are using data structure, here some necessaries is as follows:

I) It is necessary for efficient data storage and organization.

II) It helps to store a data in logical order

III) It offer abstraction and reusability

IV) It helps us to understand relationship

between one data with others.

v) It is essential for desiging and analyzing algorithms.

vi) It provide solutions to real world problems.

vii) It enable fast data retrieval and manipulation operations.

viii) Manupulation of large amount of data easier.

1 (b)

2020 – 1 (a)

1(c)

2020 – 1(c)

2(a)

~~For row major:~~

~~Base + w [n(i-1) + (j-1)]~~

$L_1 = 10 - 2 + 1 = 9$

$L_2 = -1 + 4 + 1 = 4$

$L_3 = 8 - 5 + 1 = 4$

$E_1 = 5 - 2 = 3$

$E_2 = -1 + 4 = 3$

$E_3 = 7 - 5 = 2$

For row major:

Address = Base + w [(E1 L2 + E2) L3 + E3]

= 200 + 4 [(3 · 4 + 3) 4 + 2]

= 200 + ~~4 [84 + 2]~~ 248

~~= 200 + 348~~

= 544 ~~326~~ 448 Ans.

for column major:

Base + w [(E3 L2 + E2) L1 + E1]

= 200 + 4 [(8 + 3) 9 + 3]

= 200 + 4 × 102

= 608

2(b)

An array PTR is called a pointer array if each element of PTR is a pointer.

A 2D array with high entries of null value is cause for memory waste, because array use static memory. consider the following list:

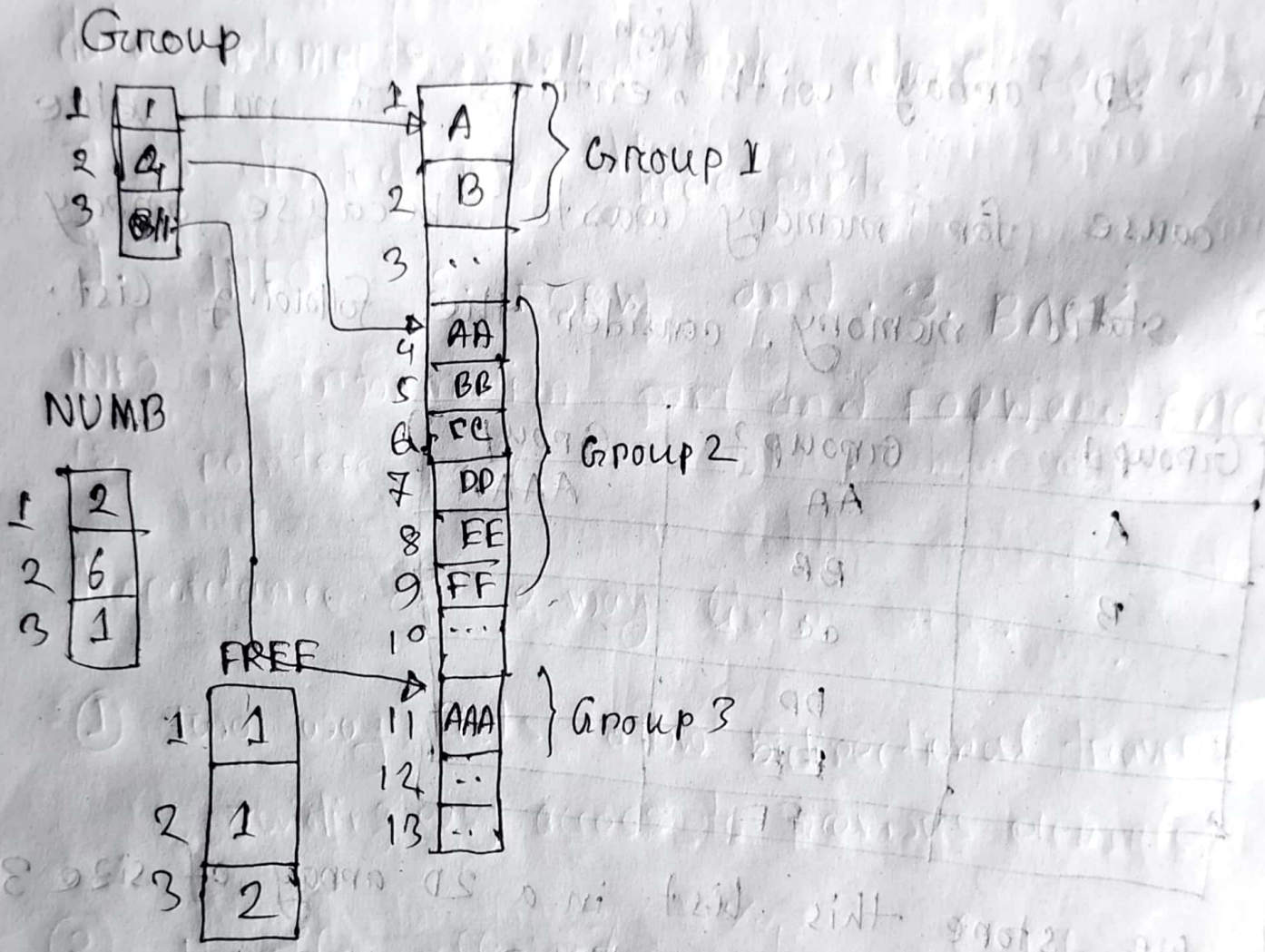
Group 1	Group 2	Group 3
A	AA	AAAA
B	BB	
	CC	
	DD	
	EE	
	FF	

If we store this list in a 2D array of size 3X6 off course it wastes 18 - 9 = 9 memory cell.

The best solution is to use a pointer array.

A pointer array is used to point each group of elements. The pointer array size depends on

the no. of groups, so we can save and utilize our memory space as needed and properly. If we use pointer as the previous list is as follows;



Fig; pointer array

2(c)

① An record may be a collection of non homogeneous data, where linear array store homogeneous data.

② The data items in a record are indexed by attribute names, so there may not be a natural order, ~~wich~~ which is occur in array.

3(a)

same as : | 2018 | 2(b) |

3(b)

same as : | 2021 | 2(c) |

~~8(b)~~

3(c)

A two way list is a linear collection of data elements, called node, where each node is divided into three parts:

1. INFO 2. FORW and 3. BACK,

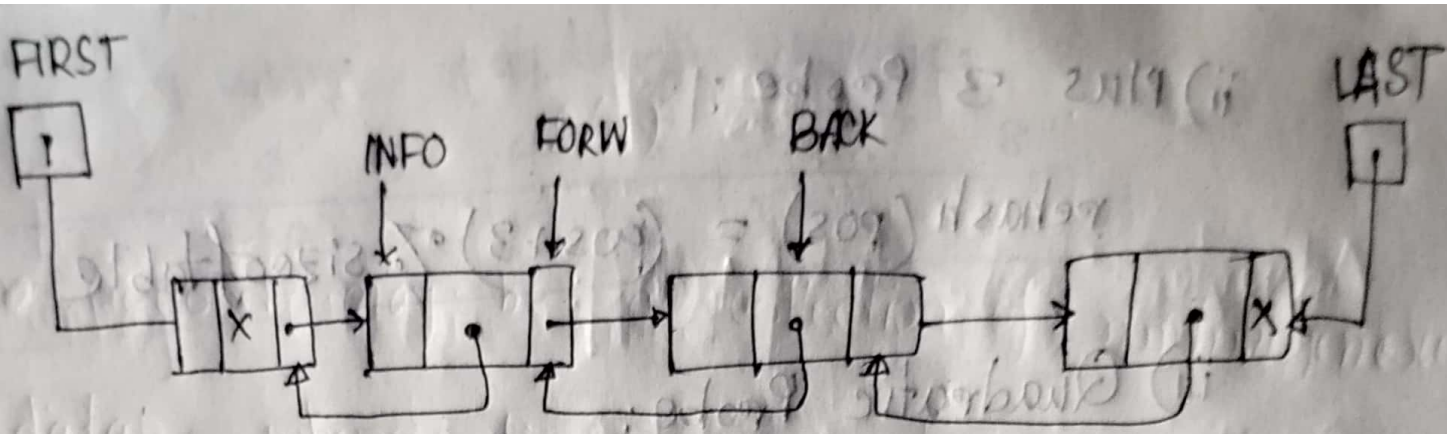
INFO is information part and FORW and BACK is pointer part.

Importance of 2 way list:

① two way lists allow bidirectional traversal, providing easy access to previous elements.

② It offer efficient insertion and delection operation

③ It can be easily reversed



Schematic Diagram of 2 way list.

4(a).

When two items hash to the same slot, we must have a systematic method for placing the second item in the hash table. This process is called collision resolution.

We have the following resolution process:

(1) Open Addressing:

we start at the ~~beginning~~ original hash value position and then move in a sequential manner through the slots until we encounter the first slot that is empty. It is 3 types:

i) Linear Probing:

rehash (pos) = (pos+1) % sizeof table

ii) Plus 3 Probe:

rehash(pos) = (pos+3) % sizeoftable

iii) Quadratic Probe:

If follow the first hash h, then the successive values are h+1, h+4, h+9 ... so on.

2) Chaining:

- It alows many items to exist at the same location in the hash table. When collision happen, the item is still placed in the proper slot of the hash table.

4(b)

size of table = 9

hash function = K % 9

Data: 5, 29, 20, 0, 27, 26, 17, 18

5 % 9 = 5
29 % 9 = 2
20 % 9 = 2
0 % 9 = 0
27 % 9 = 0
26 % 9 = 8
17 % 9 = 8
18 % 9 = 0

hash table using quadratic probe:

0	1	2	3	4	5	6	7	8
0	27	29	20	17	5	18		26

4(c)

Hash function = x mod 10

Data: 4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199

4322 % 10 = 2
1334 % 10 = 4
1471 % 10 = 1
9679 % 10 = 9
1989 % 10 = 9
6171 % 10 = 1
6173 % 10 = 3
4199 % 10 = 9

i) true
ii) true
iii) False
iv) False
v) False

2018

1(a)

~~Here~~ To sort the elements of a matrix, we have a few different options. Here two methods is written:

1) Sorting each row/column independtly:
   i) Apply a sorting algorithm to each row or column individually.
   ii) This preserves the relative order of elements within each row or column.

2) Sorting the entire matrix:
   i) Flatten the matrix into a 1-dimensional array.
   ii) Apply a sorting algorithm to the flattened array.
   iii) Reshape the sorted array back into the original matrix shape.

1 (b)

2019 - 2 (b)

1 (c)

$L_1 = 6 - 2 + 1 = 5$

$L_2 = -1 + 4 + 1 = 4$

$L_3 = 5 - 1 + 1 = 5$

$E_1 = 5 - 2 = 3$

$E_2 = -1 + 4 = 3$

$E_3 = 4 - 1 = 3$

For row major:

Base + w [n (i-1) + (j-1)]

= 100 + w [$(E_1 L_2 + E_2) L_3 + E_3$]

= 100 + 4 [(12+3) 5 + 3]

= 412

For column major:

Base + w [m (i-1) + (j-1)] [$(E_3 L_2 + E_2) L_1 + E_1$]

= 100 + 4 [(12+3) 5 + 3]

= 412

Linked List

A linnked list, or one way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.

2018 - 2 (a)

Header linked list:

A header linked list is linked list which always contains a special node, called the header node.

i) Grounded header:

- is a header list where the last node contains the null pointer.

ii) Circular linked list:

- is a header list where the last node point back to the header node.

Two-way list:

A two-way list is a linear collection of data elements, called node, where each node is divided into three parts;

i) INFO : information field.

ii) FORW : contains the location of the next node.

iii) BACK : " " " " preceding node.

2(b)

Advantage of linked list:

i) To insert or to delete data is easier than array.

ii) Efficient memory utilization as space is allocated only for required nodes.

iii) Dynamic size, it can grow or shrink as our requirement.

iv) Stack and queues are often easily implemented using linked list

v) Efficient for large data

~~Disadv~~ vi) Flexible data structure

Disadvantage:

i) It is complicated for beginners to understand

ii) In simple and small structure linked list requires extra memory spaces.

iii) Random access is not possible

iv) Traverse ~~cosume~~ consume more time.

v) Difficult to share data.

2(c)

2021 – 2(b)

3(a)

2020 – 2(b) 2(c)

3(b)

Stacks is represented usually by means of linear array.

For this reason we require a linear array.

A pointer variable Top which contains the location of the top element and a variable MAXSTK which indicates the maximum no. of elements.

Top = 0 or NULL indicates that the stack is empty.

Top = MAXSTK indicates that the stack is full.

3(c)

i) $A + (B - C + D) * E / F + G \uparrow H$

Prefix:

$A + (B - C + D) * E / F + G \uparrow H$

$= A + [-[BC] + D] * E/F + \uparrow [GH]$

$= A + [+ - BCD] * E/F + [\uparrow GH]$

$= A + [* + - BCDE] / F + [\uparrow GH]$

$= A + [/ * + - BCDEF] + [\uparrow GH]$

$= [+ A / * + - BCDEF] + [\uparrow GH]$

$= + + A / * + - BCDEF \uparrow GH$

Postfix:

$A + (B - C + D) * E / F + G \uparrow H$

$A + ([BC -] + D) * E/F + G \uparrow H$

$= A + (BC - D +) * E/F + [GH \uparrow]$

$= A + [BC - D + E *] / F + [GH \uparrow]$

$= A + [BC-D+E*F/] + [GH\uparrow]$

$= [ABC-D+E*F/+] + [GH\uparrow]$

$= ABC-D+E*F/+GH\uparrow+$

Prefix:

ii) $1+2-(3*4/5\uparrow6)*7$

$= 1+2-([*34]/[\uparrow56])*7$

$= 1+2-([/*34\uparrow56])*7$

$= 1+2-[*/*34\uparrow567]$

$= [+12]-[*/*34\uparrow567]$

$= -+12*/*34\uparrow567$

Postfix: $1+2-(3*4/5\uparrow6)*7$

$= 1+2-([34*]/[56\uparrow])*7$

$= 1+2-[34*56\uparrow/]*7$

$= [12+]-[34*56\uparrow/7*]$

= 12+ 34*56↑/7*-

4(a)

Polish notation refers to the notation in which the operator symbol is placed before its two operand.

Benefits:

To evalute infix expression, we need to follow the precedence of the opercators; otherwise connect result cannot be obtained. In case of prefix or postfix notation, parentheses never used, to determine the other order of the operations. And so postfix expression can be easily evaluated using computer.

4(b)

2021 – 3 (b)

4(c)

If we ~~insert 15 el~~ want to maintain 15 elements by array and another 15 are by linked list. The elements of the array are save by consecative memory location, on the other hand the elements of linked list is maintain by means of pointer.

If we access 9 th element in array it take ~~cost~~ constant time;

$O(1)$

But, if we access 9th element in linked list we start from start and move forward, until we reached the

9th position. so. It take

$O(9)$

We can tell from the above discuss Linked list consume more time.

2017

1(a)

A linear array is a list of finite no. n of homogeneous (same type) data elements.

No. of elements in linear array:

= UB - LB + 1

UB = upper bound

LB = lower bound

1(b)

2020 - 1(b)

1(c)

2020 - 1(c)

2(a)

2021 - 2(a)

2(b)

2020 - 2(c)

2(c)

2021 - 1(c)

3

← Check previous.

4(a)

	Stack	Queue
Order	LIFO	FIFO
Operations	Push, pop	QINSERT, QDELETE
Access	Only the top element	Both front and back elements are accessible
Use	recursion based problems	sequential processing based problems.

4(b)

prefix:

A + (B − C) * D / E + F / G ↑ H

= A + [− B C] * D / E + F / G ↑ H

= A + [* − B C D] / E + F / [↑ G H]

= A + [/ * − B C D E] + [/ F ↑ G H]

= [+ A / * − B C D E] + [/ F ↑ G H]

= + + A / * − B C D E / F ↑ G H

Postfix: $A + (B-C) * D/E + F/G \uparrow H$

$= A + [BC-] * [DE/] + F/[GH\uparrow]$

$= A + [BC-D*] / E + [FGH\uparrow /]$

$= A + [BC-D*E/] + [FGH\uparrow /]$

$= [ABC-D*E/+] + [FGH\uparrow /]$

$= ABC-D*E/+FGH\uparrow /+$

Prefix:

$(1+2) * 3 - 4/5 * 6 \uparrow 7$

$= [+12] * 3 - 4/5 * [\uparrow 67]$

$= [*+123] - [/45] * [\uparrow 67]$

$= [*+123] - [*/45\uparrow 67]$

$= -*+123*/45\uparrow 67$

Postfix: $(1+2) * 3 - 4/5 * 6 \uparrow 7$

$= [12+] * 3 - 4/5 * [67\uparrow]$

$= [12+3*] - [45/] * [67\uparrow]$

$= [12+3*] - [45/67\uparrow *]$

$= 12+3*45/67\uparrow *-$

4(d)

20, 5, 2, *, /, 2, 3, ↑, ↑, 4, /, -

Scanned	Stack
20	20
5	20, 5
2	20, 5, 2
*	20, 10
/	2
2	2, 2
3	2, 2, 3
↑	2, 8
↑	256
4	256, 4
/	64

This question is wrong.

2016

1, 2, 3

see previous

4(a)

12, 6, /, 6, 2, +, *, 12, 4, /, -

Scanned	Stack
12	12
6	12, 6
/	2
6	2, 6
2	2, 6, 2
+	2, 8
*	16
12	16, 12
4	see previous 16, 12, 4
/	16, 3
-	13

Ans: 13.

2015

← see previous

Mohon Kumer