MONDAY
JAN '23
2
002-363 WK-01

Data Structure can be classified as:

linear and non-linear
- linear means array and linear link-list.
- Non-linear such as a tree and graph.

operations: Traversing, Searching, Insertion, Deletion, Sorting, merging.

→ Array is used to store a fix size for data and in a link-list the data can be varies in size.

- linear Array
  Homogenous data

a) Elements are represented through indexs.
b) Elements are saved in sequential in memory locations.

- Number of elements N → length or size of an array.

- Representation of Array in a Memory

In general, index function: $Loc(x[i]) = Loc(x[LB]) + w*(i-LB)$

Eg: LB = 5, $Loc(x[LB]) = 1200$ and $w = 4$, find $Loc(x[8])$?

→ $Loc(x[8]) = 1200 + 4*(8-5)$
$= 1212.$

FEBRUARY
MARCH
APRIL

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

TUESDAY
JAN '23
3
WK-01 003-362

• Traversing Algorithms

→ Traversing operation means visit every element once

Step 1: [INITIALIZATION] SET I = LB (lower bound)
Step 2: Repeat steps 3 to 4 while I <= UB (Upper bound)
Step 3: Apply process to A[I]
Step 4: SET I = I+1 (END of Loop)
Step 5: EXIT.

• Insertion Algorithm

1) (INITIALIZATION) SET I = N
2) Repeat steps 3 to 4 while I >= POS
3) SET A[I+1] = A[I]
4) SET I = I-1 (END of Loop)
5) SET N = N+1
6) SET A[POS] = VAL
7) EXIT

• Insert item at the ~~end~~ last is easy if there is a space. Insert item on the middle requires the movement of all elements to the right.

December 2022

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	28	29	30	31								

WEDNESDAY
JAN '23
4
004-361 WK-01

- Fixed sized Array

C/C++
int arr[100] } (Stack allocated)
int arr[n]
int * arr = new int[n] } (Heap allocated)
int arr[] = {1,2,3,4} -> (Stack allocated)

Java
int [] arr = new int [100] } (Heap allocated)
int [] arr = new int [n]
int arr [] = {1,2,3}

- Dynamic Size Array. (Resize the array)

=> Arrayy list in Java.

Example program

```
import java.util.Arraylist;
class Test
{ public static void main (String args[])
  { Arraylist <Integer> al = new Arraylist <Integer>();
    al.add(10);
    al.add(20);
    al.add(30);
    System.out.println(al);
  }
}
```

THURSDAY
JAN '23
5
WK-01 005-360

• operations on Assay

Search (unsorted array)

I/p: arr [] = {20, 5, 7, 25}
x = 5
O/p = 1

I/p: arr [] = {20, 5, 7, 25}
x = 15
O/p = -1.

```
int Search (int arr[], int n, int x)
{
  for (int i=0; i<n; i++)
    if (arr[i] == x)
      return i;
  return -1;
}
```

• Insert an element

|5|7|10|20|_|

```
int insert (int arr[], int n, int x, int cap, int pos)
{
  if (n == cap)
    return n;
  int idx = pos - 1;
  for (int i=n-1; i>=idx; i--)
    arr[i+1] = arr[i];
  arr [idx] = x;
  return (n+1);
}
```

∴ insert · 3, pos = 2
O/p = |5|3|7|10|20|

FRIDAY
JAN '23
6
006-359 WK-01

Time complexity : O(n)
At the end of insertion : O(1)
At the beginning of insertion : θ(n)

Deletion Algorithm

a) Array indexc - 12, 3, 44, 19, 100 · · ·, 5, 10, 18

K = Size of an array.

Algorithm:
DELETE (LA, N, K, ITEM)
1) ITEM = LA[k]
2) Reapeat for I = K to N-1
[Shift element, forward]
LA[I] = LA[I+1]
3) [end of loop]
4) [Reset N in LA] : N = N-1
5) Exit

after deletion: 12, 3, 44, 100, · · ·, 5, 10, 48.

FEBRUARY
MARCH
APRIL

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

SATURDAY
JAN '23
7
WK-01 007-358

• Sequential Search

Compare successive elements of a given list with a search ITEM until

i) Either a match is encountered
ii) Or the list is exhausted without a match.

Algorithm

Sequential Search (LA, N, ITEM, LOC)

1) I = 0 // If LB = 0
2) Repeat Step 2.1 while (i < N and LA[I] != ITEM)
   2.1 I = I + 1
3) If LA[I] == ITEM then
   Return found at LOC = I.
   Else
   Return not found
4) Exit.

December 2022

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
19	20	21	22	23	24	25	26	27	28	29	30	31							

Binary Search Algorithm

SUNDAY
JAN '23
8
008-357 WK-01

- Binary Search algorithm is efficient if the array is sorted.
- A binary search is used whenever the list starts to become large.
- consider to use binary searches whenever the lists contains more than 16 elements.
- The binary search starts by testing the data in the element at the middle of the array to determine if the target is in the ~~list~~ first or second half of the list.
- ~~If it is in the first half, we do not need to check the second half. In words we eliminate half the list from further consideration~~
- If it is in the first half, we do not need to check the second half. If it is in the second half, we do not need to check the first half. In other words we eliminate half the list from further consideration. We repeat this process until we find the target or determine that it is not in the list.

```
BSEARCH (A,N item)
1) LOC = -1
2) B=1 , E=N
3) While B ≤ E
4) mid = [(B+E)/2]
5) if item = A[mid] then
6)     LOC = mid   {exit loop}
7) else if item > A[mid]
8)         B=mid+1
   else
9)         E = mid-1
10) Return LOC
```

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

FEBRUARY
MARCH
APRIL

MONDAY

JAN '23

9

WK-02 009-356

Search 87.

Q) B E

24	36	39	47	78	87	92	112	156
1	2	3	4	5 (mid)	6	7	8	9

$\frac{B+E}{2} = \frac{1+9}{2} = \frac{10}{2} = 5$.

On 5 - 78 (not match) (78 < 87)

next step: M+1 = B

5+1 = B ⇒ 6 is the new beginning.

$\frac{6+9}{2} = 7.5 \approx 7$

on 7 is 92 (not match)

92 > 87

next step: M-1 = B

7-1 = B

B = 6

now E is move to 6.

then, $\frac{B+E}{2} = \frac{6+6}{2} = 6$

on 6 is 87 matched

Binary search complete.

December 2022

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	28	29	30	31								

TUESDAY
JAN '23
10
010-355 WK-02

Merging Algorithm

Suppose A is a sorted list with R elements and B is a sorted list with S elements. The operation that combines the element of A and B into a single sorted list C with n = R+S elements is called merging.

Step 1: Set NA = 1, NB = 1 and NC = 1
Step 2: Repeat while NA ≤ R and NB ≤ S:
    if A[NA] ≤ B[NB], then:
        Set C[NC] = A[NA]
        Set NA = NA + 1
    else
        Set C[NC] = B[NB]
        Set NB = NB + 1 {End of if structure}
    Set NC = NC + 1
{End of loop}
Step 3: If NA > R, then:
    Repeat while NB ≤ S:
    Set C[NC] = B[NB]
    Set NB = NB + 1
    Set NC = NC + 1 {End of loop}
  else
  Repeat while NA ≤ R:
  Set C[NC] = A[NA]
  Set NC = NC + 1
  Set NA = NA + 1 {End of loop}
  {End of if structure}
Step 4: Return C[NC].

Time complexity: $f(n) \leq n = O(n)$

FEBRUARY
MARCH
APRIL

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

THURSDAY
JAN '23
12
012-353 WK-02

Time complexity

```
public static void main (String args [ ]) {
    Scanner sc = new Scanner (System.in);
    int n = sc.nextInt ();
    for (int i=0; i<n; i++) {
        System.out.println ("Hello")
}}
```

Time complexity - n.

- Best case {n(1)} .

  let example x = [1,2,3,4]
  find 1.

  Base case => pos. of 1 is 1.

- Average case x = [1,4,3,5,2] find 2.

  => $\underbrace{1+2+3+4+5}_{n}$ ∴ Average case - $\theta(\frac{n+1}{2})$

- Worst case x = [4,3,1,5,2] ∴ Worst case = O(n)

  => n.

FEBRUARY
MARCH
APRIL

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

FRIDAY
JAN '23
13
WK-02 013-352

Q. public static void main (String args[])
Scanner sc = new Scanner (System.in);
int n = sc.nextInt();
for (int i=0; i<n; i++) {
for (int j=0; j<n; j++) {
System.out.println ("hello");
}}

find time complexity:

> let n=5

i=0 -> j = 0, 1, 2, 3, 4
i=1 -> j = 0, 1, 2, 3, 4
} n times.
i = 4 -> j = 0, 1, 2, 3, 4

n times then, time complexity = $n \times n \Rightarrow n^2$
$= O(n^2)$

Q. for (int i=0; i<n; i++) {
for (int j=0; j<m; j++) {
print ("hello"); }}

> T.C > $n \times m \Rightarrow O(nm)$

Q. for (int i=0; i<n; i++) {
print ("hello");}
for (int j=0; j<m; j++) {
print ("hello");}

> TC = $n+m$ if n is 10^5 and m = 3 then
$= O(n+m)$ T.C. $n \Rightarrow$ Big O (n)

December 2022

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	28	29	30	31								

SATURDAY
JAN '23
14
014-351 WK-02

Compare :	$O(n)$	$O(n^2)$	$O(n^3)$
$n=1$	1	1	1
$n=2$	2	4	8
$n=3$	3	9	27
⋮			
$n=10^5$	10^5	10^{10}	10^{15}

- Comparison of functions on the basis of time complexity

$O(n)^n > O(n!) > O(n^3) > O(n^2) > O(n \cdot \log(n)) > O(n \cdot \log(\log(n)))$
$> O(n) > O(\sqrt{n}) > O(\log(n)) > O(1)$.

FEBRUARY
MARCH
APRIL

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

SUNDAY
JAN '23
15
WK-02 015-350

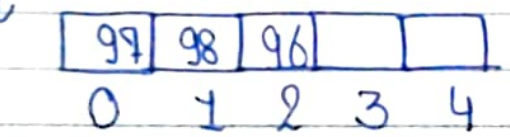
• Array in Java

Defining an array (1)

• type [ ] arrayName = new type [Size];

Enter marks of 3 Subjects.

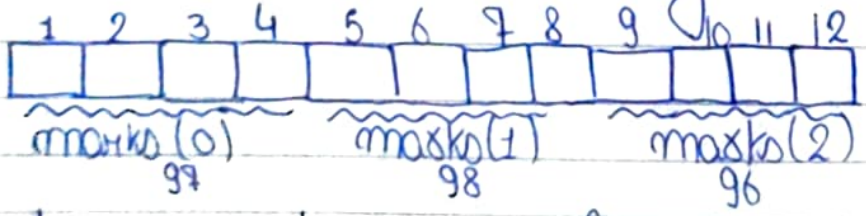
=> int [ ] marks = new int [5]
marks [0] = 97;
marks [1] = 98;
marks [2] = 96;



print the marks of 3 subject in one line code.

=> for (int i=0; i<3; i++) {
System.out.println (marks[i]); }

How data store in memory



because int takes 4 bit.

December 2022

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	28	29	30	31								

MONDAY
JAN '23
16
016-349 WK-03

Defining an array (2)

int marks[] = {97, 98, 96};

- write an function to take value from user.

```
import java.util.*;
public class Assayo {
    public static void main (String args[]) {
        Scanner sc = new Scanner (System.in);
        int size = sc.nextInt();
        int number[] = new int [size];
        // input
        for (int i=0; i<size; i++) {
            number [i] = sc.nextInt(); }
        // output
        for (int i=0; i<size; i++) {
            System.out.println (number [i]);
        }}}
```

FEBRUARY
MARCH
APRIL

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

TUESDAY

JAN '23

17

WK-03 017-348

• Write a code for finding a number x in array index. (All values are taken from user)

⇒ Algorithm.

```
for (int i=0; i<size; i++) {
    number[i] = sc.nextInt();
}
int x = sc.nextInt();
for (int i=0; i<number.length; i++) {
    if (number[i] == x) {
    System.out.println("x found at index " + i); }
```

1	3	4	5
0	1	2	3

x = 5

i = 0 ⇒ 1 ≠ 5

i = 1 ⇒ 3 ≠ 5

i = 2 ⇒ 4 ≠ 5

i = 3 ⇒ 5 = 5 ✓

This concept is called linear search.

December 2022

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	28	29	30	31								

WEDNESDAY
JAN '23
18
018-347 WK-03

2) - arrays in Java

row = 3
column = 5

1) declaration: type [][] arrayName = new type [row][column];
int [][] number = new int [3][5];

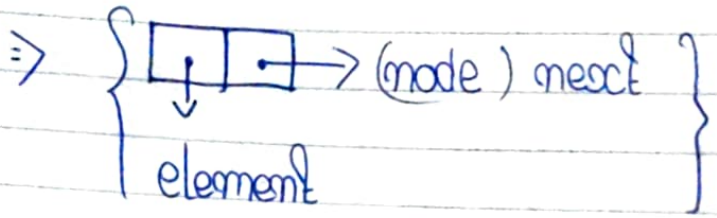
• Declare 2) array.

```
import java.util.*;
public class TwoDarrays {
    psvm (String args[]) {
    Scanner sc = new Scanner (System.in);
     int row = sc.nextInt();
     int column = sc.nextInt();
     int [][] number = new int [row][column];

// input, row for (int i=0; i<row; i++) {
 // column      for (int j=0; j<col; j++) {
                    number [i][j] = sc.nextInt(); }}

// output    for (int i=0; i<row; i++){
              for (int j=0; j<cols; j++) {
                    System.out.print (number [i][j] + " "); }
                System.out.println();
    }}}
```

SUNDAY
DEC '23
31
WK-52 365-000

Unit-2 (linked list)

• A linked list (one-way list) is a linear collection of data elements, called nodes, where the linear order is given by means of pointer.
→ Each node is divided into two parts.
→ First part contains the information of the element.
→ Second part contains the address of the next node of the list.



• Important points
→ linear collection of nodes
→ Connected by pointer links
→ Accessed via a pointer to the first node of the list.
→ link pointer in the last node is set to null to mark the list's end.
→ linked list contains a list Pointer variable called START or NAME, which contains the address of the first node.

• Need of linked list
~ Many applications require resizing.
~ Required size not always immediately available.

• Uses of linked lists instead of an array when
~ You have an unpredictable number of data elements.
~ You want to insert and delete quickly.

November 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30										

SATURDAY
DEC '23
30
364-001 WK-52

- Types of linked lists

i) Singly linked lists

~ Begins with a pointer to the first node.
~ Terminates with a null pointer.
~ Only traversed in one direction.

ii) Circular, Singly linked

~ Pointer in the last node points back to the first point.

iii) Doubly linked lists
~ Two start pointers - 1st element and last element
~ Each node has a foward pointer and backward element.
~ Allows traversals both fowards and backward.

iv) Circulary, doubly linked lists
~ Foward pointer of the last node points to the first node and backward pointer of the first node points to the last node.

January 2024

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31											

```
Node current = head;
while (current.next != null) {
    current = current.next;
}
    current.next = newNode; }
public void traversal() {
    Node current = head;
    while (current != null) {
     S.o.p (current.data + " ");
     current = current.next; }}}

Public class main {
    psvm (String[] args) {
      LinkedList list = new LinkedList();
      my list.append(1);
          "   "   (5);
        S.o.p (" Linked list element:");
        my.list traverse(); }}
```

WEDNESDAY
DEC '23
27
WK-52 361-004

• Searching (Iterative)

```
int search (Node head, int x)
{
  int pos = 1;
  Node curr = head;
  while (curr != null) {
    if (curr.data == x)
        return pos;
    else {
      pos++
      curr = curr.next
    }}
    return -1;
  }
```

• Memory Allocation

~ Together with the linked list, a special list is maintained which consists of unused memory cells.
~ This list has its own pointer.

• Garbage collection

~ Garbage collection is a technique of collecting all the deleted spaces or unused spaces in memory.
~ Garbage collection may take place when there is only some minimum amount of space or no ~~place~~ space is left in free storage list.
~ It is invisible to the programmer.

November 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30										

TUESDAY
DEC '23
26
360-005 WK-52

- Garbage collection Process

i) The computer runs through all lists tagging those cells which are currently in use.
ii) Then computer runs through the memory, collecting all the untagged spaces onto the free storage list.

- overflow
When a new data are to be inserted into a data structure but there is no available space i.e, the free storage list is empty.
- Overflow occurs when AVAIL = NULL, and we want insert an element.
- It can be handled by printing the overflow message and/or by adding space to the underlying data structure.

- Underflow
When a data item is to be deleted from an empty data structure.
Underflow occurs when START = NULL, and we want to delete an element.
Underflow can be handled by printing the UNDERFLOW message.

January 2024

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31											

MONDAY
DEC '23
25
WK-52 359-006

• Insertion
- Insertion at beginning

1.) If AVAIL = NULL, then: write: OVERFLOW, and exit.
2.) {Remove first node from AVAIL list}
Set NEWNODE = AVAIL and AVAIL = AVAIL -> LINK.
3.) Set NEWNODE -> INFO = ITEM. (Copy the new data to the node)
4.) Set NEWNODE -> LINK = START. (New node points to the original first node.)
5.) Set START = NEWNODE. (START points to the new node.)
6.) EXIT.

• Java

```
class Node {
    int data;
    Node next;
    Node (int x)
    {
      data = x;
      next = null;
    }}
class Test {
    static Node insertBegin (Node head, int x) {
    Node temp = new Node (x);
     temp.next = head;
    return temp;
    }
```

November 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30										

psvm (String [] args) {
Node head = null;
head = insertBegin (head, 30);
" " (" 28);
return 0; } }

SUNDAY
DEC '23
24
358-007 WK-51

. Insertion (at the end)

} Overflow if AVAIL = NULL

Java.

```
class Node {
int data;
Node next;
Node (int x) {
data = x;
next = null; }
class Test {
public static Node insertEnd (Node head, int x) {
Node temp = new Node(x);
if (head == null)
    return = temp;
Node curr = head;
while (curr.next != null)
    curr = curr.next;
curr.next = temp;
return head;
}
```

January 2024

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31											

SATURDAY
DEC '23
23
WK-51 357-008

• Insertion (between two nodes)

~ If ITEM is to be inserted into a sorted linked list. Then ITEM must be inserted between nodes A and B such that: INFO[A] < ITEM < INFO[B].

First of all find the location of NODE A.
Then insert the node after NODE A.

Java:

```
Node insertPos(Node head, int pos, int data) {
  Node temp = new Node(data);
  if (pos == 1) {
     temp.next = head;
     return temp; }
  Node curr = head;
* { for (int i = 1; i < pos-2; && curr != null; i++)
     curr = curr.next.
  if (curr == null)
        return head;
* { temp.next = curr.next;
  curr.next = temp;
  return head;
  }
```

November 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30										

TREE

THURSDAY
FEB '23
2
033-332 WK-05

- A tree is a hierarchical structure.
- collection of nodes or finite set of nodes.
- This collection can be empty Nodes and Edges.

Node: A node is the key component of the tree data structure that stores the data element and may contain zero, one or more link to other sucessor node for connectivity. It consists finite set of nodes.

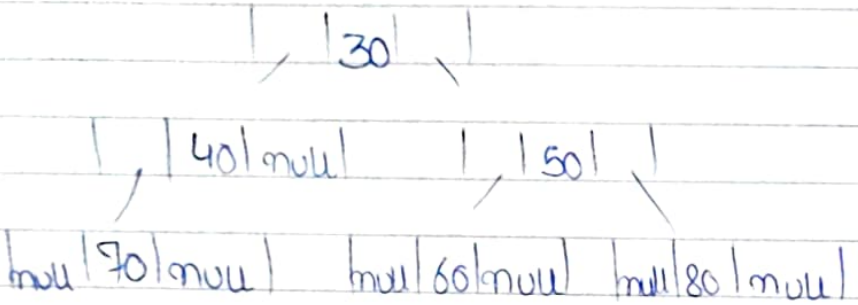
FRIDAY
FEB '23
3
WK-05 034-331

• Binary Tree

~ A binary tree is a special form of tree in which a node can have atmost two children.

~ A node in a binary tree maynot necessarily have the maximum of two children i.e, either 1, 2 or 0 child.

~ A tree which does not contain any node is called empty binary node.



~ Java

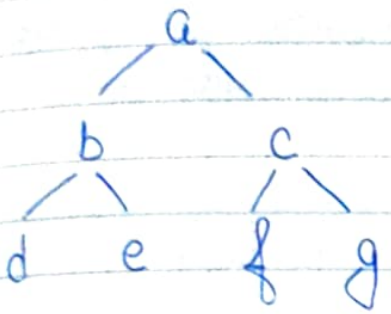
```
class Node
{
    int key;
    Node left;
    Node right;
    Node (int k)
    {
        key = k;
    }
class Test {
    psvm (String[] args)
    {
```

January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

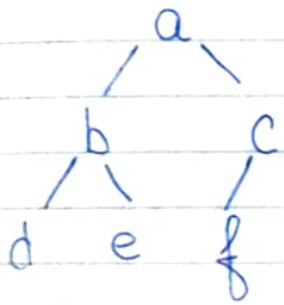
```
Node root = new Node(10);
root.left = new Node(20);
root.right = new Node(30);
root.left.left = new Node(40);
}}
```

- Full binary tree



Each node has exactly zero or two children.

- complete binary tree



A complete binary tree is either a full binary tree or one in which every level is fully occupied except possibly for the bottommost level where all the nodes must be as far left as possible.

- Properties

~ The maximum number of nodes in a binary tree on a given level (say L) is 2^L, where $L >= 0$.

~ The maximum number of nodes in an binary tree with height h is $2^{h+1} - 1$.

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

MARCH
APRIL

SUNDAY
FEB '23
5
WK-05 036-329

~ The minimum number of nodes possible in a binary tree of height h is $h+1$.

~ If n is the number of nodes and e is the number of edges in an non-empty binary tree then $n = e+1$.

~ If n_0 is the number of leaf nodes (no child) and n_2 is the number of nodes with two children in a non-empty binary tree then $n_0 = n_2 + 1$.

~ For a complete binary tree T with n nodes, the height is floor $(\log_2^{(n+1)} - 1)$.

• Tree traversal

~ Breadth First Search
~ Depth First Search (Inorder, Preorder, Postorder)

Recursive
Traverse root Traverse left subtree Traverse right subtree.

• Inorder (left Root Right)
• Preorder (Root left Right)
• Postorder (left Right Root)

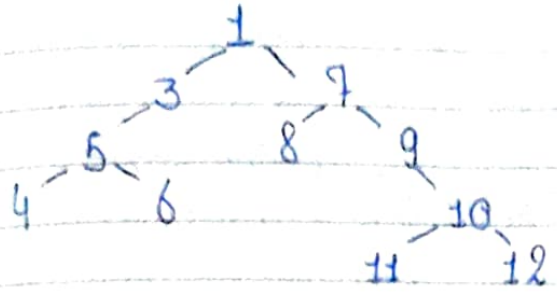
• Infix - postfix
• Infix - prefix

January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

MONDAY
FEB '23
6
037-328 WK-06

Q.



Preorder: 1 3 4 5 6 7 8 9 10 11 12
Postorder: 4 6 5 3 8 11 12 10 9 7 1
Inorder: ~~1 4 5 6 3 8 11 12 1~~
4 5 6 3 1 8 7 9 11 10 12

- Inorder Traversal

Java ~

```
void inorder (Node root)
{
  if (root != null)
  {
    inorder (root.left)
    System.out.println(root.key + " ");
    inorder (root.right);
  }}
```

Arithmetic ~ A/B*C*D+E

MARCH
APRIL

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

TUESDAY
FEB '23
7
WK-06 038-327

• Preorder traversal

Java ~ (+ * * / A B C ^ ^ E)

```
void preorder (Node root)
{
    if (root != null)
    {
        System.out.println(root.key + " ");
        preorder (root.left);
        preorder (root.right);
    }
}
```

```
void postorder (Node root)
{
    if (root != null)
    {
        postorder (root.left)
        postorder (root.right)
        System.out.println(root.key + " ");
    }
}
```

{ A B * C * D * E + }
{ A B / C * D ^ E ^ + }

January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

WEDNESDAY
FEB '23
8
039-326 WK-06

Arithmetic Expression

preorder -> + * * / A B C D t
prefix expression

inorder -> A / B * C * D + t
infix expression

postorder -> A B / C * D * t +
postfix expression

- Binary Search Tree

Binary Search tree ~~abbe~~ abbreviated as BST is a special form of binary tree whose nodes arranged in such a way that for every node N, the values contained in all nodes in its left sub tree are less the value contained in N and values contained in the right sub tree are larger than the node N.

- Searching

- Searching an item is the key operation performing on a BST. When we want to search given item in BST, we begin by first comparing the item with the value in root.

a) If they are equal then the location of the root node is ~~returned~~ returned.
b) If the item is less than the value of the root then we need to search the left sub tree of the root. The right sub tree is eliminated.

MARCH
APRIL

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

THURSDAY
FEB '23
9
WK-06 040-325

c) If the item is greater than the value of the root then we need to search the right sub tree of the root.

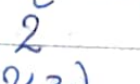
- Construct a BST.

values[] = {5, 1, 3, 4, 2, 7} compare to make BST

left node is always less weight as compare to right node

=>

(l.w) (1<5) 1 7 (7>5) (m.w)



(2<3) (3<4)

less weight more weight

January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

FRIDAY
FEB '23
10
041-324 WK-06

- Binary Search Tree (Background)

	Array(us)	Array(s)	Linked	BST	Hash
Search	O(n)	O(logn)	O(n)	O(logn)	O(1)
Insert	O(1)	O(n)	O(1)	"	"
Delete	O(n)	O(n)	O(n)	"	"
Find closest	O(n)	O(logn)	O(n)	"	"
Sorted Traversal	O(nlogn)	O(n)	O(nlogn)	O(H)	O(nlogn)

- Search BST in Java

```
boolean search (Node root, int x)
{
  while (root != null)
  {
    if (root.key == x)
        return true;
    else if (root.key < x)
        root = root.right;
    else
        root = root.left;
  }
  return false;
}
```

∴ Time complexity : O(H) : H = Height
auxiliary space by recursive = O(H)
" " " Iterative = O(1)

MARCH
APRIL

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

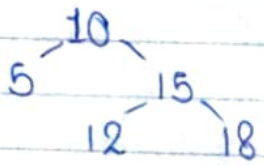
SATURDAY
FEB '23
11
WK-06 042-323

. Insertion in BST (Java)

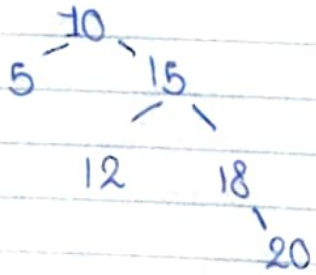
∴ Recursion

```
Node insert (Node root, int x)
{
    if (root == null)
        return new Node (x);
    if (root.key > x)
        root.left = insert (root.left, x);
    else if (root.key < x)
        root.right = insert (root.right, x);
    return root;
}
```

x = 20



⇒

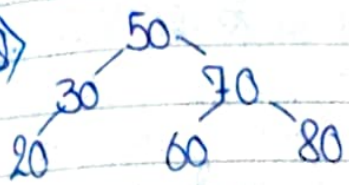


January 2023

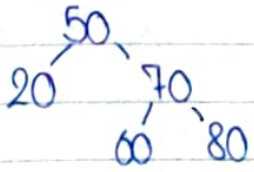
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

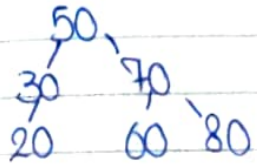
SUNDAY
FEB '23
12
043-322 WK-06

• Deletion in BST (Java)

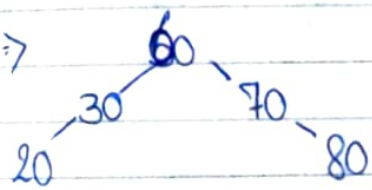
Q) 

x = 30

=> 

Q) 

x = 50
take closest highest value

=> 


```
Node delNode (Node root, int x)
{
  if (root == null) return null;
  if (root.key > x)
      root.left = delNode (root.left, x);
  else if (root.key < x)
      root.right = delNode (root.right, x);
  else
  {
    if (root.left == null) return root.right;
```

MARCH
APRIL

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

MONDAY
FEB '23
13
WK-07 044-321

```
else if (root.right == null) return root.left;
else
{
    Node succ = getSucc(root);
    root.key = succ.key;
    root.right = delNode(root.right, succ.key);
}
return root;
}
}
Node getSucc(Node root)
{
Node curr = root.right;
while (curr != null && curr.left != null)
    curr = curr.left;
return curr;
}
```

January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

TUESDAY
FEB '23
14
045-320 WK-07

• Infix to postfix.

i) priorities of operations: ^ highest, (*, /) → next, (+, -) lowest.

ii) no two operators of same priority can stay ~~before~~ together in stack column.

iii) lowest priority cannot be placed before highest priority.

Eg. (A + B / C * (D + E) - F)

Symbol	Stack	Postfix
A	(	A
+	(+	A
B	(+	AB
/	(+ /	AB
C	(+ /	ABC
*	(+ *	ABC /
(	(+ * (	ABC /
D	(+ * (	ABC / D
+	(+ * (+	ABC / D
E	(+ * (+	ABC / D E
)	(	ABC / D E + * +
-	(-	ABC / D E + * +
F	(-	ABC / D E + * + F
-	-	ABC / D E + * + F -

ABC / D E + * F - ans.

MARCH
APRIL

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

WEDNESDAY
FEB '23
15
WK-07 046-319

• Infix to prefix.

Q. A * B - C / D + E

=> A * B - C / D + E
=> (* AB) - C / D + E
(* AB) - (/ CD) + E ∴ * AB = T1 , / CD = T2
T1 - T2 + E
(- T1 T2) + E ∴ - T1 T2 = T3
T3 + E
+ T3 + E
+ - T1 T2 E
+ - * AB / CD E

Q. (A + B) / C * D - E

+ AB / * CD - E
T1 / T2 - E
/ T1 T2 - E
T3 - E

(+ AB) / C * D - E
(T1 / C) * D - E
(/ T1 C) * D - E
T2 * D - E
* T2 D - E
T3 - E
- T3 E
- * T2 D E
- * / T1 C E
- * / + ABC D E

THURSDAY
FEB '23
16
047-318 WK-07

Q. Convert postfix to infix.

AB- D E + F * /

Reading of Postfix	Stack Top	Expression
• A	A	A
• B	B	B A
• -	(A-B)	A-B
• D	D	D A-B
• E	E	E D A-B
• +	D+E	D+E A-B
• F	F	F D+E A-B
• *	D+E*F	D+E*F A-B
• /	A-B / D+E*F	A-B / D+E*F

A-B / D+E*F ans.

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

FRIDAY
FEB '23
17
WK-07 048-317

• Prefix to infix

Q. + - * AB / C ↑ E

Reverse prefix expression

E ↑ C / B A * - +

Reading	Stack top	Expression
E	E	E
↑	↑	↑ E
C	C	C ↑ E
/	C / ↑	C / ↑ E
B	B	B C / ↑ E
A	A	A B C / ↑ E
*	A * B	A * B C / ↑ E
-	A * B - C / ↑	A * B - C / ↑ E
+	A * B - C / ↑ + E	A * B - C / ↑ + E

∴ A * B - C / ↑ + E ans.

January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

SATURDAY
FEB '23
18
049-316 WK-07

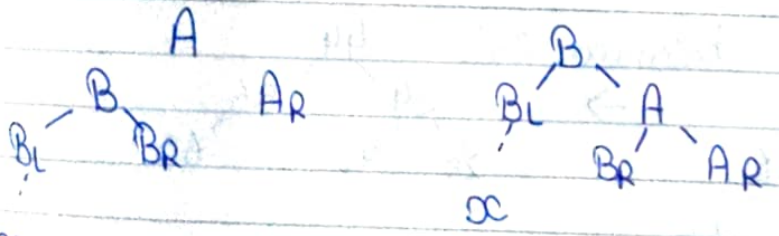
. AVL Tree means Balanced BST

For each node, the height of the left and right subtrees can differ at most 1, -1, 0.

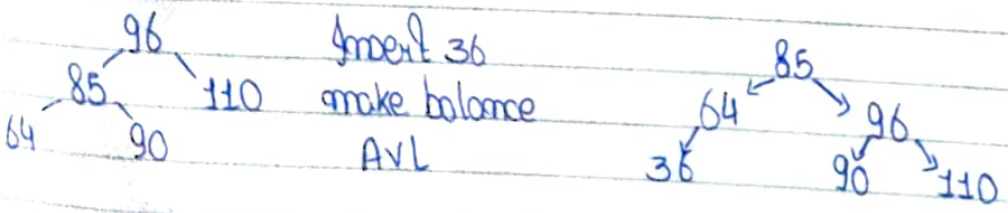
Rotations: Inserting an element in an AVL search tree in its first phase is similar to that of the one used in a binary search tree.

. LL Rotation: Inserted node in the left subtree of the left subtree of A.

This rotation is done when the element is inserted in the left subtree of A. To rebalance the tree, it is rotated so as to allow B to be the root with B_L and A to be its left subtree and right child and B_R and A_R to be the left and right subtrees of A. The rotation results in a balanced tree. (clockwise)



x → new node



∴ Balance Factor = (height of left subtree - height of right subtree)

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

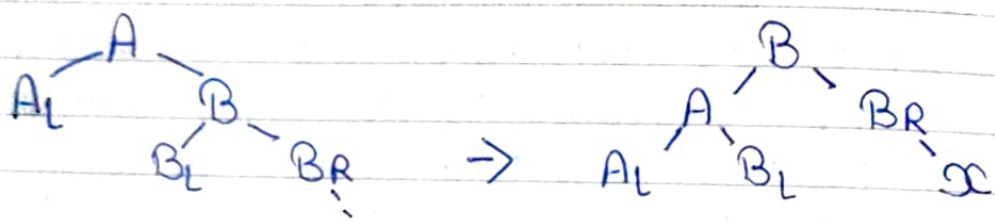
SUNDAY

FEB '23

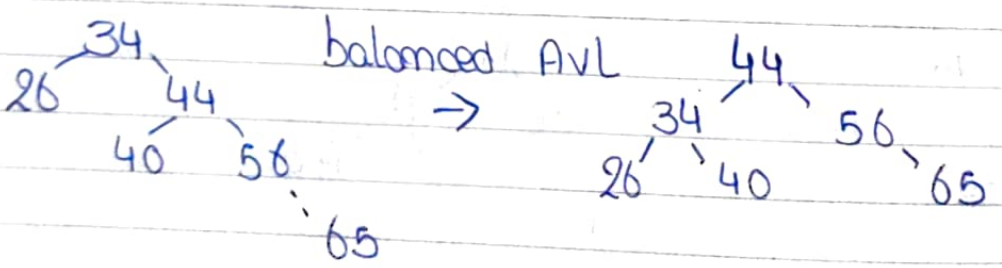
19

WK-07 050-315

RR Rotation - This rotation is applied if the new element is inserted right subtree of right subtree of A. The rebalancing rotation pushes B upto the root with A as its left child and B_R as its right subtree and A_L and B_L as the left and right subtrees of A. (anti-clockwise).



x → new node

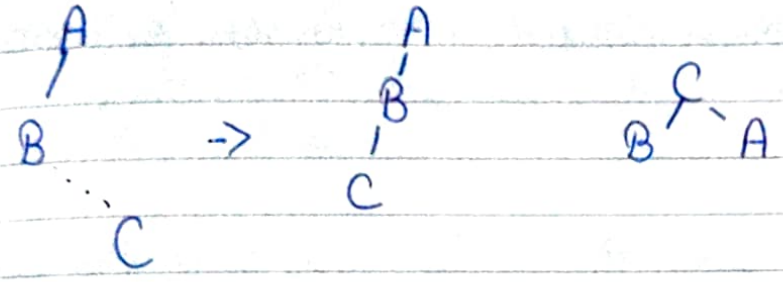


January 2023

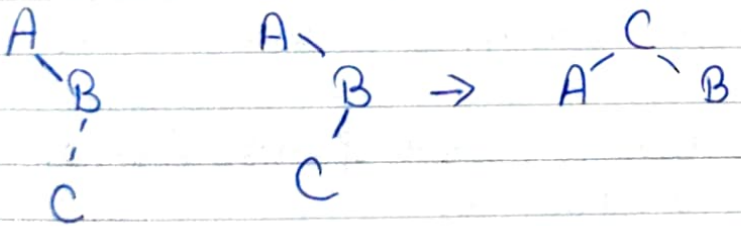
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

MONDAY
FEB '23
20
051-314 WK-08

LR (left to right): When x is inserted to the left or right of A, that is in the left subtree of the right subtree of A, the rules of rotation should be used for balancing (clockwise)



RL (Right of left) When x is inserted to the right of left of A, that is, in the right subtree of the left subtree of A, the rules of rotation should be used for balancing. (anti-clockwise)



MARCH
APRIL

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

TUESDAY
FEB '23
21
WK-08 052-313

• AVL Tree Deletion

• For deletion, after rotation, we need to continue tracing upward to see if AVL-Tree property is violated at other node.

Different rotations are classified in R_0, R_1, R_{-1}, L_0, L_1 and L_{-1}.

L Rotation

~ L Rotation is applied when the deleted node is in the left subtree of node A.
There are three different types of rotations based on the balanced factor of node B.

• L_0 Rotation: When the balance factor of node B is 0.
Apply RR rotation on node A.

• L_1 Rotation: When the balance factor of node B is -1.
Apply RR rotation on node A.

• L_{-1} Rotation: When the balance factor of node B is +1.
Apply RL rotation (LL rotation on B and RR rotation on node A)

January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

WEDNESDAY
FEB '23
22
053-312 WK-08

- Heap Tree

~ A heap tree is a complete binary tree in which data values stored in any node is greater than or equal to the value of search of its children.

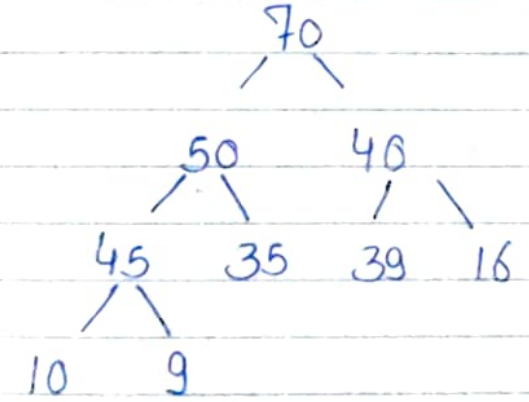
There is only restriction between parent and its children and not within the left or right subtree as in case of binary search trees.

The value stored in the root node of a heap tree is always the largest or smallest in tree. Such a heap tree is called max-heap or min-heap.

- Max-heap

for every node i, the value of node is less than or equal to its parent value.

$A[parent(i)] \geq A[i]$ {except root node}



70	50	40	45	35	39	16	10	9
1	2	3	4	5	6	7	8	9

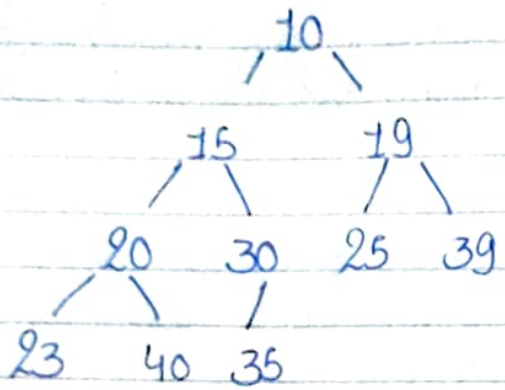
MARCH
APRIL

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

THURSDAY
FEB '23
23
WK-08 054-311

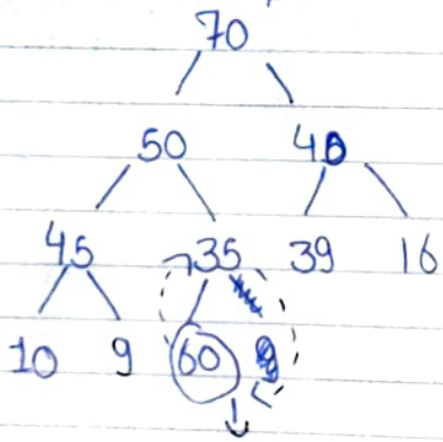
• Min heap
for every node i, the value of node is greater than or equal to its parent value.
$A[parent[i]] \leq A[i]$

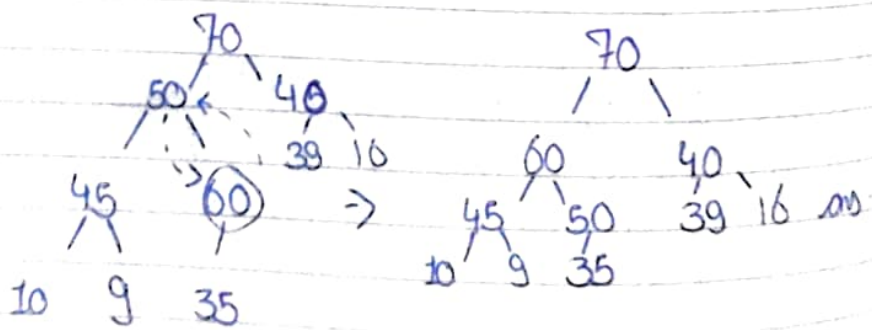


10 | 15 | 19 | 20 | 30 | 25 | 39 | 23 | 40 | 35

Q. Insert 60. (max heap)

Time complexity $\geq O(\log$





January 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

FRIDAY
FEB '23
24
055-310 WK-08

Code.

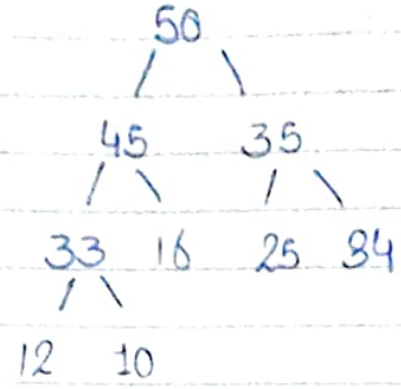
```
insert Heap (A, n, value)
{
    n = n + 1;
    A[n] = value;
    i = n;
while (i > 1)
{
    parent = [i/2]
    if (A [parent] < A [i])
    {
      swap (A [parent], A[i]);
          i = Parent;
    }
   else
      {
        return;
      }
  }
}
```


March 2023

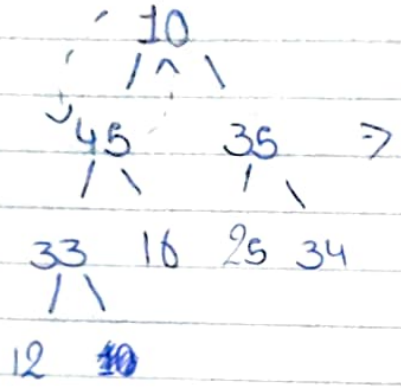
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

SATURDAY
FEB '23
25
WK-08 056-309

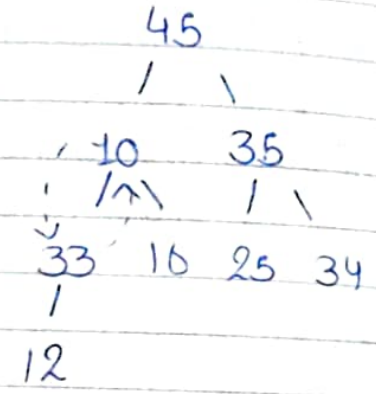
max heap deletion (always delete root)
OR
(lowest in right most)



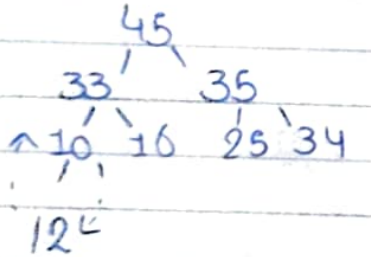
=>



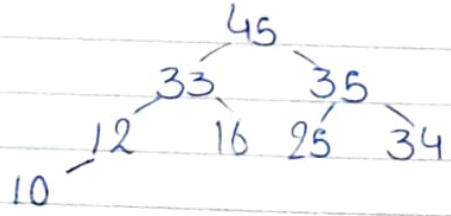
=>



=>



=>

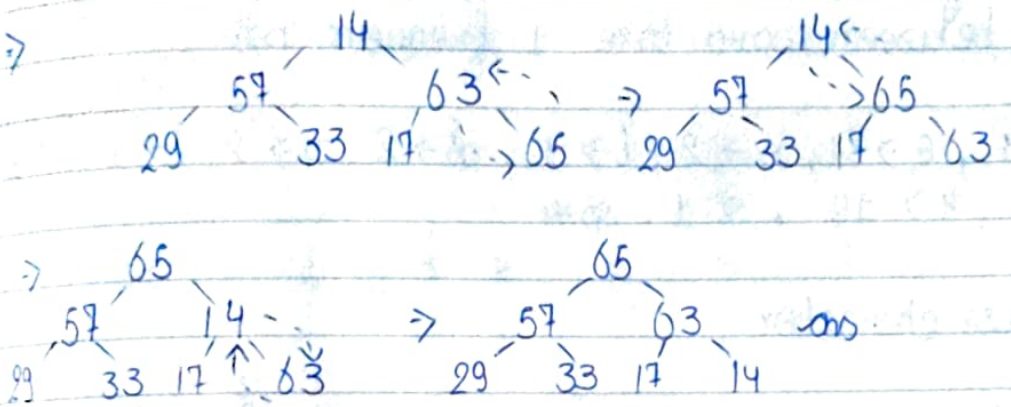


Time complexity ~ O(log n)

SUNDAY
FEB '23
26
057-308 WK-08

• Heap sort

Q. 14 57 63 29 33 17 55
max heap sort.



Time complexity → O(nlogn).

March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

MONDAY

FEB '23

27

WK-09 058-307

• Huffman coding

. Eerie eyes seen near lake .

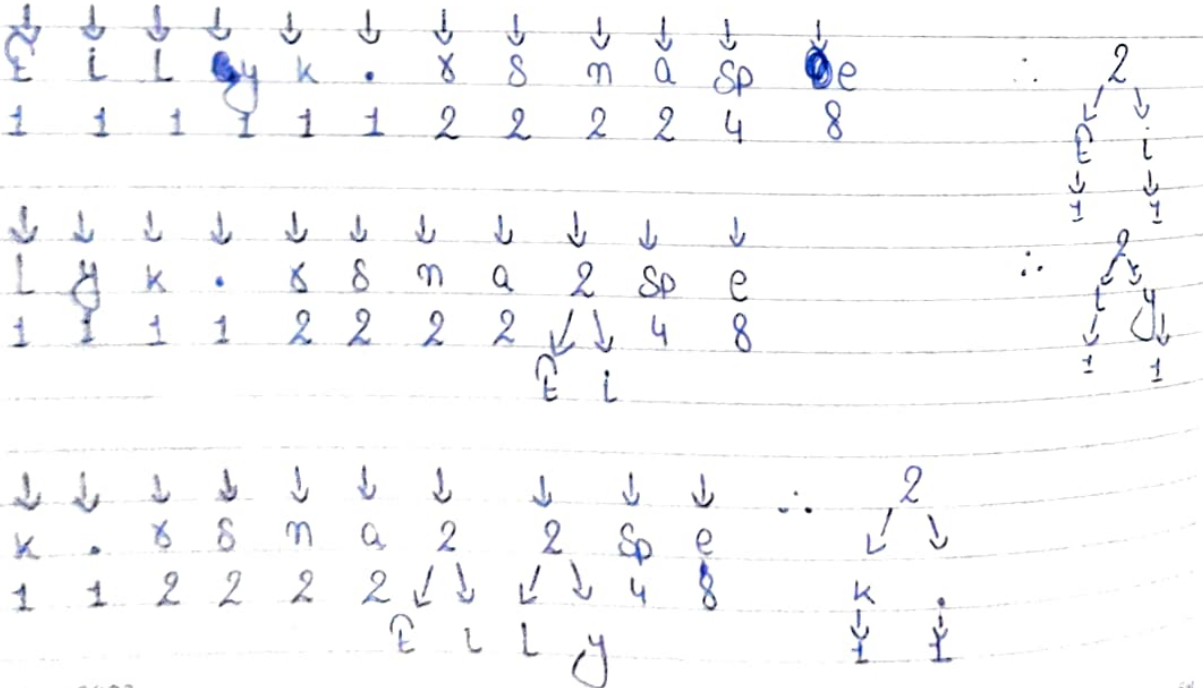
→ Space between word take 1 frequencies.

→ E → 1, e → 8, r → 2, i → 1, s → 2, n → 2, L → 1, a → 1, y → 1, k → 1, . → 1, Sp → 4

• Prioritize characters

~ Create binary tree nodes with character and frequency of each character.

~ Place nodes in a priority queue.
The lower the occurrence, the higher the priority in the queue.

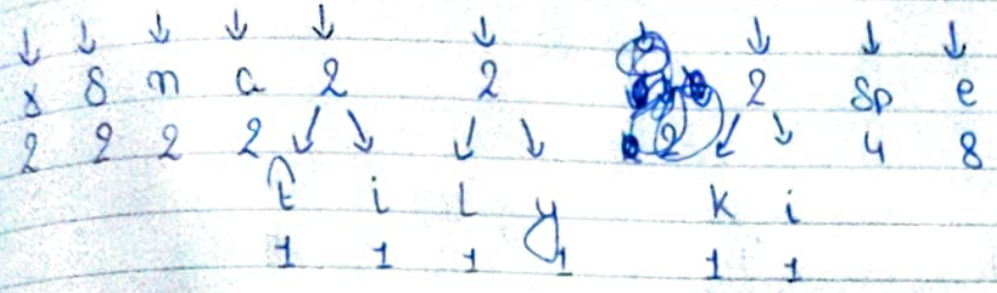


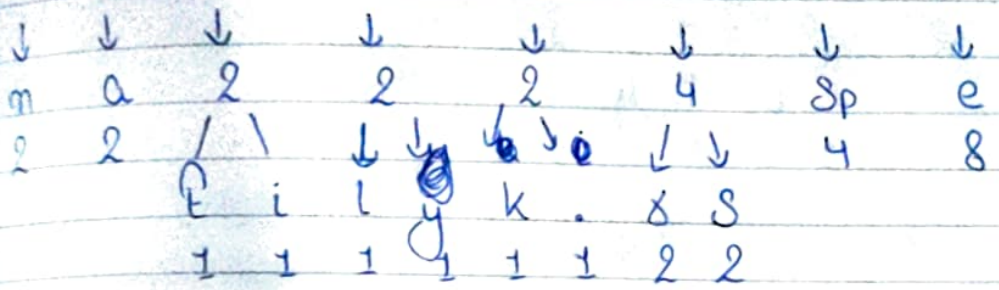
TUESDAY

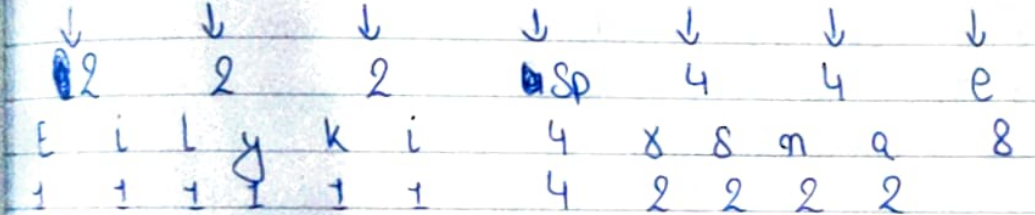
FEB '23

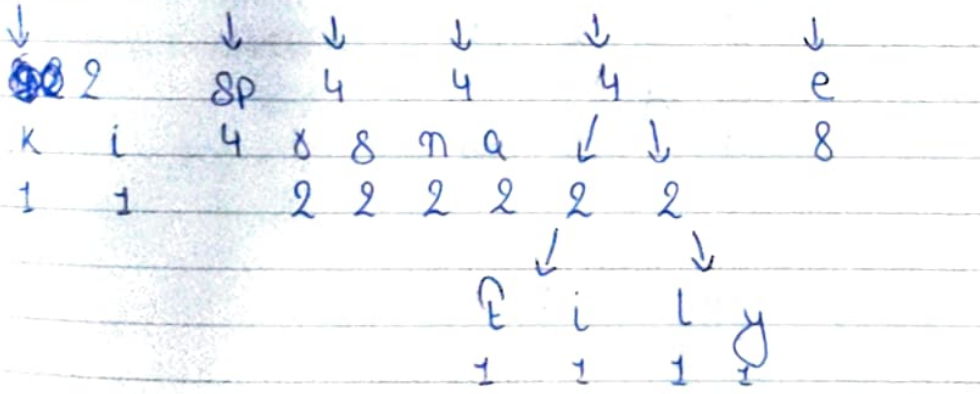
28

059-306 WK-09









March 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31									

MARCH

APRIL

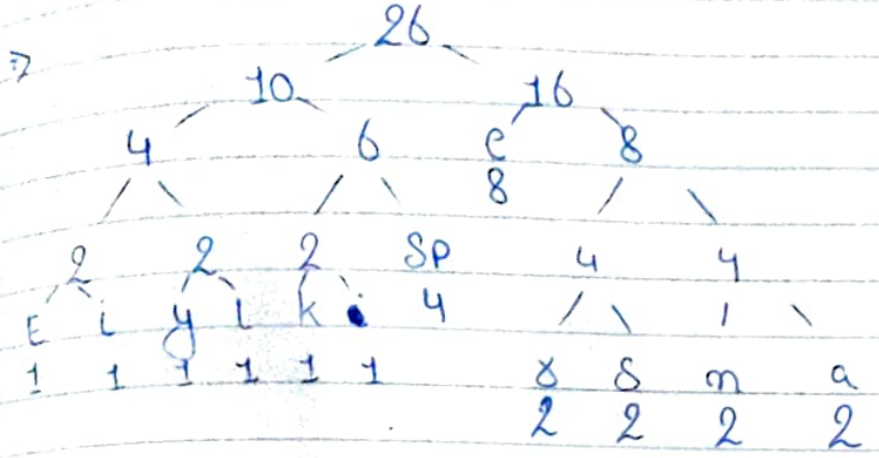
NOTES

4 4 4 6 8
⇒ δ S n a / \ / ↓ 8
2 2 2 2 2 2 2 4
Ei ly ki Sp
1 1 1 1 1 1

4 6 8 e
⇒ 2 2 2 4 4 4 8
Ei ly ki Sp δ S n a
1 1 1 1 2 2 2 2

8 e 10
4 4 8 4 6
⇒ δ S n a 2 2 2 4
2 2 2 2 Ei ly ki 4
11 11 11 Sp

⇒ 10 16
4 6 8 8
2 2 2 4 4 4
Ei ly ki Sp δ S n a
11 11 11 4 2 2 2 2



Char	
E	0000
i	0001
y	0010
L	0011
k	0100
.	0101
SP	0110
e	0100
x	1100
δ	1101
m	1110
a	1111

WEDNESDAY

MAR '23

1

WK-09 060-305

• Convert infix to postfix.

(K + L - M * N + (O ^ P) * W / U / V * T + Q)

Symbol	Stack	Postfix
K	(	K
+	(+	K
L	(+	K L
-	(-	K L +
M	-	K L + M
*	- *	K L + M
N	- *	K L + M N
+	+	K L + M N - *
O	+	K L + M N - * O
^	+ ^	K L + M N - * O
P	+ ^	K L + M N - * O P
*	*	K L + M N - * O P ^ +
W	*	K L + M N - * O P ^ + W
/	/	K L + M N - * O P ^ + W *
U	/	K L + M N - * O P ^ + W * U
/	/	K L + M N - * O P ^ + W * U /
V	/	K L + M N - * O P ^ + W * U / V
*	*	K L + M N - * O P ^ + W * U / V /
T	*	K L + M N - * O P ^ + W * U / V / T
+	+	K L + M N - * O P ^ + W * U / V / T *
Q	+	K L + M N - * O P ^ + W * U / V / T * Q
-	-	K L + M N - * O P ^ + W * U / V / T * Q +

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

THURSDAY
MAR '23
2
061-304 WK-09

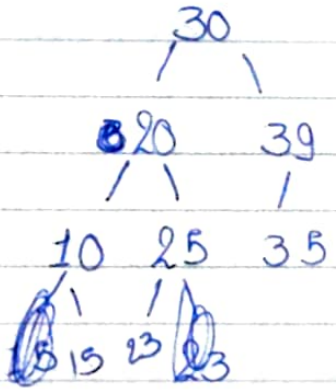
Q.) construct a unique binary tree.
30, 20, 10, 15, 25, 23, 29, 35, 39 (preorder traversal)

To construct binary tree we make the array in ascending order i.e, inorder traversal.

=> 10, 15, 20, 23, 25, 29, 30, 35, 39

We know in preorder sequence is (Root, left, Right).
So, 30 is the root and in inorder traversal before root is left part of tree and after is the right part of tree so,

10, 15, 20, 23, 25, 29 is left sub tree and 35, 39 is right sub tree.



April 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30							

APRIL

SATURDAY
MAR '23
4
063-302 WK-09

- Hashing
- linear search has a running time proportional to O(n), while search, binary search takes time proportional to O(log n) while in hashing its takes O(1).

- We will use the term hash table for an array and the function that will carry out the transformation will be called a hash function.

Different hash function

- Division method
  h(x) = x mod M (It is best to choose M is prime number)

Q. Calculate the hash values of keys 1234 and 5462. (100 two digit)

=> 100 two digits => 00 . . . 99 (chose closest prime number to 99 i.e, 97)

=> h(1234) = 1234 mod 97
= 70
h(5462) = 5462 mod 97
= 16

APRIL

April 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30							

SUNDAY
MAR '23
5
WK-09 064-301

• Mid Square method

The mid square method is a good has function which works in two steps:

1) Square the value of the key. That is find k^2.
2) Extract the middle x digits of the result obtain in step 1.

• Calculate the hash value for keys 1234 and 5642 using the mid-square method. (100 memory locations of two digits)

⇒ x = 2 (two digits) chose 3rd and 4th digit.

$k^2 = (1234)^2 = 1522756 = 27$
$k^2 = (5642)^2 = 31832164 = 21$

• Collisions
~ Collisions occur when the hash function maps two different keys to the same location.

• Resolving collisions by open addressing

i) linear probing

$h(k,i) = [h'(k) + i] \bmod m$.
i is the probe number.
i always start from 0 to m-1.

MONDAY
MAR '23
6
065-300 WK-10

Q 72, 27, 36, 24, 63, 81, 92 : m = 10

72 mod 10 = 2 , 36 mod 10 = 6 , 63 mod 10 = 3
27 mod 10 = 7 , 24 mod 10 = 4 , 81 mod 10 = 1
92 mod 10 = 2.

∴ 72 and 92 is get same place then

$h(k,i) = (h'(k,i) + i) \bmod m$

= 2 + 1 mod 10
= 3 is also occupied then

= 2 + 2 mod 10
= 4 also occupied then

= 2 + 3 mod 10
= 5 is vaccant also, 92 placed there.

0	1	2	3	4	5	6	7
-	81	72	63	24	92	36	27

ans.

APRIL

April 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30							

TUESDAY
MAR '23
7
WK-10 066-299

• Quadratic Probing

$h(k,i) = [h'(k) + c_1 i + c_2 i^2] \bmod m$.

Q. Consider a hash table of size 10. Using quadratic probing, insert the keys 72, 27, 36, 24, 63, 81, 101 into the table.
∴ $m = 10$, $c_1 = 1$ and $c_2 = 3$.

⇒ $72 \bmod 10 = 2$, $27 \bmod 10 = 7$, $36 \bmod 10 = 6$, $24 \bmod 10 = 4$,
$63 \bmod 10 = 3$, $81 \bmod 10 = 1$, $101 \bmod 10 = 1$.

1 is occupied by 81 then.

$h(k,i) = [h'(k,i) + c_1 i + c_2 i^2] \bmod 10$.

⇒ $[1 + 1 \cdot 1 + 3 \cdot 1^2] \bmod 10$
$5 \bmod 10$
$= 5$

∴ 5 is a vacoant placed 101 in 5.

0	1	2	3	4	5	6	7	8
-	81	72	63	24	101	36	27	-

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

WEDNESDAY
MAR '23
8
067-298 WK-10

• Double hashing

$h(k,i) = [h_1(k) + i h_2(k)] \bmod m$. i start

Q. Consider a hash table of size = 10. Using double hashing, insert the keys 72, 27, 36, 24, 63, 81, 101. Take $h_1 = k \bmod 10$, $h_2 = k \bmod 10$.

→ $72 \bmod 10 = 2$, $27 \bmod 10 = 7$, $36 \bmod 10 = 6$, $24 \bmod 10 = 4$, $63 \bmod 10 = 3$, $81 \bmod 10 = 1$, $101 \bmod 10 = 1$.

$\therefore h(k,i) = [h_1(k) + i h_2(k)] \bmod m$

$= [101 \bmod 10 + 0 \cdot 101 \bmod 8] \bmod 10$

$= [1 + 0] \bmod 10$

$= 1 \bmod 10$ not accept

$= (101 \bmod 10 + 1 \cdot 101 \bmod 8) \bmod 10$

$= (1 + 5) \bmod 10$

$= 6$ not accept.

$= (101 \bmod 10 + 2 \cdot 101 \bmod 8) \bmod 10$

$= (1 + 10) \bmod 10$

$= 1$ not accept its goes continue

$\therefore$ if 0 is vaccant then placed it.

APRIL

April 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30							

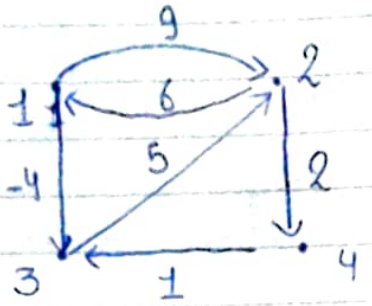
SATURDAY

MAR '23

11

WK-10 070-295

• All pair shortest path Floyd warshall



	1	2	3	4
1	0	9	-4	∞
2	6	0	∞	2
3	∞	5	0	∞
4	∞	∞	1	0

$D^0 \Rightarrow$

	1	2	3	4
1	0	9	-4	∞
2	6	0	2	2
3	∞	5	0	∞
4	∞	∞	1	0

$D^1 \Rightarrow$

At past 1(i,j) will be same as it is.

$\therefore D^0(2,3) = D^0[2,1] + D^0[1,3)$ always take box.

$\infty > = 6 + (-4)$

$= +2$ so 2 in (2,3)

$\therefore D^0(2,4) = D^0[2,1] + D^0(1,4)$

$2 < = 6 + \infty$

$\therefore D^0(3,2) = D^0(3,1) + D^0(1,2)$

$5 < \infty + 9$

$\therefore D^0(3,4) = D^0(3,1) + D^0(1,4)$

$\infty = \infty + \infty$

February 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28												

SUNDAY
MAR '23
12
071-294 WK-10

$\Delta^0(4,3) \Rightarrow \Delta^0(4,1) + \Delta^0(1,3)$

$1 < \infty + (-4)$

Q. Find value of $\Delta^2, \Delta^3, \Delta^4$

2, 3, 4 taken as a middle element.

APRIL

April 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30							