DSA ASSIGNMENT

NAME : BHUT TUSHAR
ROLL NO : 20BCP023

Name :- Bhut Tushar G.
Roll no :- 20BCP023
Subject :- DSA

1] Primitive and Non-Primitive data types.

Primitive data type	Non-Primitive data type
• Basic data types directly operated by Machine instructions.	• Group of homogeneous or heterogeneous Primitive data types.
• Ex. Integer, Float, Character, Boolean, Pointer	Ex Array, String, Enumeration, Structure, Union, etc.

2] Linear and Non-linear Data Structure.

Linear Data structure	Non-linear data Structure
1] It have data elements arranged in Sequential manner.	1] It haven't set sequence of connecting all its elements
• Each member of elements of linear data structure is connected to its previous and next element	• Each element of non-linear data structure can have multiple paths to connect to other elements.
• elements of linear data structure are present in single level and can be traversed in single run.	• elements of non-linear data structure can be present in multi-levels and often can't be traversed in single run.
• Array, Stack, Queue, linked list	• Graph, trees

CELEBRO
DATE
PAGE 5

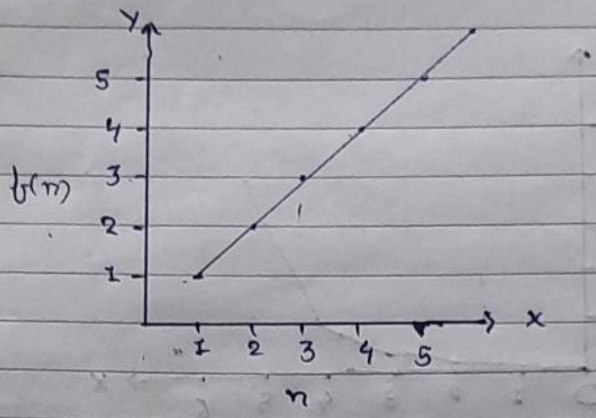
3] Space and time complexity. 20BCP023

Space Complexity	Time Complexity
• It is the amount of computer memory that is required from start of its execution to its completion with respect to the input size.	the amount of time taken by an algorithm to run with respect to the input size.
• It can be calculated as 1] Fixed component 2] Variable component	• It can be calculated by counting number of elementary operations performed by algo.
• 1] Fixed component: Space needed to store instructions, constants, variables, and structured variables. 2] Variable Component: Space needed for recursion stacks and for structured variables that are allocated space dynamically during the runtime of program.	It considers that each elementary operation takes a fixed amount of time to perform.

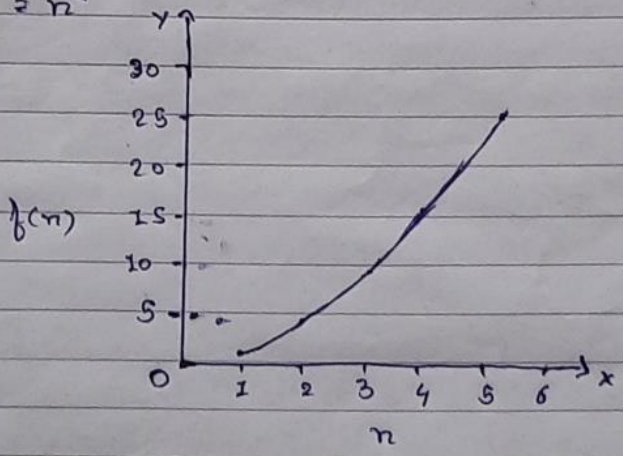
4]

Big Oh	Omega	Theta
• It measures worst case time complexity. • It is the formal way to express upper bound of an algo's time. • Notation : O	• It measures best case time complexity. • It is the formal way to express lower bound of an algo's time. • Notation : Ω	• It measures average case time complexity. • Notation: Θ

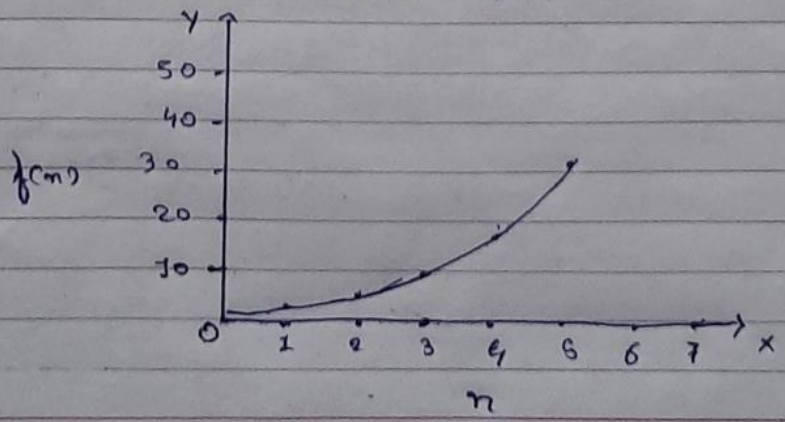
1] $f(n) = n$ 20BCP023



2] $f(n) = n^2$

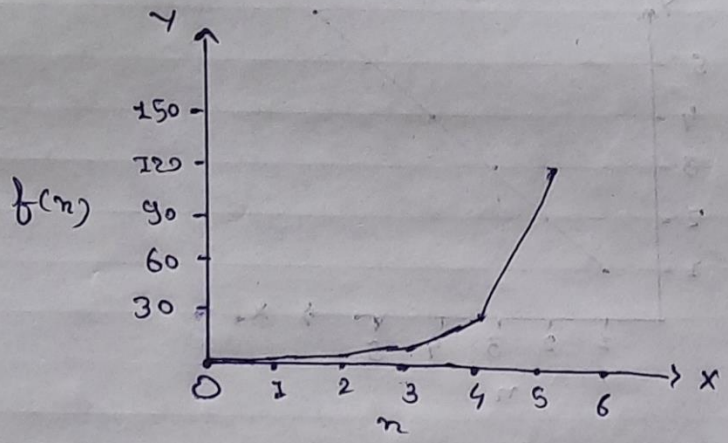


3] $f(n) = 2^n$



20BCP023

4] $f(n) = n!$



Arrays

Write Psuedocode :

----->1.

For an array of integers perform the following operations: (1) Insert at Index (2) Delete first element (3)Traverse in Reverse order.

```
Pseudocode:
 declare an array A, size, index, element.
 Get input of  array size and elements from user.
 1. get element and index where user want to insert element by user.
 for  i=size-1 ;i>=index-1 ;i--
        A[i+1] = A[i]
    A[index-1] = element

 2. Deleting first element
  for  i=0 ;i<size ;i++
       A[i] = A[i+1]

 3. Traverse in reverse order
      for   i=size-1 ;i>=0 ;i--
         print A[i]
```

---->2.

For a string perform the following operations: (1) Check Palindrome (2) Find occurrence of a given character (3) Compare 2 strings.

Pseudocode:

```
 declare int i=0 and h=(length of str1)-1, count=0, a=0,character
str1 and str2 .

 get a string entered by user in variable str1 and str 2.

  1. Check Palindrome
     for i=0 ;i<h+1 ;i++
          if str1[i] != str1[h-i]
               print String is not Palindrome
               break
     print String is Palindrome

  2. Get input a character c for finding its occurrence in string
     for i=0 ;i<=h ;i++
          if str[i] == c
               count++
     print count

  3. Compare two strings
     str1 and str2 are taken as input from user
     i = 0
     if length(str1) == length(str2)
          while str1[i] != '\0'
               if str1[i] != str2[i]
                    print Strings are not equal
                    break
               i++

          print strings are equal
     else
          print strings are not equal
```

---->3.

Multiply two 3 x 3 matrices.

Pseudocode:

```
 function to get elements of matrices.
 for int i=0; i<row; i++
     for int j=0; j<column; j++
          scanf("%d",&matrix[i][j])

int main()
get rows and columns for matrices A and B by user.
declared two matrices A(r1,c1) and B(r2,c2)
where r1,r2 = raw and c1,c2 = column

if (c1!=r2)
     print multiplication is not possible Enter data again.
    get rows and columns for matrices A and B by user

get element for matrices A and B by calling above function.
 //Matrice Result[r1][c2] initialize all element by 0

for  i=0; i<r1; i++
     for int j=0; j<c2; j++
          Result[i][j]=0
```

```
 for i=0; i<r1; i++
     for j=0; j<c1; j++
          for k=0; k<c2; k++
              Result[i][k] += A[i][j]*B[j][k]

 //Print Result
for int i=0; i<r1; i++
     for int j=0; j<c2; j++
          print Result[i][j].
```

Stack

---1>

Pusedocode to check a palindrome string with stack.

```
Pseudocode:
define Max 100 and initialize stack[MAX], top=-1 and front=0.

push(char a) // for pushing a character to stack
     Stack[++top] = a.

pop() // for getting a topmost element from stack
      return Stack[top--]

main()
     char s[100];
     Print Enter String:
     scanf(“%s”,&s)
     i = 0
     while i<strlen(s)/2
          push(str[i])
          i++;
     if strlen(s) % 2 == 0
          i++

     bool isPalindrome = true
```

```
    while s[i] != '\0'
        char c = pop()
        if c != s[i]
            isPalindrome = false
            break;
        i++
    if isPalindrome
        print Strig is Palindrome
    else
        print String isn't Palindrome
```

---2>

Psuedocode to convert infix expression into prefix.

```
Pseudocode:
define Max 100 and initialize stack[MAX], top =-1.
isEmpty()
    If top < 0
        return -1.
    return 0;
push(char x)
    stack[++top] = x.
pop()
    if(!isEmpty())
        return stack[top--]
peek()
```

```
    return stack[top].
precedence(char x)
    if x == '('
        return 0;
    if x == '+' or x == '-'
        return 1;
    if x == '*' or x == '/'
        return 2
    if x == ^
     return 3
    return 0.
checkIfOperand(char ch)
    return ch >= 'a' and ch <= 'z') or (ch >= 'A' and  ch <= 'Z'
getPostfix(char exp[])
    int i, j
    for i = 0, j = -1; exp[i]; ++i
        if checkIfOperand(exp[i])
            exp[++j] = exp[i]
        else if exp[i] == '('
            push(exp[i])
        else if exp[i] == ')'
            while !isEmpty() and peek(stack) != '('
                exp[++j] = pop()
            if !isEmpty() and peek() != '('
                return -1.
```

```
            else
                pop()
        else
              while !isEmpty() and precedence(exp[i] <= precedence(pee
         k())
                exp[++j] = pop()
            push(exp[i])
    while !isEmpty()
        exp[++j] = pop()
    exp[++j] = '\0'
reverse(char exp[])
int size = strlen(exp)
j = size, i = 0
char temp[size]
temp[j--] = ‘\0’
while(exp[i] != ‘\0’)
     temp[j] = exp[i]
     j-- i++
strcpy(exp,temp)
brackets(char exp[])
    int i = 0
    while exp[i]!='\0'
        if exp[i]=='('
            exp[i]=')'
        else if exp[i]==')'
            exp[i]='('
```

```
        i++.
main()
    char exp[100];
    print "The infix is: "
    gets(exp)
    InfixtoPrefix(exp)

    Print "The prefix is: "
    puts(exp)
```

---3>

Convert the following expressions into prefix and postfix using stack: (1) a*(b-c*d)+e (2) a+((b-c)*d)/e .

Infix expression	Postfix expression	Prefix expression
a*(b-c*d)+e	abcd*-*e+	+*a-b*cde
a+((b-c)*d)/e	abc-d*e/+	+a/*-bcde

---4>

Psuedocode to evaluate a postfix expression.

```
Pseudocode:
 define Max 100 and initialize stack[MAX] , top = -1.
 push(int ele)
     If top >= MAX-1
```

```
        Print “stack overflow”.
    Else
        Top = top + 1;
        Stack[top] = ele
 pop()
    If top < 0
         print “stack under flow”.
     Else
         Int item = stack[top]
         top = top - 1
         return item.
evaluate(char exp)
Initialize A, B and ans.
For int i=0; exp[i] != ‘\0’; i++
    char ch = exp[i]
    if isdigit(ch)
        push(ch-‘0’)
    else if ch==’+’ or ch==’-‘ or ch==’*’ or ch==’/’ or ch==’$’
        B = pop()
        A = pop()
        switch (ch)
            case '*':ans = A * B
                     break
            case '/':ans = A / B
                     break
```

```
            case '+':ans = A + B
                     break
            case '-':ans = A - B
                     break
             case '$':ans = pow(A,B)
                     break
             push(ans);
  print “"Result of expression evaluation : %d",pop()”
main()
    char exp[MAX]
    print "Enter an Expression : "
    scanf("%s",&exp)
    evaluate(exp)
```

---5>

Evaluate the following expressions using stack: (1) 34+86-* (2) 222$$3*2+2* .

expression	Postfix evaluation
1) 34+86-*	14
2) 222$$3*2+2*	100

---6>

Psuedocode for Fibonacci series with recursion.

```
Pseudocode:
declare fiboinacci function
int fibonacci(int n)
   if n==0 or n==1
       Return 1
   else
      Return Fibonacci(n-1)+Fibonacci(n-2)
 main()
     declare n.
     print “Enter a number of terms:”
     scanf n.
     print “%dth term:”.
     fibonacci(n)
```

Queue

---1>

Psuedocode for Linear Queue.

```
Pseudocode:
define max = 5

struct queue
     declare front as integer
     declare rear as integer
     declare ele[max]

intit(q,front,rear)
     front = 1
     rear = 0
     return
insert(q, front,rear,max,item)
     if(rear = max)
          display “Queue is Overflow”
          return
     rear++
     q->ele[rear] = item
     return

delete(q,front,rear,item)
```

```
     if(front = rear+1)
          display “Queue is Underflow”
          return
     item = q->[front++]
     return
isEmpty(q,front,rear)
     if(front = rear+1)
          return true
     return false
isFull(q,front,rear,max)
     if(rear = max)
          return true
     return false

call function as usage
```

---2>

Psuedocode for Double-Ended Queue.

```
Pseudocode:
Define max = 5
struct dequeue
     declare front as integer
     declare rear as integer
     declare count as integer
     declare ele[max]
```

```
intit(dq,front,rear,count)
     front = 1
     rear = -1, count = 0
     return
insertAtFront(dq,front,rear,max,count,item)
     if(count == max)
          print “DeQueue is Overflow”
     if(front = 1)
          front = max
     else front = front-1
     dq->ele[front] = item
     count++
     return

isEmpty(dq,count)
     if(count = 0) return false
     return true
isFull(dq,count,max)
     if(count = max)return true
     return false
inserAtRear(dq,front,rear,count,item)
     if(isFull(dq))
          print “Dequeue is overflowed”
          return
     dq->[rear] = item
```

```
     count++
deleteFromRear(dq,front,rear,count,max,item)
     if(count = 0)
          print “DeQueue is underflowed”
          return
     q->ele[rear] = item
     if(rear = 1)
          rear = max
     else rear = rear -1
     count = count – 1
     return


deleteFromFront(dq,front,rear,count,max,item)
     if(isEmpty(dq))
          print “Dequeue is underflowed”
          return
     item = q->ele[front]
     front++
     count = count – 1

call function from main() as usage
```

Linked-list

---1>

Psuedocode for singly linked list operations.

```
Pseudocode:
struct node
     declare data as integer
     declare next as node pointer (node*)
insert(node* head, item)
     node* cur = head
     while(cur != NULL)
          cur = cur->next
     cur->data = item
traverse(node* head)
     node* cur = head
     while(cur != NULL)
          print cur->data+” ”
          cur = cur->next
delete(node* head,item)
     node* cur = head
     while(cur != NULL)
          if(cur->data == item)
               Node*temp = cur
               cur = cur->next;
               free(temp)
          cur = cur->next
```

---2>

Psuedocode for stack using linked list.

Pseudocode:

- Step 1 - Include all the **header files** which are used in the program. And declare all the user defined functions.
- Step 2 - Define a '**Node**' structure with two members **data** and **next**.
- Step 3 - Define a **Node** pointer '**top**' and set it to **NULL**.
- Step 4 - Implement the **main** method by displaying a Menu with a list of operations and make suitable function calls in the **main** method.

push()

- Step 1 - Create a **newNode** with a given value.
- Step 2 - Check whether stack is **Empty** (**top** == **NULL**)
- Step 3 - If it is **Empty**, then set **newNode** → **next** = **NULL**.
- Step 4 - If it is **Not Empty**, then set **newNode** → **next** = **top**.
- Step 5 - Finally, set **top** = **newNode**.

pop()

- Step 1 - Check whether the stack is **Empty** (**top** == **NULL**).
- Step 2 - If it is **Empty**, then display "**Stack is Empty!!! Deletion is not possible!!!**" and terminate the function
- Step 3 - If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.
- Step 4 - Then set '**top** = **top** → **next**'.
- Step 5 - Finally, delete '**temp**'. (**free**(**temp**)).

display()

- Step 1 - Check whether the stack is **Empty** (**top** == **NULL**).
- Step 2 - If it is **Empty**, then display '**Stack is Empty!!!**' and terminate the function.
- Step 3 - If it is **Not Empty**, then define a Node pointer '**temp**' and initialize with **top**.
- Step 4 - Display '**temp** → **data** --->' and move it to the next node. Repeat the same until **temp** reaches the first node in the stack. (**temp** → **next** != **NULL**).
- Step 5 - Finally! Display '**temp** → **data** ---> **NULL**.

---3>

Psuedocode for queue using linked list .

Pseudocode:

- Step 1 - Include all the header ıles which are used in the program. And declare all the user deıned functions.
- Step 2 - Deıne a 'Node' structure with two members data and next.
- Step 3 - Deıne two Node pointers 'front' and 'rear' and set both to NULL.
- Step 4 - Implement the main method by displaying a Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

enqueue(value)

- Step 1 - Create a newNode with a given value and set 'newNode → next' to NULL.
- Step 2 - Check whether queue is Empty (rear == NULL)
- Step 3 - If it is Empty then, set front = newNode and rear = newNode.
- Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode

dequeue()

- Step 1 - Check whether the queue is Empty (front == NULL).
- Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
- Step 3 - If it is Not Empty then, deıne a Node pointer 'temp' and set it to 'front'.
- Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

dsiplay()

- Step 1 - Check whether the queue is Empty (front == NULL).
- Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
- Step 3 - If it is Not Empty then, deıne a Node pointer 'temp' and initialize with front.
- Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).
- Step 5 - Finally! Display 'temp → data ---> NULL.

---4>

Psuedocode for singly circular linked list operations: (1) Insert first (2) Insert last (3) Insert after (4) Delete first (4) Delete last (6) Delete after .

Pseudocode:

insertAtFirst(item)

1. Create a new node as

   newnode=(NodeType*)malloc(sizeof(NodeType));
2. if start==NULL then
   set newnode->info=item
   set newnode->next=newnode
   set start=newnode
   set last newnode
   end if
3. else
   set newnode->info=item
   set newnode->next=start
   set start=newnode
   last->next=newnode
   end else
4. End

insertAtLast(value)

1. Create a new node as
   newnode=(NodeType*)malloc(sizeof(NodeType));
2. if start==NULL then
   set newnode->info=item
   set newnode->next=newnode
   set start=newnode
   set last newnode
   end if
3. else
   set newnode->info=item
   set last->next=newnode
   set last=newnode
   set last->next=start
   end else
4. End

insertAt(value,location)

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty then, set head = newNode and newNode → next = head.
- Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.
- Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).
- Step 6 - Every time check whether temp is reached to the last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.
- Step 7 - If temp is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).
- Step 8 - If temp is last node then set temp → next = newNode and newNode → next = head.
- Step 8 - If temp is not last node then set newNode → next = temp → next and temp → next = newNode.

deleteFromFirst()

1. if start==NULL then
   "empty list" and exit
2. else
   set temp=start
   set start=start->next
   print the deleted element=temp->info
   set last->next=start;
   free(temp)
   end else
3. End

deleteFromLast()

1. if start==NULL then
   "empty list" and exit
2. else if start==last
   set temp=start
   print deleted element=temp->info
   free(temp)
   start=last=NULL
3. else
   set temp=start
   while(temp->next!=last)
   set temp=temp->next
   end while

set hold=temp->next
set last=temp
set last->next=start
print the deleted element=hold->info
free(hold)
end else

4. End

deleteAt(value)

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
- Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.
- Step 5 - If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function.
- Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node (temp1 → next == head)
- Step 7 - If list has only one node and that is the node to be deleted then set head = NULL and delete temp1 (free(temp1)).
- Step 8 - If list contains multiple nodes then check whether temp1 is the first node in the list (temp1 == head).
- Step 9 - If temp1 is the first node then set temp2 = head and keep moving temp2 to its next node until temp2 reaches to the last node. Then set head = head → next, temp2 → next = head and delete temp1.
- Step 10 - If temp1 is not first node then check whether it is last node in the list (temp1 → next == head).
- Step 11- If temp1 is last node then set temp2 → next = head and delete temp1 (free(temp1)).
- Step 12 - If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

---5>

Psuedocode for doubly circular linked list operations: (1) Insert first (2) Insert last (3) Delete first (4) Delete last.

Pseudocode:

insertAtFirst(item)

1. Allocate memory for the new node as,
   newnode=(NodeType*)malloc(sizeof(NodeType))
2. Assign value to info field of a new node
   set newnode->info=item
3. set temp=head->next
4. set head->next=newnode
5. set newnode->prev=head
6. set newnode->next=temp
7. set temp->prev=newnode
8. End

insertAtEnd(item)

1. Allocate memory for the new node as,

newnode=(NodeType*)malloc(sizeof(NodeType))

2. Assign value to info field of a new node
   set newnode->info=item
3. set temp=head->prev
4. set temp->next=newnode
5. set newnode->prev=temp
6. set newnode->next=head
7. set head->prev=newnode
8. End

deleteFromFirst(item)

1. if head->next==NULL then
   print “empty list” and exit
2. else
   set temp=head->next;
   set head->next=temp->next
   set temp->next=head
   free(temp)
3. End

```
deleteFromLast(item)
```

1. if head->next==NULL then
   print “empty list” and exit
2. else
   set temp=head->prev;
   set head->left=temp->left
   free(temp)
3. End

Trees

---1>

Algorithm for evaluating an expression using tree.

If t is not null then

If t.value is operand then

Return t.value

A = solve(t.left)

B = solve(t.right)

// calculate applies operator 't.value'

// on A and B, and returns value

Return calculate(A, B, t.value)

Algorithm :

Let t be the syntax tree

If t is not null then

If t.info is operand then

Return t.info

Else

A = solve(t.left)

B = solve(t.right)

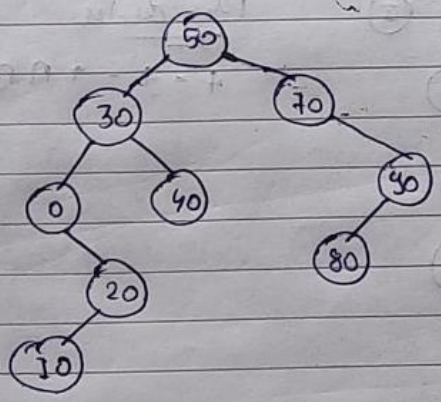
return A operator B

where operator is the info contained in t

20BCP023

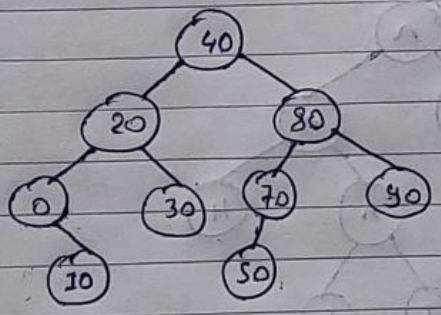
Q. Draw binary search tree and traverse in-order, pre-order and post-order.

1] 50, 30, 0, 20, 70, 90, 80, 10, 40



Inorder : 0, 10, 20, 30, 40, 50, 70, 80, 90
Preorder : 50, 30, 0, 20, 10, 40, 70, 90, 80
Post order : 10, 20, 0, 40, 30, 80, 90, 70, 50

2] 40, 20, 0, 30, 80, 90, 70, 10, 50

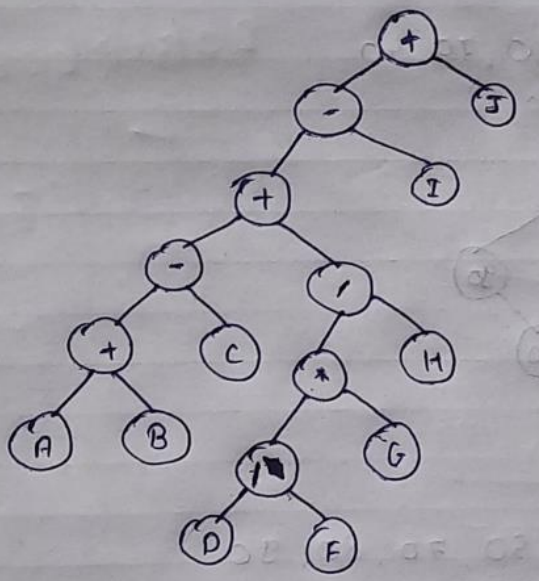


Inorder : 0, 10, 20, 30, 40, 50, 70, 80, 90
Preorder : 40, 20, 0, 10, 30, 80, 70, 50, 90
Post order : 10, 0, 30, 20, 50, 70, 90, 80, 40

20BCP023

Q. Draw Expression trees and traverse Pre-order and post-order

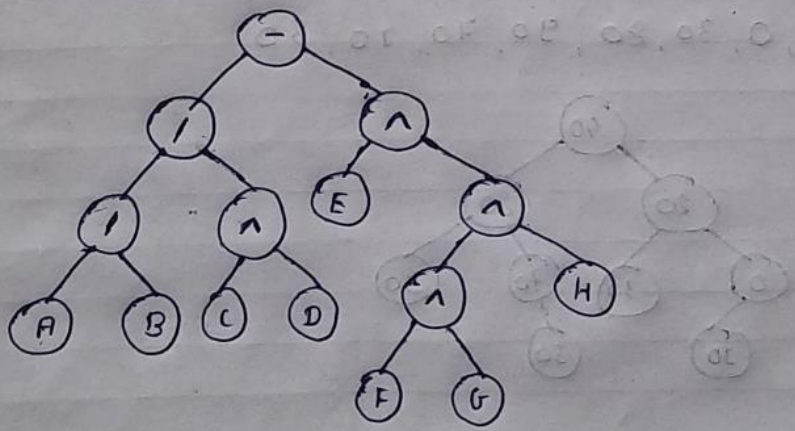
1] A + B - C + D / F * G / H - I + J



Pre order

+ - + - + A B C / * / D F G H I J

2] A / B / C ^ D - E ^ (F ^ G) ^ H

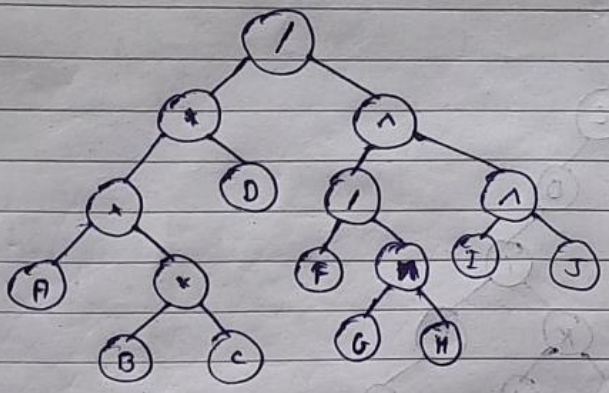


Pre order : - / / A B ^ C D ^ E ^ ^ F G H

Post order : A B / C D ^ / E F G ^ H ^ ^ -

CELEBRO
DATE
PAGE 5

3] A*(B*C)*D/(F/G^H)^I^J 20BCP023

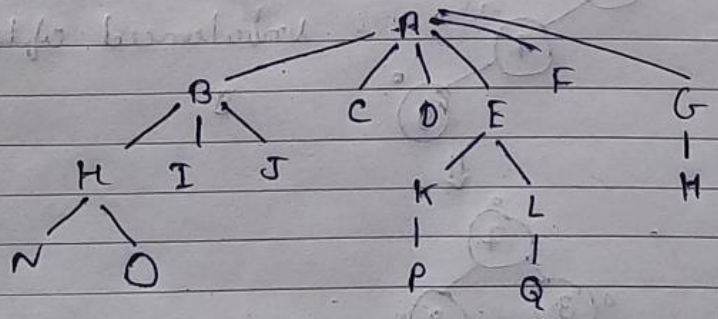


Pre order : /**A*BCD^/F^GH^IJ

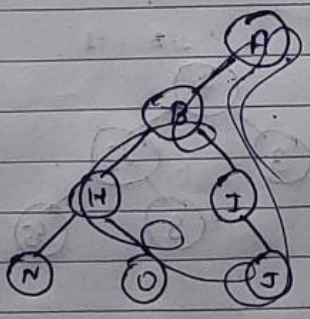
~~Post order : ABC**D*FGH^^~~

Post order : ABC**D*FGH^/IJ^^/

- Convert the following tree to binary tree.

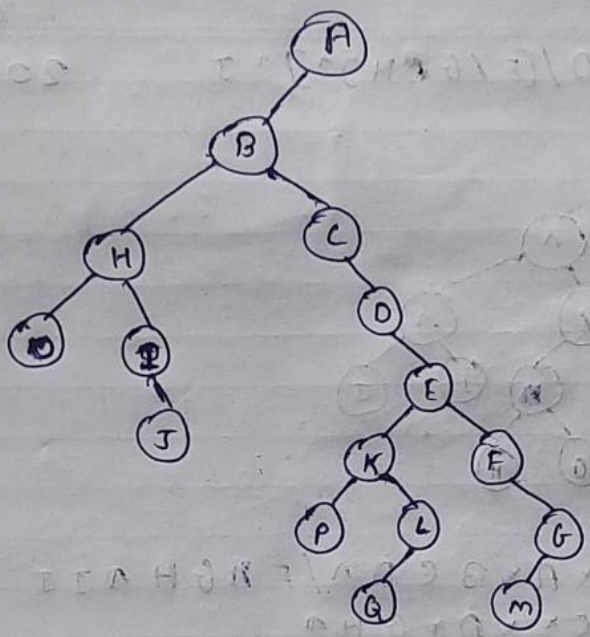


Binary tree



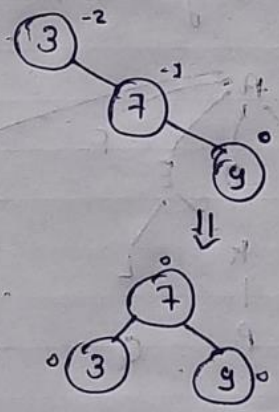
Binary Tree

20BCP023



(i) AVL Tree : 3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 36.

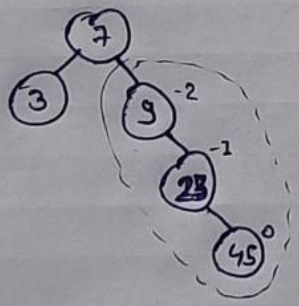
1]



Unbalanced after adding.

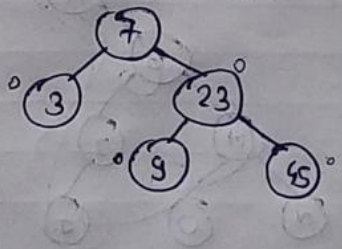
9

2]



=>

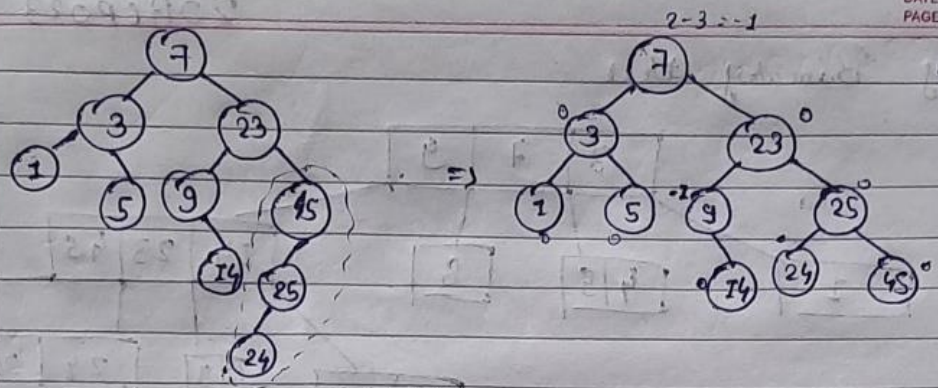
1-2 = -1



Unbalanced after adding 45.

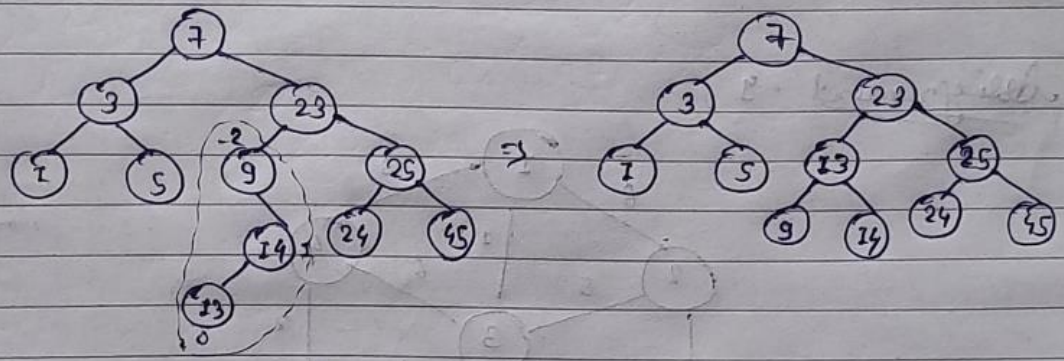
20BCP023

3]



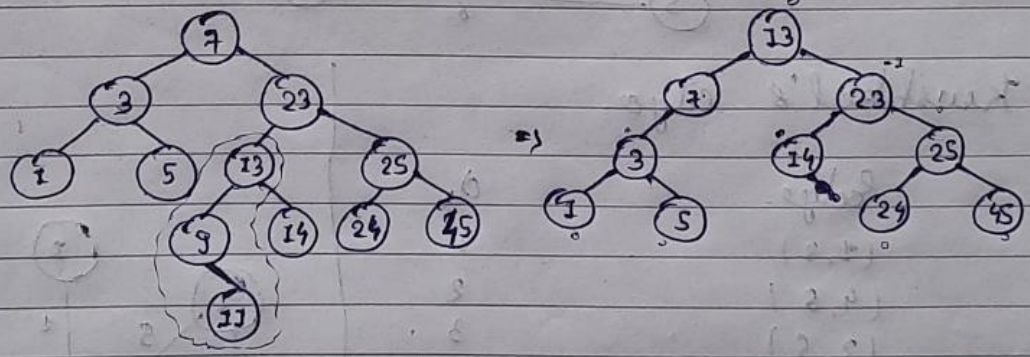
Unbalanced after adding 24

4]



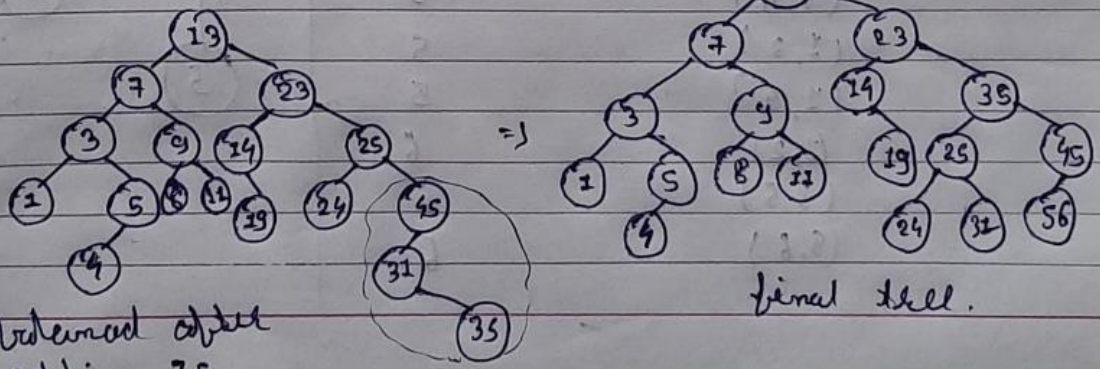
Unbalanced after adding 13.

5]



Unbalanced after adding 11

6

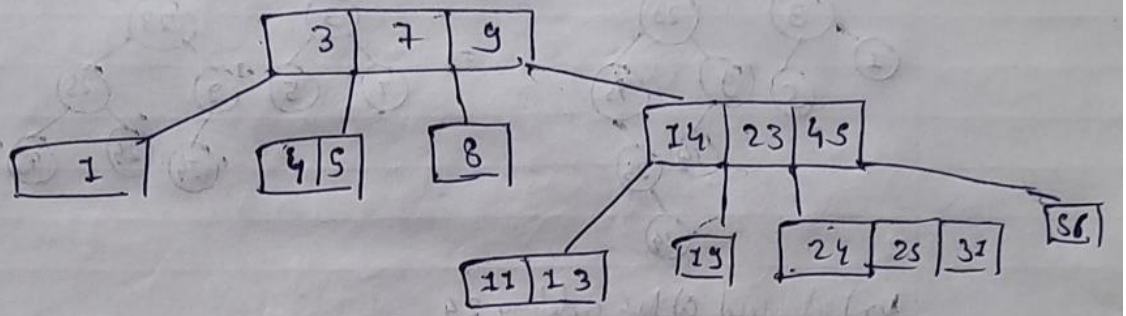


Unbalanced after adding 35

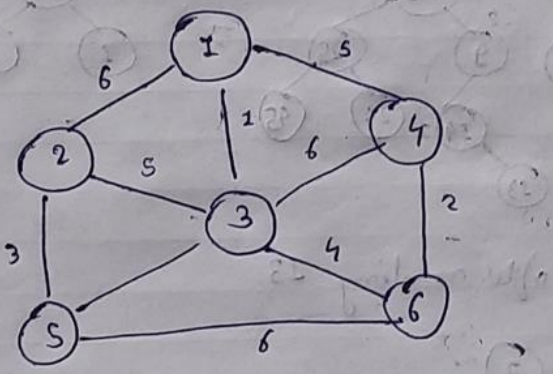
final tree.

20BCP023

* 4- Way Binary tree

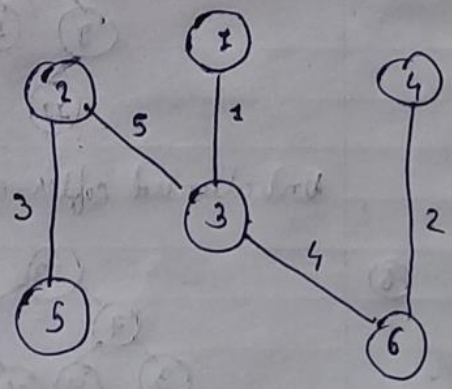


- Assignment - 9



Kruskal's algo.

Edge	d_k
(1,3)	1
(4,6)	2
(2,5)	3
(3,6)	4
(3,4)	5
(1,3)	5
(1,4)	5
(1,2)	6
(3,5)	6
(5,6)	6



2] Prim's Algo. 20BCP023

1]

	K	d_k	P_k
1			
2		3	5
3		6	5
4			
5	T	0	-
6		6	5

②

	K	d_k	P_k
1		6	2
2	T	3	5
3		5	2
4			
5	T	0	-
6		6	5

3]

	K	d_k	P_k
1		1	3
2	T	3	5
3	T	5	2
4		5	3
5	T	0	-
6		4	3

4]

	K	d_k	P_k
1	T	1	3
2	T	3	5
3	T	5	2
4		5	3
5	T	0	-
6		4	3

5]

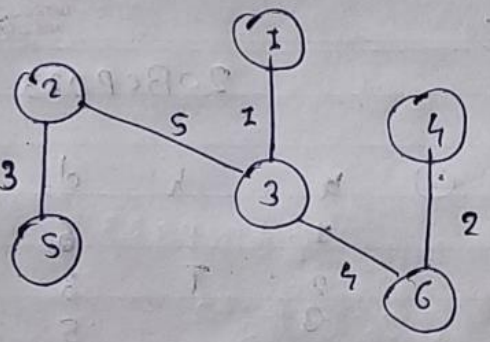
	K	d_k	P_k
1	T	1	3
2	T	3	5
3	T	5	2
4		2	6
5	T	0	-
6	T	4	3

6]

	K	d_k	P_k
1	T	1	3
2	T	3	5
3	T	5	2
4	T	2	6
5	T	0	-
6	T	4	3

	K	d_k	P_k	edge
1	T	1	3	(1,3)
2	T	3	5	(2,5)
3	T	5	2	(3,2)
4	T	2	6	(4,6)
5	T	0	-	
6	T	4	3	(6,3)

20BCPD23



* BFS and DFS of above graphs:

BFS

→ Step 1 : Select 1 vertex as a starting point.

1					

Step 2 : Visit all the adjacent vertex of 1 which are not visited. Insert newly visited vertex in queue and delete 1 from queue.

	2	3	4		

Step 3 : Visit all adjacent vertex of 2 which are not visited (5) Insert newly visited in queue and delete 2 from queue.

		3	4	5	

Step 4 : Repeat same process for 3.

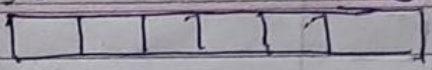
		4	5	6	

Step 5 : Repeat same process for 4.

				5	6

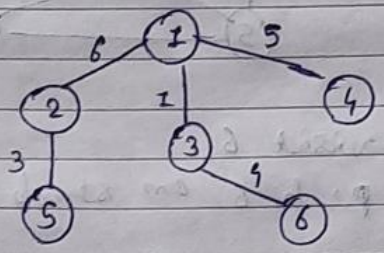
Step 6 : Repeat same for 5.

				6

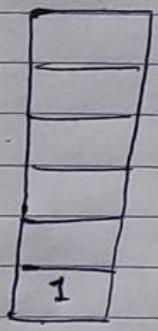
Step : 7 - Repeat of 6. ⇒ 

Now Queue becomes empty, and stop BFS process.

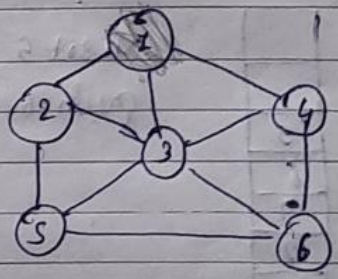
Result of BFS

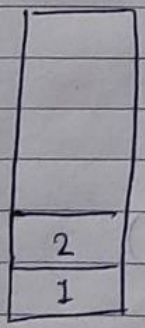


DFS

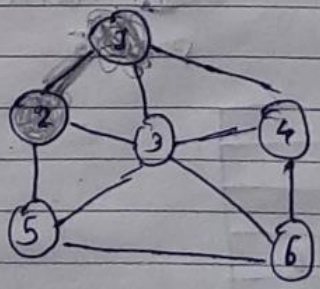
Step 1: 

Select vertex 1
-> Push 1



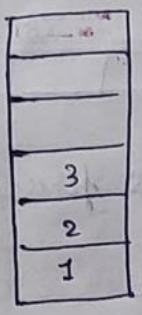
Step 2: 

Push 2 on stack
visit 2

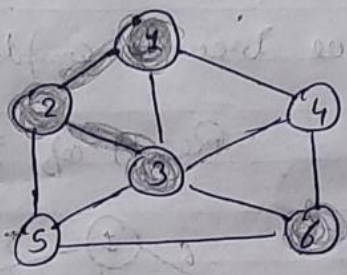


20BCP023

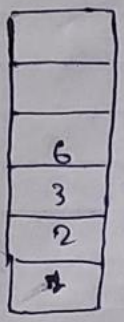
Step 3:



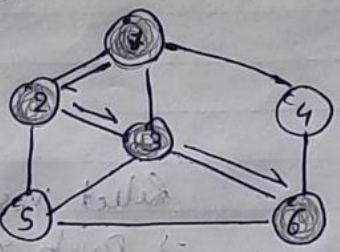
Visit adjacent of 2 (3)
Push 3 on stack.



Step 4:

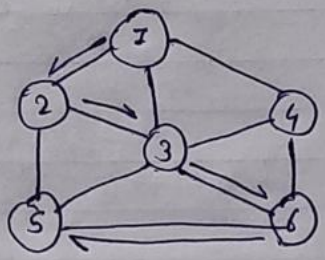


Visit 6
Push 6 on stack

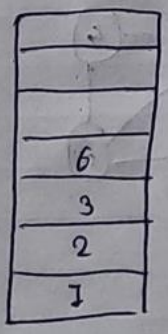


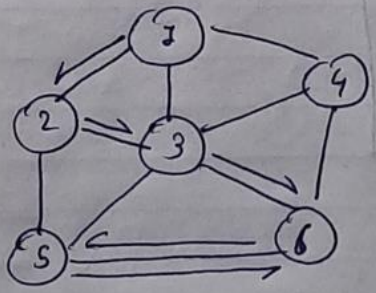
Step 5:

Visit 5
Push 6.



Step 6: No vertex from 5.
Pop 5



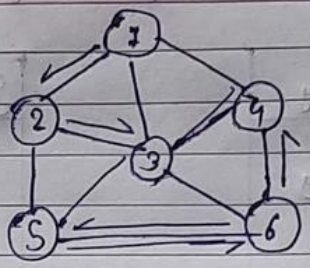


20BCP023

Step 7

4
6
3
2
1

Visit 4
push 4



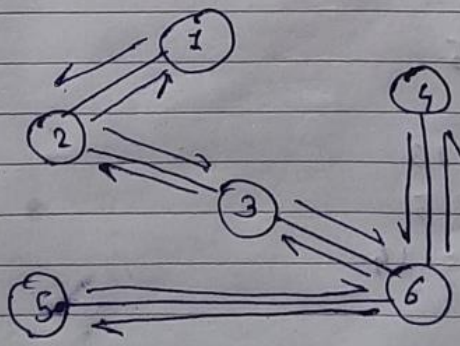
Step-8 No new vertex from 4
pop 4

6
3
2
1

Step 9: pop 6
Step 10 pop 3
Step 11 pop 2
Step 12 pop 1

{ no new vertex }

Final DFS



1] $m = 15$, $h(k) = (2k + 3) \bmod m$, $i = 0 \ldots (m-1) = 0 \ldots 14$

i	key
0	
1	29
2	
3	45
4	150
5	16
6	
7	120
8	10
9	
10	
11	
12	
13	50
14	8

Index	key	Probes
3	45	1
1	29	1
5	16	1
13	50	1
8	25	1
4	150	2
7	120	3
14	8	6
9	10	2

v is location

for 150 = k, v = 3

$k = (v + i^2) \cdot 1 \cdot m$

$150 = (3 + 0) \cdot 1 \cdot 15 = 3$

$150 = (3 + 1) \cdot 1 \cdot 15 = 4$

for 120, v = 3

$120 = (3 + 0) \cdot 1 \cdot 15 = 3$

$120 = (3 + 1) \cdot 1 \cdot 15 = 4$

$120 = (3 + 4) \cdot 1 \cdot 15 = 7$

for k = 10, v = 8

$10 = (8 + 0) \cdot 15 = 8$

$10 = (8 + 1) \cdot 1 \cdot 15 = 9$

for k = 8, v = 4

$8 = (4 + 0) \cdot 1 \cdot 15 = 4$

$8 = (4 + 1) \cdot 1 \cdot 15 = 5$

$8 = (4 + 4) \cdot 1 \cdot 15 = 8$

$8 = (4 + 9) \cdot 1 \cdot 15 = 13$

$8 = (4 + 16) \cdot 1 \cdot 15 = 5$

$8 = (4 + 25) \cdot 1 \cdot 15 = 14$

2] $m = 24$, $h_1(k) = (2k+1)$, $h_2(k) = (3+k)$ v is location.
$v = h_2(k) \cdot v \cdot m$ and $i = 0$ to 23.

i	Key
0	
1	24
2	
3	
4	144
5	122
6	
7	199
8	
9	100
10	50
11	
12	
13	120
14	
15	
16	36
17	
18	
19	
20	
21	10
22	
23	88

Index	Key	Probes
5	122	1
1	24	1
7	99	1
16	36	2
9	100	1
10	50	2
4	144	2
13	120	5
23	88	3
21	10	1

v for 36, v = 1
$v = 15$
$k = (v + v \cdot i) \cdot v \cdot m$
$36 = (1 + 15 \cdot 0) \cdot v \cdot 24 = 1$
$36 = (1 + 15) \cdot v \cdot 24 = 16$

v for 50, v = 5
$v = 5$
$50 = (5 + 5 \cdot 0) \cdot v \cdot 24 = 5$
$50 = (5 + 5) \cdot v \cdot 24 = 10$

v for 120, v = 1
$v = 3$
$120 = (1 + 0) \cdot v \cdot 24 = 1$
$120 = (1 + 3) \cdot v \cdot 24 = 4$
$120 = (1 + 6) \cdot v \cdot 24 = 7$
$120 = (1 + 9) \cdot v \cdot 24 = 10$
$120 = (1 + 12) \cdot v \cdot 24 = 13$

for 144, $v = 1$

$v = 3$

$144 = (1+0) \cdot 1 \cdot 24 = 1$

$144 = (1+3) \cdot 1 \cdot 24 = 4$

for 88, $v = 9$

$v = 19$

$88 = (9+0) \cdot 1 \cdot 24 = 9$

$88 = (9+9) \cdot 1 \cdot 24 = 18$

$88 = (9+38) \cdot 1 \cdot 24 = 23$