Programme	:	B.Tech (ECE|CSE)	Semester	:	FS 2017-18
Course	:	DATA STRUCTURES AND ALGORITHMS	**Code**	:	CSE2003
Faculty	:	Dr.Vetrivelan.P	**Slot**	:	G1

DATA STRUCTURES AND ALGORITHMS

RECORD BOOK

NAME : DHRUV SHEKHAR GARG

REG NO: 16BCE1190

INDEX

DHRUV GARG 16 BCE 1190

EXPERIMENT 1: IMPLEMENTATION OF STACKS AND QUEUES USING ARRAY AND LINKED LIST

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: STACK USING ARRAY

CODE

```
#include<stdio.h>
#include<stdlib.h>
#define stack_size 5
void push (int value);
void pop();
void peek();
int size();
void view();
int stack[stack_size],top=-1;
int main()
{
        int x, data, item;
        printf("\nRepresentation of Stack:");
        printf("\n1. Push, 2. Pop, 3. Peek, 4. Size, 5.View, 6. Exit");
        while(1)
        {
                printf("\n Enter the choice:");
                scanf("%d", &x);
                switch(x)
                {
                        case 1:
                                        printf("\nEnter the element:");
                                        scanf("%d", &data);
                                        push(data);
                                        break;
                        case 2:
```

```
                                        pop();
                                        break;
                                case 3:
                                        peek();
                                        break;
                                case 4:
                                        printf("\n stack size=%d", stack_size);
                                        printf("\n Current stack size=%d", size());
                                        break;
                                case 5:
                                        view();
                                        break;
                                default:
                                        printf("\n End of the program\n");
                                        exit(0);}}
return 0;}
int isfull(){
        extern int stack[], top;
        if(top==stack_size-1)
                return(1);
        else
                return(0);
}
int isempty(){
        extern int stack[], top;
        if(top==-1)
                return(1);
        else
                return(0);
```

```
}
void push(int value)
{
        extern int stack[], top;
        if(isfull())
                printf("\n Stack is full");
        else
        {
                top++;
                stack[top]=value;
        }
}
void pop(){
        int value;
        extern int stack[], top;
        if(isempty())
                printf("\n Stack is empty");
        else
        {
                value=stack[top];
                printf("\n The popped value is %d", value);
                top-- ;
        }
}
void peek()
{
        int item;
        extern int stack[], top;
        if(isempty(1))
```

```
                printf("\n Stack is empty");
        else
        {
                item=stack[top];
                printf("\n The peek of the stack is %d", item);
        }
}
int size()
{
        extern int stack[], top;
        if(isempty())
                return(0);
        else
                return(top+1);
}
void view()
{
        extern int stack[], top;
        int f;
        if(isempty())
                printf("\n Stack is empty");
        else
        {
                printf("\n Content of the stack is .... \n top-->;");
                for(f=top;f>=0;f--)
                        printf("%d-->", stack[f]);
        }
        if(isfull())
                printf("\n Stack is full");}
```

OUTPUT VERIFICATION

7/8/17

STACK USING ARRAY

EXPERIMENT-1(a)

AIM: To study and implement stack using array.

OUTPUT:

```
** REPRESENTATION OF STACK USING ARRAY **

1. Push
2. Pop
3. View
4. Peek
5. Size
6. Exit
Enter a choice : 1
Enter the element you want to push : 10
Enter a choice : 1
Enter the element you want to push : 20
Enter a choice : 1
Enter the element you want to push : 30
Enter a choice : 1
Enter the element you want to push : 40
Enter a choice : 1
Enter the element you want to push : 50
Enter a choice : 1
Stack is full
Enter a choice : 3
Content of the stack is : 50 40 30 20 10
Enter a choice : 2
The popped value is : 50
Enter a choice : 4
The topmost value is : 40
Enter a choice : 5
The size of stack is : 4
```

Enter a choice : 2

The popped value is : 40

Enter a choice : 2

The popped value is : 30

Enter a choice : 2

The popped value is : 20

Enter a choice : 2

The popped value is : 10

Enter a choice : 2

~~The popped value is~~

The stack is empty.

O/P verified

INLAB EXERCISE: QUEUE USING ARRAY

CODE

```
#include<stdio.h>
#define qsize 5
void enqueue(int value);
void dequeue();
void peek();
int size();
void view();
int queue[qsize],front=-1,rear=-1;
void main()
```

```
{
   int choice,data,item;
   printf("\n Representation of linear queue");
   printf("\n1.enqueue,2.dequeue,3.peek,4.size,5.view,6.exit");
   while(1)
   {
      printf("\nEnter the choice");
      scanf("%d",&choice);
      switch(choice)
      {
      case 1:
         printf("\nenter the element");
         scanf("%d",&data);
         enqueue(data);
         break;
      case 2:
         dequeue();
         break;
      case 3:
         peek();
         break;
      case 4:
         printf("\n queue size=%d",qsize);
         printf("\n current queue size=%d",size());
         break;
      case 5:
         view();
         break;
      default:
```

```
            printf("\n end of the programme");
            exit(0);
        }
    }
}
int isfull()
{
    extern int queue[],front,rear;
    if(rear==(qsize-1))
        return(1);
    else
        return(0);
}
int isempty()
{
    extern int queue[],front,rear;
    if(front==-1&&rear==-1)
        return(1);
    else
        return(0);
}
void enqueue(int value)
{
    extern int queue[],front,rear;
    if(isfull())
        printf("\n queue is full!!");
    else
    {
        if(isempty())
```

```
            front=rear=0;
        else
            rear=rear+1;
        queue[rear]=value;
    }
}
void dequeue(){
    int value;
    extern int queue[],front,rear;
    if(isempty())
        printf("\n queue is empty");
    else
    {
        value=queue[front];
        printf("\n The dequeue value is %d",value);
    }
    if(front==rear)
        front=rear=-1;
    else
        front=front+1;
}
void peek(){
    int item;
    extern int queue[],front,rear;
    if(isempty())
        printf("\n queue is empty");
    else{
        item=queue[front];
        printf("\n The peek of the queue is %d",item);
```

```
    }
}
int size(){
    extern int queue[],front,rear;
    if(isempty())
        return (0);
    else
        return(rear-front+1);
}
void view()
{
    extern int queue[],front,rear;
    int f;
    if(isempty())
    {
        printf("\n queue is empty");
    }
    else
    {
        printf("\n content of the queue is.....\n front-->");
        for(f=front; f!=rear+1; f=f+1)
        {
            printf("%d-->",queue[f]);
        }
        printf("rear");
    }
    if(isfull())
        printf("\n queue is full");
}
```

OUTPUT VERIFICATION

QUEUE USING ARRAY EXPERIMENT 1(b)

AIM: To study and implement queue using array.

OUTPUT :

```
** REPRESENTATION OF LINEAR QUEUE USING ARRAY **
1. Enqueue
2. Dequeue
3. Peek
4. Size
5. View
6. Exit

Enter a choice : 1
Enter the element to enqueue : 95
Enter a choice : 1
Enter the element to enqueue : 83
Enter a choice : 1
Enter the element to enqueue : 26
Enter a choice : 1
Enter the element to enqueue : 85
Enter a choice : 1
Enter the element to enqueue : 99
Enter a choice : 1
Enter the element to enqueue : 50
Queue is full.
```

Enter a choice : 4

Queue size = 5

Current queue size = 5

Enter a choice : 5

Content of the queue is : front → 95 83 26 85 99 → rear

Enter a choice : 2

The dequeued value is : 95

Enter a choice : 2

The dequeued value is : 83

Enter a choice : 3

The peek value of the queue is : 26

Enter a choice : 4

Queue size = 5

Current queue size = 3

Enter a choice : 5

Content of the queue is : front --> 26 85 99 --> rear

Enter a choice : 2

The dequeued value is : 26

Enter a choice : 2

The dequeued value is : 85

Enter a choice : 2

The dequeued value is : 99

Enter a choice : 2

Queue is empty.

DHRUV GARG 16 BCE 1190

POST LAB EXERCISE: STACK USING LINKED LIST

CODE

```
#include<stdio.h>
struct Node
{
  int data;
  struct Node *next;
}*top = NULL;

void push(int);
void pop();
void display();

void main()
{
  int choice, value;
  printf("\n:: Stack using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
        case 1: printf("Enter the value to be insert: ");
                scanf("%d", &value);
                push(value);
                break;
        case 2: pop(); break;
```

```
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nWrong selection!!! Please try again!!!\n");
      }
   }
}
void push(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(top == NULL)
      newNode->next = NULL;
   else
      newNode->next = top;
   top = newNode;
   printf("\nInsertion is Success!!!\n");
}
void pop()
{
   if(top == NULL)
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      printf("\nDeleted element: %d", temp->data);
      top = temp->next;
      free(temp);
   }
}
```

```
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
    }
    printf("%d--->NULL",temp->data);
  }
}
```

OUTPUT

```
:: Stack using Linked List ::

****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20

Insertion is Success!!!

****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 30

Insertion is Success!!!

****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
30--->20--->NULL
****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Deleted element: 30
****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
20--->NULL
```

DHRUV GARG 16 BCE 1190

POST LAB EXERCISE: QUEUE USING LINKED LIST

CODE

```
#include<stdio.h>
struct Node
{
  int data;
  struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();
void main()
{
  int choice, value;
  printf("\n:: Queue Implementation using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
        case 1: printf("Enter the value to be insert: ");
                scanf("%d", &value);
                insert(value);
                break;
        case 2: delete(); break;
        case 3: display(); break;
```

```
            case 4: exit(0);
            default: printf("\nWrong selection!!! Please try again!!!\n");
        }
    }
}
void insert(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode -> next = NULL;
    if(front == NULL)
        front = rear = newNode;
    else{
        rear -> next = newNode;
        rear = newNode;
    }
    printf("\nInsertion is Success!!!\n");
}
void delete()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        front = front -> next;
        printf("\nDeleted element: %d\n", temp->data);
        free(temp);
    }
```

```
}
void display()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
    }
    printf("%d--->NULL\n",temp->data);
  }
}
```

OUTPUT

```
:: Queue Implementation using Linked List ::

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20

Insertion is Success!!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 30

Insertion is Success!!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted element: 20

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
30--->NULL

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
```

DHRUV GARG 16 BCE 1190

EXPERIMENT 2: IMPLEMENTATION OF BINARY SEARCH TREE

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: IMPLEMENTATION OF BST

CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct bst
{
 int data;
 struct bst *left,*right;
};
typedef struct bst node;

void create();
node *search(node *,int,node **);
void delete(node *,int);
void display(node *);
void inorder(node *);
void preorder(node *);
void postorder(node *);
node *get_node();
node *root;

void main()
{int ch,val;
 node *New,*tmp,*parent;
 root=NULL;
parent=NULL;
 printf("\nbinary search tree");
```

```
printf("\n1.create\n2.delete\n3.search\n4.display\n5.Exit");
while(1)
        {
printf("\n\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
                {
case 1:create();
                break;
                case 2:printf("\nEnter the value to delete:");
                scanf("%d",&val);
                if(val==root->data)
                printf("\nCannot delete the root");
                else
                delet(root,val);
                break;
                case 3:printf("\n Enter the element to search:");
                scanf("%d",&val);
                tmp=search(root,val,&parent);
                if(tmp!=NULL)
                        {
printf("\nThe element %d is present ",tmp->data);
                        printf("\nThe parent node is: %d",parent->data);
                        }
                printf("\n The element is not present");
                break;
                case 4:if(root==NULL)
                printf("\nThe tree is empty");
              inorder(root);
```

```
                    break;
                    case 5:
                    default:printf("\nEnd of program");
                    exit(0);
            }
 getch();
 }
}

void create()
{
 node *New;
 int val;
 char ans;
 do
            {
            New=get_node();
            if(New==NULL)
                    {
printf("\nMemory not created");
                    return;
                    }
            printf("\nEnter the element to insert:");
            scanf("%d",&val);
            New->data=val;
            if(root==NULL)
            root=New;
            else
            insert(root,New);
```

```
            printf("\nDo u wanna continue(y/n):");
            ans=getche();
            }while((ans=='Y')||(ans=='y'));
printf("\nThe  bst is created");
}

node *get_node()
{
node *temp;
temp=(node*)malloc(sizeof(node));
temp->right=NULL;
temp->left=NULL;
 return(temp);
}

void insert(node *root,node *New)
{
 if(New->data>root->data)
            {
if(root->right==NULL)
                                {
root->right=New;
                                printf("\nElement is inserted to right of %d",root->data);
                                }
                        else
                        insert(root->right,New);
 }
 else
{
```

```
if(New->data<root->data)
                    {
if(root->left==NULL)
                            {
root->left=New;
                            printf("\nElement is inserted to left of %d",root->data);
                            }
                            else
                            insert(root->left,New);
                    }
                    else
                    printf("\nThe values are equal");
}
}
void delete(node *root,int val)
{
 node *temp,*parent,*temp_succ;
 temp=search(root,val,&parent);
 if(temp!=NULL)
 {
if((temp->left!=NULL)&&(temp->right!=NULL))
 {
parent=temp;
          temp_succ=temp;
          while(temp_succ->left!=NULL)
                    {
parent=temp_succ;
                    temp_succ=temp_succ->left;
                     }
```

```
 temp->data=temp_succ->data;
 parent->left=NULL;
 printf("\n The element is deleted");
 return;
}
if(temp->left!=NULL&&temp->right==NULL)
{
if(parent->left==temp)
  parent->left=temp->left;
 else
  parent->right=temp->left;
 temp=NULL;
 free(temp);
 printf("\nThe element is deleted");
 return;
}
if(temp->left==NULL&&temp->right!=NULL)
{
if(parent->left==temp)
          parent->left=temp->right;
         else
          parent->right=temp->right;
         temp=NULL;
          free(temp);
         printf("\nThe element is deleted");
          return;
}
if(temp->left==NULL&&temp->right==NULL)
{
```

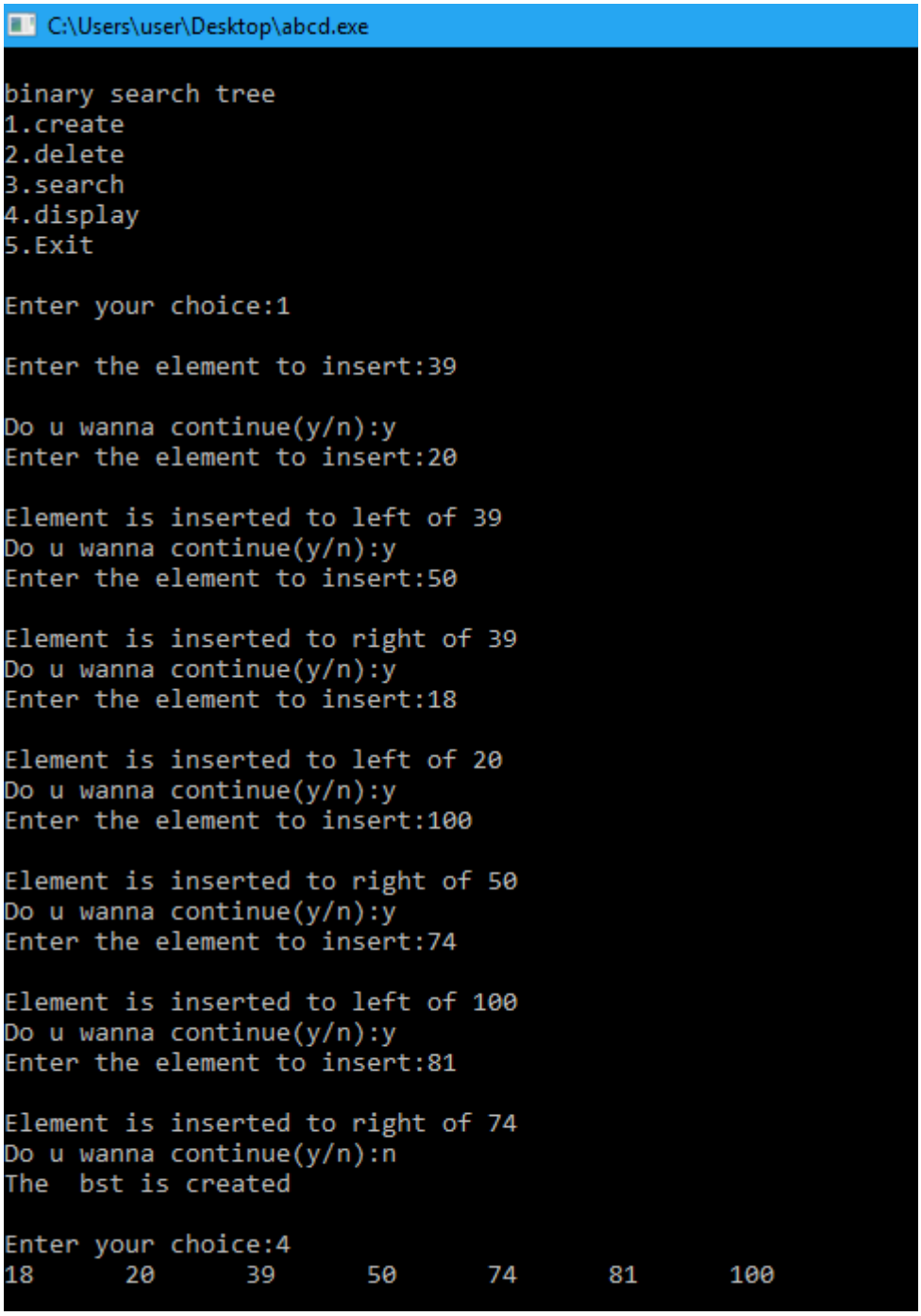
```
if(parent->left==temp)
   parent->left=NULL;
  else
   parent->right=NULL;
  printf("\nThe element is deleted");
  free(temp);
 }
 }
 else
  printf("\nElement is not present in the list");
}

node *search(node *root,int key,node **parent)
{
node *temp;
 temp=root;
 if(root==NULL)
 {printf("\nThe tree is empty");
  return(NULL);
 }
 while(temp!=NULL)
 {if(temp->data==key)
  {printf("\nThe %d element is present",temp->data);
   return(temp);
  }
  *parent=temp;
  if(temp->data>key)
   temp=temp->left;
  else
```

```
  temp=temp->right;
 }
 return(NULL);
}

void display(node *temp)
{
char ans;
printf("\nEnter the choice:");
printf("\ni.Inorder\np.Postorder\nm.Preorder\n");
ans=getche();
switch(ans)
        {
        case 'i':inorder(temp);
        break;
        case 'p':postorder(temp);
        break;
        case 'm':preorder(temp);
        break;
        default: printf("\nEntered wrong choice");
        break;
        }
}
void inorder(node *temp)
{
 if(temp!=NULL)
 {
inorder(temp->left);
printf("%d\t",temp->data);
```

```
inorder(temp->right);
 }
}

void preorder(node *temp)
{
if(temp!=NULL)
        {
printf(" %d\t",temp->data);
        preorder(temp->left);
        preorder(temp->right);
        }
}

void postorder(node *temp)
{
if(temp!=NULL)
        {
postorder(temp->left);
        postorder(temp->right);
        printf(" %d\t",temp->data);
 }
}

/*node*insert(struct node* node, int data)
{
   if (node == NULL) return get_Node(data);

   if (data< node->data)
```

```
            node->left  = insert(node->left,data);
        else if (data> node->data)
            node->right = insert(node->right, data);

        return node;
    }
```

OUTPUT



```
Enter your choice:4
18      20      39      50      74      81      100

Enter your choice:3

 Enter the element to search:81

The 81 element is present
The element 81 is present
The parent node is: 74
```

```
Enter your choice:2

Enter the value to delete:20

The 20 element is present
The element is deleted

Enter your choice:4
18      39      50      74      81      100

Enter your choice:2

Enter the value to delete:100

The 100 element is present
The element is deleted
```

```
Enter your choice:4
18      39      50      74      81
```

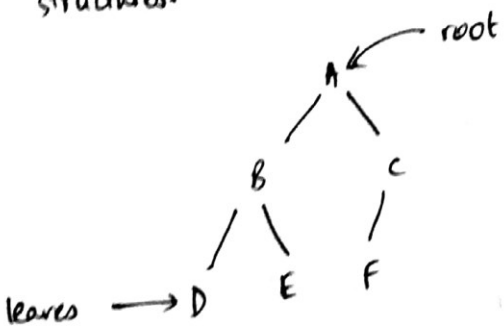
OUTPUT VERIFICATION

Binary Tree

Aim: To study and implement the binary tree.

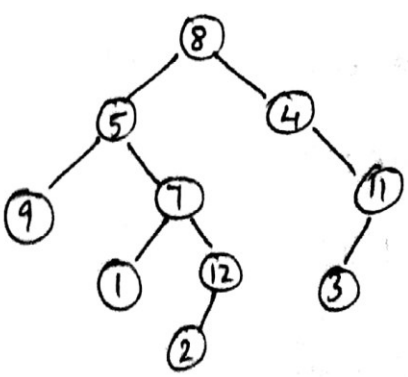
Theory:

Unlike arrays, linked lists, stacks and queues, which are linear data structures, trees are hierarchical data structures.



- **Full binary tree:** A tree is full if every node has 0 or 2 children
- **Complete binary tree:** A tree is complete if all levels are completely filled except possibly the last level and the last level has all keys as left as possible.
- **Perfect binary tree:** All internal nodes have two children and all leaves are at the same level.

Eg of a binary tree:



DHRUV GARG 16 BCE 1190

EXPERIMENT 3: IMPLEMENTATION OF HEAP

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: IMPLEMENTATION OF MAX HEAP

CODE

```
#include <stdio.h>
void max_heapify(int *a,int i,int n)
{
   int j, temp;
   temp = a[i];
   j = 2 * i;
   while (j <= n)
   {
      if (j < n && a[j+1] > a[j])
         j = j + 1;
      if (temp > a[j])
         break;
      else if (temp <= a[j])
      {
         a[j/2] = a[j];
         j = 2 * j;
      }
   }
   a[j/2] = temp;
   return;
}
void build_maxheap(int *a, int n)
{
   int i;
   for(i = n/2; i >= 1; i--)
   {
```

```
        max_heapify(a,i,n);
    }
}
void print_heap(int *a, int n)
{
    int i;
    for (i = 1; i <= n; i++)
    {
        printf(" %d ", a[i]);
    }
    printf("\n");
}
int main()
{
    int n, i, x;
    printf("\n\tMAX HEAP IMPLEMENTATION\n");
    printf("\nEnter no of elements of array : ");
    scanf("%d", &n);
    int a[15] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    for (i = 1; i <= n; i++)
    {
        printf("\nEnter element %d : ", i);
        scanf("%d", &a[i]);
        build_maxheap(a, n);
        print_heap(a, n);
    }
    build_maxheap(a, n);
    printf("\nMax Heap : ");
    print_heap(a, n);}
```

OUTPUT VERIFICATION

28/8/17

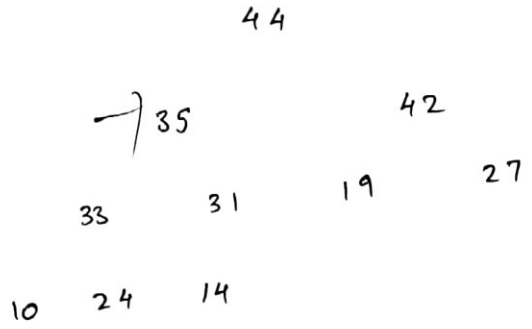
Implementation of heap

AIM: To study and implement heap (max heap)

(i) Construct a max heap from the given numbers

35 33 42 10 14 19 27 44 26 31

Constructed max heap



OUTPUT:

Enter the number of elements in the heap - 10

Enter element 1 - 35

Enter element 2 - 33

Enter element 3 - 42

Enter element 4 - 10

Enter element 5 - 14

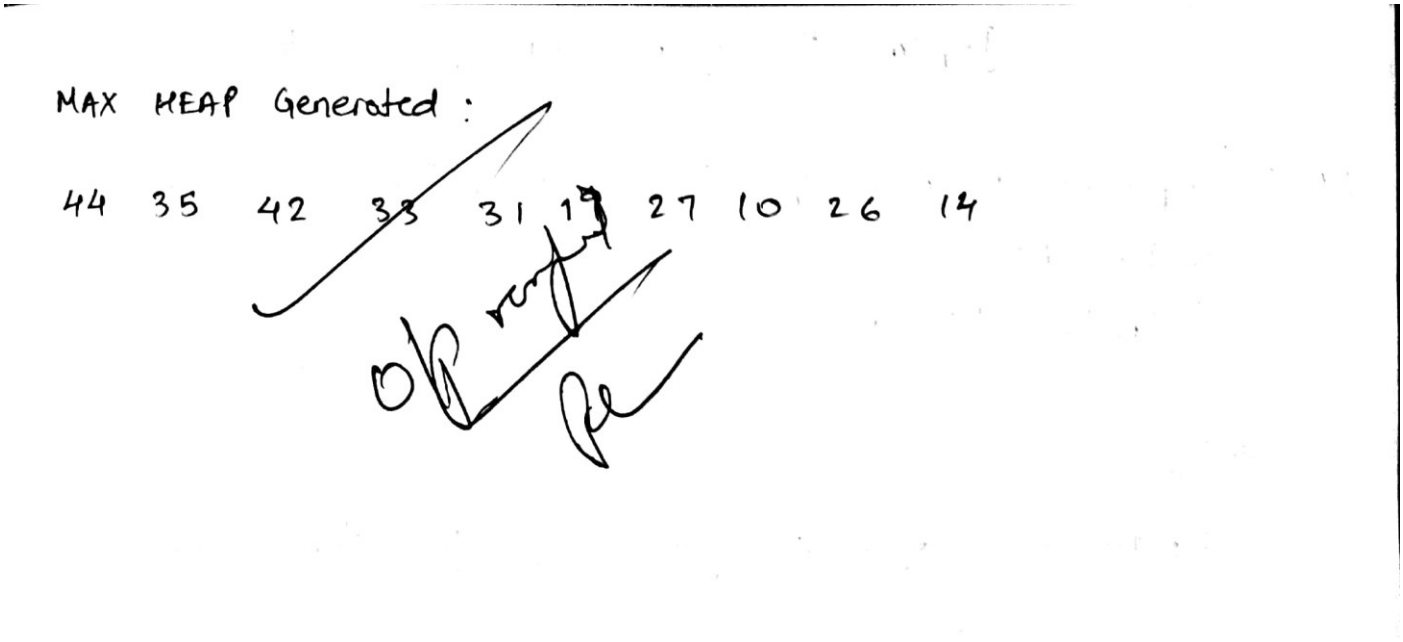
Enter element 6 - 19

Enter element 7 - 27

Enter element 8 - 44

Enter element 9 - 26

Enter element 10 - 31



POST LAB: IMPLEMENTATION OF MIN HEAP

CODE

```
#include <stdio.h>
void min_heapify(int *a,int i,int n)
{
   int j, temp;
   temp = a[i];
   j = 2 * i;
   while (j <= n)
   {
      if (j < n && a[j+1] < a[j])
         j = j + 1;
      if (temp < a[j])
         break;
      else if (temp >= a[j])
      {
         a[j/2] = a[j];
         j = 2 * j;
```

```
        }
    }
    a[j/2] = temp;
   return;
}
void build_minheap(int *a, int n)
{
    int i;
    for(i = n/2; i >= 1; i--)
    {
        min_heapify(a,i,n);
    }
}
void print_heap(int *a, int n)
{
    int i;
    for (i = 1; i <= n; i++)
    {
        printf(" %d ", a[i]);
    }
    printf("\n\n");
}
int main()
{
    int n, i, x;
    printf("\n\tMIN HEAP IMPLEMENTATION\n");
    printf("\nEnter no of elements of array : ");
    scanf("%d", &n); printf("\n");
    int a[15] = {999,999,999,999,999,999,999,999,999,999,999,999,999,999,999};
```

```
    for (i = 1; i <= n; i++)
    {
       printf("Enter element %d : ", i);
       scanf("%d", &a[i]);
       build_minheap(a, n);
       print_heap(a, n);
    }
    build_minheap(a, n);
    printf("\nMin Heap : ");
    print_heap(a, n);
}
```

OUTPUT

C:\Users\Dhruv\Documents\C\DSA_minHeap.exe

```
        MIN HEAP IMPLEMENTATION

Enter no of elements of array : 10

Enter element 1 : 510
 510  999  999  999  999  999  999  999  999  999

Enter element 2 : 120
 120  510  999  999  999  999  999  999  999  999

Enter element 3 : 682
 120  510  682  999  999  999  999  999  999  999

Enter element 4 : 66
 66  120  682  510  999  999  999  999  999  999

Enter element 5 : 326
 66  120  682  510  326  999  999  999  999  999

Enter element 6 : 121
 66  120  121  510  326  682  999  999  999  999

Enter element 7 : 948
 66  120  121  510  326  682  948  999  999  999

Enter element 8 : 73
 66  73  121  120  326  682  948  510  999  999

Enter element 9 : 250
 66  73  121  120  326  682  948  510  250  999

Enter element 10 : 40
 40  66  121  120  73  682  948  510  250  326


Min Heap :  40  66  121  120  73  682  948  510  250  326
```

DHRUV GARG 16 BCE 1190

EXPERIMENT 4: MINIMUM SPANNING TREE

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: PRIM'S ALGORITHM

CODE

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0}
,min,mincost=0,cost[10][10];
void main() {
        printf("\n Enter the number of nodes:");
        scanf("%d",&n);
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++) {
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                   cost[i][j]=999;
        }
        visited[1]=1;
        printf("\n");
        while(ne<n) {
                for (i=1,min=999;i<=n;i++)
                  for (j=1;j<=n;j++)
                   if(cost[i][j]<min)
                    if(visited[i]!=0) {
                          min=cost[i][j];
                          a=u=i;
                          b=v=j;
                }
                if(visited[u]==0 || visited[v]==0) {
```

```
                printf("\n Edge %d:(%d %d) cost:%d \n",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
    }
```

OUTPUT VERIFICATION

Implementation of minimum spanning Tree

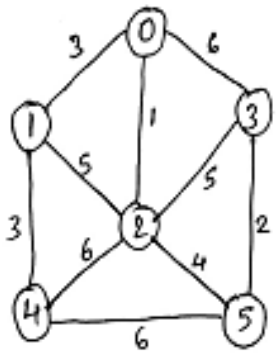
Aim To implement minimum spanning tree using the following:

(i) Prims Algorithm

(ii) Kruskal's Algorithm

Strategy

(a) Get the adjacency matrix of the graph from the user.



Adjacency matrix [6][6]

	0	1	2	3	4	5
0	0	3	1	6	999	999
1	3	0	5	999	3	999
2	1	5	0	5	6	4
3	6	999	5	0	999	2
4	999	3	6	999	0	6
5	999	999	4	2	6	0

(b)

Select

Add the selected edge and vertex to the minimum spanning tree.

Repeat this for covering all vertices.

Output

IMPLEMENTATION OF PRIM'S ALGORITHM

Number of nodes: 6

Input the adjacency matrix:

0	3	1	6	999	999
3	0	5	999	3	999
1	5	0	5	6	4
6	999	5	0	999	2
999	3	6	999	0	6
999	999	4	2	6	0

Edge 1: (1 3) cost: 1

Edge 2: (1 2) cost: 3

Edge 3: (2 5) cost: 3

Edge 4: (3 6) cost: 4

Edge 5: (6 4) cost: 2

Minimum cost: 13

CP Lab Verified

DHRUV GARG 16 BCE 1190

POSTLAB EXERCISE: KRUSKAL'S ALGORITHM

CODE

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
   printf("\n\tIMPLEMENTATION USING
KRUSKAL'S ALGORITHMS\n");
   printf("\nNumber of vertices : ");
   scanf("%d",&n);
   printf("\nEnter adjacency matrix : \n");
   for(i=1; i<=n; i++)
   {
      for(j=1; j<=n; j++)
      {
         scanf("%d",&cost[i][j]);
         if(cost[i][j]==0)
            cost[i][j]=999;
      }
   }
printf("\nEdges of the Minimum Spanning Tree ( MST ) are : \n");
   while(ne < n)
   {
      for(i=1,min=999; i<=n; i++)
```

```
    {
      for(j=1; j <= n; j++)
      {
        if(cost[i][j] < min)
        {
          min=cost[i][j];
          a=u=i;
          b=v=j;
        }
      }
    }
    u=find(u);
    v=find(v);
    if(uni(u,v))
    {
      printf("%d Edge (%d,%d) = %d\n",ne++,a,b,min);
      mincost +=min;
    }
    cost[a][b]=cost[b][a]=999;
  }
  printf("\nMinimum cost = %d\n",mincost);
}
int find(int i)
{
  while(parent[i])
    i=parent[i];
  return i;
}
int uni(int i,int j)
```

```
{
  if(i!=j)
  {
    parent[j]=i;
    return 1;
  }
  return 0;
}
```

OUTPUT

C:\Users\Dhruv\Downloads\kruskal.exe

```
        IMPLEMENTATION USING KRUSKAL'S ALGORITHMS

Number of vertices : 6

Enter adjacency matrix :
0 3 1 6 999 999
3 0 5 999 3 999
1 5 0 5 6 4
6 999 5 0 999 2
999 3 6 999 0 6
999 999 4 2 6 0

Edges of the Minimum Spanning Tree ( MST ) are :
1 Edge (1,3) = 1
2 Edge (4,6) = 2
3 Edge (1,2) = 3
4 Edge (2,5) = 3
5 Edge (3,6) = 4

Minimum cost = 13

Process returned 19 (0x13)   execution time : 78.651 s
Press any key to continue.
```

DHRUV GARG 16 BCE 1190

EXERCISE 5: TREE TRAVERAL USING BFS, DFS AND DJIKSTRA'S ALGORITHM

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: IMPLEMENTATION OF DFS

CODE

```
#include<stdio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
   for(i = 1; i <= n; i++)
      if(a[v][i] && !visited[i])
      q[++r]=i;
   if(f <= r){
         visited[ q[f] ] = 1;
   bfs(q[f++]);
   }
}

void main()
{
   int v;
   printf("\n\n\tBREADTH FIRST SEARCH - USING QUEUE\n");
   printf("\n Enter the number of vertices : ");
   scanf("%d",&n);
   for(i = 1; i <= n; i++){
         q[i] = 0;
   visited[i] = 0;
   }
   printf("\n Enter the graph as adjacency matrix : \n");
   for(i = 1;i <= n; i++)
      for(j = 1;j <= n; j++)
```

```
      scanf("%d",&a[i][j]);
   printf("\n Enter the starting vertex : ");
   scanf("%d",&v);
   bfs(v);
   printf("\n The nodes which are reachable are : \n");

   for(i = 1;i <= n; i++)
      if(visited[i])
      printf("\n -> %d\n",i);
   else
      printf("\n Breadth first search (BFS) is not possible! \n");
}
```

INLAB EXERCISE: IMPLEMENTATION OF DFS

CODE

```
#include<stdio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
   int i;
   reach[v] = 1;
   for(i = 1;i <= n; i++)
      if(a[v][i] && !reach[i])
      {
         printf("\n %d is connected to %d",v,i);
         dfs(i);
      }
   }
```

```
void main() {
   printf("\n\n\tDEPTH FIRST SEARCH - USING STACK\n");
   int i,j,count=0;
   printf("\nEnter number of vertices : ");
   scanf("%d",&n);
   for(i = 1;i <= n; i++){
      reach[i]=0;
   for(j = 1;j <= n;j++)
      a[i][j] = 0;
   }
   printf("\n Enter the adjacency matrix : \n");
   for(i = 1;i <= n;i++)
      for(j = 1;j <= n;j++)
      scanf("%d",&a[i][j]);
   printf("\n The traversal is as follows : \n");
   dfs(1);
   printf("\n");
   for(i=1;i <= n;i++){
         if(reach[i])
         count++;
   }
   if(count == n)
      printf("\n Graph is connected\n\n");
   else
      printf("\n Graph is not connected\n\n");
}
```

OUTPUT VERIFICATION

Graph traversal - BFS and DFS

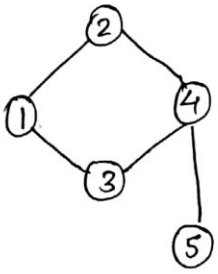
AIM: To implement graph traversal using the following:

(i) Depth first search - DFS (using stack)

(ii) Breadth first search - BFS (Using queue)

Strategy

(a) Get the adjacency matrix from the user



Adjacency matrix:

0	1	1	0	0
1	0	0	1	0
1	0	0	1	0
0	1	1	0	1
0	0	0	1	0

o/p verified 18/9/17

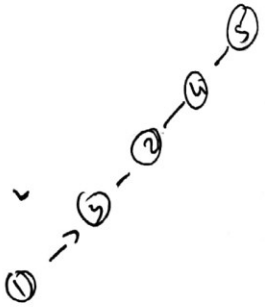
Output

(A) DEPTH FIRST SEARCH - USING STACK

Enter number of vertices : 5

Enter the adjacency matrix :

0	1	1	0	0
1	0	0	1	0
1	0	0	1	0
0	1	1	0	1
0	0	0	1	0



The traversal is as follows:

1 → 2

2 → 4

4 → 3

4 → 5

The graph is connected!

(B) BREADTH FIRST SEARCH – USING QUEUE

Enter the number of vertices : 5

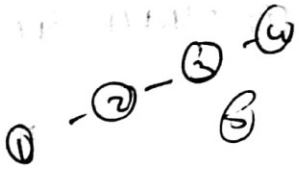
Enter the graph as adjacency matrix:

0 1 1 0 0
1 0 0 1 0
1 0 0 1 0
0 1 1 0 1
0 0 0 1 0

Enter the starting vertex : 1

The nodes that are reachable are:

→ 1
→ 2
→ 3
→ 4
→ 5



O/P verified 18/9/17

POSTLAB EXERCISE: IMPLEMENTATION OF KRUSKAL'S ALGORITHM

CODE

```
#include<stdio.h>

#define INFINITY 9999

#define MAX 10


void dijkstra(int G[MAX][MAX],int n,int startnode)
```

```
{
   int cost[MAX][MAX], distance[MAX], pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,j;
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
         if(G[i][j]==0)
            cost[i][j]=INFINITY;
         else
            cost[i][j]=G[i][j];
   for(i=0; i<n; i++)
   {
         distance[i]=cost[startnode][i];
         pred[i]=startnode;
         visited[i]=0;
  }

   distance[startnode]=0;
   visited[startnode]=1;
   count=1;
 while( count < n-1)
   {
      mindistance=INFINITY;
      for(i=0; i<n; i++)
         if(distance[i]<mindistance&&!visited[i])
         {
            mindistance=distance[i];
            nextnode=i;
         }
      visited[nextnode]=1;
```

```
    for(i=0; i<n; i++)
        if(!visited[i])
            if(mindistance+cost[nextnode][i]<distance[i])
            {
                distance[i]=mindistance+cost[nextnode][i];
                pred[i]=nextnode;
            }
        count++;
    }

    for(i=0; i<n; i++)
        if(i!=startnode)
        {
            printf("\n\nDistance of node %d = %d ",i,distance[i]);
            printf("\nPath = %d ",i);
            j=i;
            do
            {
                j=pred[j];
                printf(" <- %d",j);
                }
                while(j!=startnode);
        }
}

int main()
{
    printf("\n\n\tIMPLEMENTATION OF DJIKSTRA'S ALGORITHM\n\n");
    int G[MAX][MAX],i,j,n,u;
```

```
    printf("Enter no. of vertices : ");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix : \n");

    for(i=0; i<n; i++)
       for(j=0; j<n; j++)
          scanf("%d",&G[i][j]);
    printf("\nEnter the starting node : ");
    scanf("%d",&u);
    dijkstra(G,n,u);
    return 0;
}
```

OUTPUT

C:\Users\Dhruv\Documents\C\DSA_dijkstra.exe

```
        IMPLEMENTATION OF DJIKSTRA'S ALGORITHM
Enter no. of vertices : 9

Enter the adjacency matrix :
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0

Enter the starting node : 0
```

C:\Users\Dhruv\Documents\C\DSA_dijkstra.exe

```
Enter the starting node : 0


Distance of node 1 = 4
Path = 1  <- 0

Distance of node 2 = 12
Path = 2  <- 1 <- 0

Distance of node 3 = 19
Path = 3  <- 2 <- 1 <- 0

Distance of node 4 = 21
Path = 4  <- 5 <- 6 <- 7 <- 0

Distance of node 5 = 11
Path = 5  <- 6 <- 7 <- 0

Distance of node 6 = 9
Path = 6  <- 7 <- 0

Distance of node 7 = 8
Path = 7  <- 0

Distance of node 8 = 14
Path = 8  <- 2 <- 1 <- 0
Process returned 0 (0x0)   execution time : 113.799 s
Press any key to continue.
```

DHRUV GARG 16 BCE 1190

EXPERIMENT 6: IMPLEMENTATION OF SEARCHING AND SORTING TECHNIQUES

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: HEAP SORT

CODE

```
#include <stdio.h>
void main(){
   int heap[10], no, i, j, c, root, temp;
   printf("\n Enter no of elements :");
   scanf("%d", &no);
   printf("\n Enter the nos : ");
   for (i = 0; i < no; i++)
      scanf("%d", &heap[i]);
   for(i = 1; i < no; i++)
   {
      c = i;
   do{
         root = (c - 1) / 2;
   if (heap[root] < heap[c]){
      temp = heap[root];
   heap[root] = heap[c];
   heap[c] = temp;
   }
   c = root;
   } while (c != 0);
   }
   printf("Heap array : ");
   for (i = 0; i < no; i++)
      printf("%d\t ", heap[i]);
   for(j = no - 1; j >= 0; j--){
         temp = heap[0];
   heap[0] = heap[j];
   heap[j] = temp;
   root = 0;
   do{
      c = 2 * root + 1;    /* left node of root element */
   if ((heap[c] < heap[c + 1]) && c < j-1)
      c++;
```

```
        if (heap[root]<heap[c] && c<j){
              temp = heap[root];
        heap[root] = heap[c];
        heap[c] = temp;
        }
        root = c;
        } while (c < j);
        }
        printf("\n The sorted array is : ");
        for (i = 0; i < no; i++)
            printf("\t %d", heap[i]);

}
```

INLAB EXERCISE: INSERTION SORT

CODE

```
#include<stdio.h>


void insertSort(int array[], int n)
{
  int i, curr, j, k;
  for (i = 1; i < n; i++)
  {
     curr = array[i];
     j = i-1;
     while (j >= 0 && array[j] > curr)
     {
        array[j+1] = array[j];
        j = j-1;
     }
     array[j+1] = curr;
     for (k = 0; k < n; k++)
          printf("%d  ",array[k]);
    printf("\n");
  }
}
```

```
int main()
{
                printf("\n\n\tIMPLEMENTING INSERTION SORT ON AN ARRAY\n\n");
        int i, j;
        int n, array[100];

        printf("Number of elements in the array (max size = 100) : ");
        scanf("%d",&n);
        printf("\n");
        printf("Enter UNSORTED array elements : \n");

   insertSort(array,  n);

   printf("\n\nThe SORTED array after implementing insertion sort is : \n");
   for (i = 0; i < n; i++)
        printf("%d\n",array[i]);
                return 0;
}
```

INLAB EXERCISE: LINEAR SEARCH

CODE

```
        #include <stdio.h>

        int main()
        {
                printf("\n\n\tIMPLEMENTING LINEAR SEARCH IN AN ARRAY\n\n");
          int array[100], search = 0;
          int i, n;

          printf("Number of elements in array (max size = 100) : ");
          scanf("%d",&n);

          printf("Enter the elements : \n");

          for (i = 0; i < n; i++)
```

```
    scanf("%d", &array[i]);

  printf("Enter the number to search : \n");
  scanf("%d", &search);

  for (i = 0; i < n; i++)
  {
    if (array[i] == search)
    {
      printf("Found! %d is present at position %d in the array.\n\n\n", search, i+1);
      break;
    }
  }
  if (i == n)
    printf("Sorry! %d is not present in array.\n\n\n", search);

  return 0;
}
```

INLAB EXERCISE: BINARY SEARCH

CODE

```
#include <stdio.h>

int main()
{
  printf("\n\n\tIMPLEMENTING BINARY SEARCH IN AN ARRAY\n\n");
  int i, first = 0, last = 0, middle = 0;
  int n, search, array[100];

  printf("Number of elements in the array (max size = 100) : ");
  scanf("%d",&n);

  printf("Enter elements in SORTED ORDER : \n");

  for (i = 0; i < n; i++)
    scanf("%d",&array[i]);
```

```
  printf("Enter value to search in the sorted array : ");
  scanf("%d", &search);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
      printf("Found!  %d is present at position %d in the array.\n\n\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if (first > last)
    printf("Sorry! %d is not present in the array.\n\n\n", search);

  return 0;
}
```

OUTPUT VERIFICATION

Sorting and Searching

Aim: To implement the following techniques on an array:

(a) Linear search
(b) Binary search
(c) Insertion sort
(d) Heap sort

OUTPUT

(A) LINEAR SEARCH

IMPLEMENTING LINEAR SEARCH IN AN ARRAY

Number of elements in the array (max size = 100): 10
Enter the elements:
11
38
579
314
33
58
69
47
27
22
Enter the number to search: 33
Found! 33 is present at position 5 in the array.

(B) BINARY SEARCH

IMPLEMENTING BINARY SEARCH IN AN ARRAY

Number of elements in the array (max size = 100): 10

Enter the elements in SORTED ORDER:

-20
-18
-7
0
10
14
16
22
38
41

Enter the value to search in the sorted array: -18

Found! -18 found at position 2 in the array.

(c) INSERTION SORT

IMPLEMENTING INSERTION SORT ON AN ARRAY

Number of elements in the array (max size = 100): 10

Enter UNSORTED array elements:

-21 10 -99 0 14 28 -200 99 75 66

The sorted array after implementing insertion sort is:

-200
-99
-21
0
10
14
28
66
75
99

(e) QUICK SORT (Divide and conquer)

3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9

The array is pivoted about its first element Pivot (P) = 3

From the left: see the first element P<X and place overline

From the right: see the first element P>X and place underline.

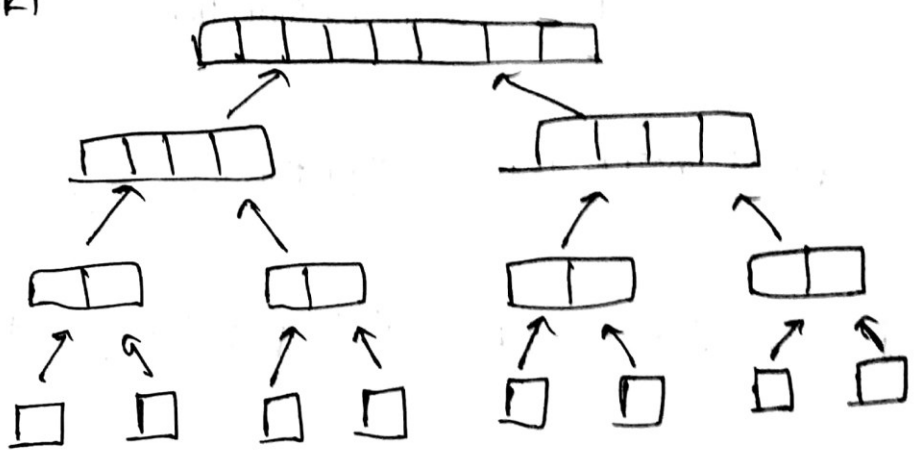
Swap the 2 elements.

After every swap, the overline & underline ~~pairs~~ pointers go back to their extreme positions.

If the 2 pointers cross each other, swap the pivot with overline, and now you get 2 sub arrays.

Time complexity = $n \log n$

(f) MERGE SORT



Time complexity = $n \log n$

DHRUV GARG 16 BCE 1190

POSTLAB EXERCISE: QUICK SORT

CODE

```
#include<stdio.h>


void quickSort(int a[10],int b,int c);


int main()
{
   int arr[20], n, i;
   printf("\n\n\t    IMPLEMENTATION OF QUICK SORT\n\n");
   printf("\n\tSize of the array (max size = 20) : ");
   scanf("%d", &n);


   printf("\n\n\tEnter the elements : \n\n\t");
   for(i = 0; i < n; i++)
      {
         scanf("%d", &arr[i]);
         printf("\n\t");
      }


   quickSort(arr, 0, n-1);


   printf("\n\n\tSORTED ARRAY USING QUICK SORT IS : \n\n\t");
   for(i = 0; i < n; i++)
      printf("  %d  ", arr[i]);
   printf("\n\n");
   return 0;
}
```

```
void quickSort(int arr[10], int first, int last){
   int pivot, j, temp, i;
   if(first < last)
   {
      pivot = first;
      i = first;
      j = last;

      while(i < j)
      {
         while((arr[i] <= arr[pivot]) && (i < last))
            i++;
         while(arr[j] > arr[pivot])
            j--;
         if(i < j)
         {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }
      temp = arr[pivot];
      arr[pivot] = arr[j];
      arr[j] = temp;
      quickSort(arr, first, j-1);
      quickSort(arr, j+1, last);
   }
}
```

OUTPUT

C:\Users\Dhruv\Documents\C\quickSort.exe

```
            IMPLEMENTATION OF QUICK SORT

        Size of the array (max size = 20) : 10

        Enter the elements :
        -67
        84
        12
        59
        231
        -81
        34
        50
        11
        121


        SORTED ARRAY USING QUICK SORT IS :
          -81    -67    11    12    34    50    59    84    121    231

Process returned 0 (0x0)   execution time : 40.025 s
Press any key to continue.
```

DHRUV GARG 16 BCE 1190

POSTLAB EXERCISE: MERGE SORT

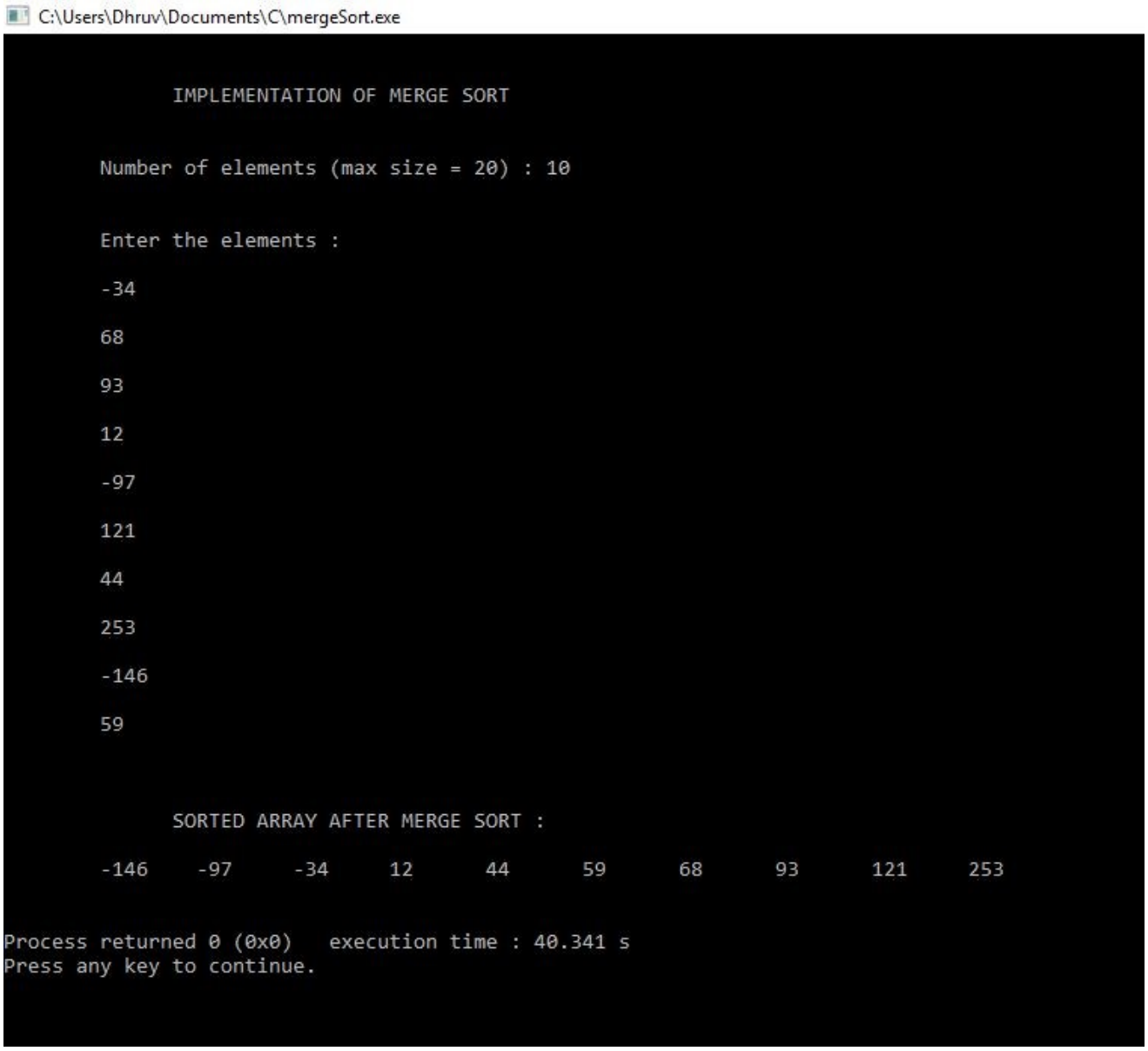
CODE

```
#include<stdio.h>


void merge_sort(int a, int b);
void merge_array(int a, int b, int c, int d);
int arr_sort[20];


int main()
{
   int i, n;


   printf("\n\n\t      IMPLEMENTATION OF MERGE SORT\n\n");
   printf("\n\tNumber of elements (max size = 20) : ");
   scanf("%d", &n);
   printf("\n\n\tEnter the elements : \n\n\t");


   for (i = 0; i < n; i++)
   {
      scanf("%d", &arr_sort[i]);
      printf("\n\t");
   }
   merge_sort(0, n - 1);


   printf("\n\n\t      SORTED ARRAY AFTER MERGE SORT : \n\n");
   for (i = 0; i < n; i++)
   {
      printf("\t%d", arr_sort[i]);
```

```
    }
    printf("\n\n");
    return 0;
}

void merge_sort(int i, int j)
{
    int m;

    if (i < j)
    {
        m = (i + j) / 2;
        merge_sort(i, m);
        merge_sort(m + 1, j);
        merge_array(i, m, m + 1, j);
    }
}

void merge_array(int a, int b, int c, int d)
{
    int t[50];
    int i = a, j = c, k = 0;
    while (i <= b && j <= d)
    {
        if (arr_sort[i] < arr_sort[j])
            t[k++] = arr_sort[i++];
        else
            t[k++] = arr_sort[j++];
    }
```

```
        while (i <= b)
            t[k++] = arr_sort[i++];

        while (j <= d)
            t[k++] = arr_sort[j++];

        for (i = a, j = 0; i <= d; i++, j++)
            arr_sort[i] = t[j];
    }
```

OUTPUT



DHRUV GARG 16 BCE 1190

EXPERIMENT 7: DYNAMIC PROGRAMMING

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: MATRIX CHAIN MULTIPLICATION

CODE

```
#include<stdio.h>
#define INFY 999999999
long int m[20][20];
int s[20][20];
int p[20],i,j,n;

void print_optimal(int i,int j)
{
   if (i == j)
      printf(" A%d ",i);
   else
   {
      printf(" ( ");
      print_optimal(i, s[i][j]);
      print_optimal(s[i][j] + 1, j);
      printf(" ) ");
   }
}

void matmultiply(void)
{
   long int q;
   int k;
   for(i=n; i>0; i--)
   {
      for(j=i; j<=n; j++)
```

```
        {
            if(i==j)
                m[i][j]=0;
            else
            {
                for(k=i; k<j; k++)
                {
                    q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
                    if(q<m[i][j])
                    {
                        m[i][j]=q;
                        s[i][j]=k;
                    }
                }
            }
        }
    }
}

void main()
{
    int k;
    printf("Enter the no. of elements: ");
    scanf("%d",&n);
    for(i=1; i<=n; i++)
        for(j=i+1; j<=n; j++)
        {
            m[i][i]=0;
            m[i][j]=INFY;
```

```
            s[i][j]=0;
        }
    printf("\nEnter the dimensions: \n");
    for(k=0; k<=n; k++)
    {
        printf("P%d: ",k);
        scanf("%d",&p[k]);
    }
    matmultiply();
    printf("\nCost Matrix M:\n");
    for(i=1; i<=n; i++)
        for(j=i; j<=n; j++)
            printf("m[%d][%d]: %ld\n",i,j,m[i][j]);
    printf("\nMatrix S for k values:\n");
    for(i=1; i<=n; i++)
        for(j=i; j<=n; j++)
            printf("m[%d][%d]: %d\n",i,j,s[i][j]);
    i=1,j=n;
    printf("\nMULTIPLICATION SEQUENCE : ");
    print_optimal(i,j);
}
```

OUTPUT VERIFICATION

Matrix chain multiplication

Aim: TO find optimal matrix chain multiplication using dynamic programming

Theory:

$$m[i,j] = m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j]$$

Q. $(A_1)_{10\times100}$, $(A_2)_{100\times5}$, $(A_3)_{5\times50}$

Q. $(A_1)_{30\times35}$, $(A_2)_{35\times15}$, $(A_3)_{15\times5}$, $(A_4)_{5\times10}$, $(A_5)_{10\times20}$, $(A_6)_{20\times25}$

OUTPUT (Answer for Q1)

```
Enter the dimensions : 3
P0: 10
P1: 100
P2: 5
P3: 50

Cost matrix M:
m[1][1]: 0
m[1][2]: 5000
m[1][3]: 7500
m[2][2]: 0
m[2][3]: 25000
m[3][3]: 0

Matrix S for k values:
m[1][1]: 0
m[1][2]: 1
m[1][3]: 2
m[2][2]: 0
m[2][3]: 2
m[3][3]: 0

MULTIPLICATION SEQUENCE: ((A1 A2) A3)
```

O/P verified

DHRUV GARG 16 BCE 1190

EXPERIMENT 8: GREEDY ALGORITHMS

DHRUV GARG 16 BCE 1190

INLAB EXERCISE: KNAPSACK 0/1

CODE

```
#include<stdio.h>

int max(int a, int b)

{

        return (a > b)? a : b;

}


int knapSack(int W, int wt[], int val[], int n)

{

  int i, w;

  int K[n+1][W+1];

  for (i = 0; i <= n; i++)

  {

     for (w = 0; w <= W; w++)

     {

        if (i==0 || w==0)

           K[i][w] = 0;

        else if (wt[i-1] <= w)

            K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);

        else

            K[i][w] = K[i-1][w];

     }

  }

  return K[n][W];

}

int main()

{
```

```
    int i, n, val[20], wt[20], W;

    printf("Enter number of items:");
    scanf("%d", &n);

    printf("Enter value and weight of items:\n");
    for(i = 0;i < n; ++i){
        scanf("%d%d", &val[i], &wt[i]);
    }

    printf("Enter size of knapsack:");
    scanf("%d", &W);

    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

INLAB EXERCISE: FRACTIONAL KNAPSACK

CODE

```
#include <stdio.h>

int n;
int c[10]; /* COST */
int v[10]; /* VALUE */
int W; /* maximum weight */
```

```
void simple_fill()
{
	int cur_w;
	float tot_v;
	int i, maxi;
	int used[10];
	for (i = 0; i < n; ++i)
		used[i] = 0;
	cur_w = W;
	while (cur_w > 0)
	{	/* while there's still room*/
		/* Find the best object */
		maxi = -1;
		for (i = 0; i < n; ++i)
			if ((used[i] == 0) && ((maxi == -1) || ((float)v[i]/c[i] > (float)v[maxi]/c[maxi])))
				maxi = i;
		used[maxi] = 1; /* mark the maxi-th object as used */
		cur_w -= c[maxi]; /* with the object in the bag, I can carry less */
		tot_v += v[maxi];
		if (cur_w >= 0)
			printf("Added object %d (%d$, %dKg) completely in the bag. Space left:
%d.\n", maxi + 1, v[maxi], c[maxi], cur_w);
		else
		{
			printf("Added %d%% (%d$, %dKg) of object %d in the bag.\n", (int)((1 +
(float)cur_w/c[maxi]) * 100), v[maxi], c[maxi], maxi + 1);
			tot_v -= v[maxi];
			tot_v += (1 + (float)cur_w/c[maxi]) * v[maxi];
		}
	}
```

```
        printf("Filled the bag with objects worth %.2f$.\n", tot_v);
}



int main(int argc, char *argv[])
{
        int i;
        printf("W = ");
        scanf("%d", &W);
        printf("n = ");
        scanf("%d", &n);
        printf("Enter the costs");
        for(i=0;i<n;i++)
                scanf("%d", &c[i]);
        printf("Enter the values");
        for(i=0;i<n;i++)
                scanf("%d", &v[i]);
        simple_fill();
        return 0;
}
```

DHRUV GARG 16 BCE 1190

OUTPUT VERIFIED

~~BACKTRACKING APPROACH~~ GREEDY

Aim: To study and implement the following greedy algorithms:

- 0/1 Knapsack
- Fractional Knapsack
- Huffman coding

(A) 0/1 Knapsack

OUTPUT

No. of items: 7

Value and weight of items:

2	10
3	5
5	15
7	7
1	6
4	18
1	3

Max size of Knapsack: 15

The max wt. that can be accomodated in sack: 11

Q/L o/p verified.

(B) Fractional knapsack:

OUTPUT

Added object 4 (7$, 7 Kg) completely in the bag. space left: 8

Added object 2 (3$, 5 kg) completely in the bag. Space left: 3

Added 19% of Object 3 (5$, 15 Kg) in the bag.

Filled the bag with objects worth 11.00$.

Q/L o/p verified.

DHRUV GARG 16 BCE 1190

POSTLAB EXERCISE: HUFFMAN CODING

CODE

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TREE_HT 100

struct MinHeapNode
{
   char data;
   unsigned freq;
   struct MinHeapNode *left, *right;
};

struct MinHeap
{
   unsigned size;
   unsigned capacity;
   struct MinHeapNode **array;
};

struct MinHeapNode* newNode(char data, unsigned freq)
{
   struct MinHeapNode* temp = (struct MinHeapNode*) malloc(sizeof(struct MinHeapNode));
   temp->left = temp->right = NULL;
   temp->data = data;
   temp->freq = freq;
   return temp;
}
```

```
struct MinHeap* createMinHeap(unsigned capacity)
{
   struct MinHeap* minHeap = (struct MinHeap*) malloc(sizeof(struct MinHeap));
   minHeap->size = 0;// current size is 0
   minHeap->capacity = capacity;
   minHeap->array = (struct MinHeapNode**)malloc(minHeap->capacity * sizeof(struct
MinHeapNode*));
   return minHeap;
}

void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b){
   struct MinHeapNode* t = *a;
   *a = *b;
   *b = t;
}

void minHeapify(struct MinHeap* minHeap, int idx)
{
   int smallest = idx;
   int left = 2 * idx + 1;
   int right = 2 * idx + 2;
   if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)
      smallest = left;
   if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)
      smallest = right;
   if (smallest != idx)
   {
      swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
      minHeapify(minHeap, smallest);
```

```
    }
}

int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}

struct MinHeapNode* extractMin(struct MinHeap* minHeap){
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* minHeapNode)
{
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i - 1)/2]->freq){
          minHeap->array[i] = minHeap->array[(i - 1)/2];
    i = (i - 1)/2;
    }
    minHeap->array[i] = minHeapNode;
}

void buildMinHeap(struct MinHeap* minHeap){
    int n = minHeap->size - 1;
```

```
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

void printArr(int arr[], int n){
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}

int isLeaf(struct MinHeapNode* root){
    return !(root->left) && !(root->right) ;
}

struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size){
    struct MinHeap* minHeap = createMinHeap(size);
    int i;
    for(i=0;i<size;++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size){
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);
```

```
    while (!isSizeOne(minHeap)){
            left = extractMin(minHeap);
             right = extractMin(minHeap);
    top = newNode('$', left->freq + right->freq);
    top->left = left;
    top->right = right;
    insertMinHeap(minHeap, top);
}
return extractMin(minHeap);
}

void printCodes(struct MinHeapNode* root, int arr[], int top){
    if (root->left){
          arr[top] = 0;
    printCodes(root->left, arr, top + 1);
}

if (root->right)
    {
       arr[top] = 1;
printCodes(root->right, arr, top + 1);
}
if(isLeaf(root))
    {
       printf("%c: ", root->data);
    printArr(arr, top);
}
}
```

```
void HuffmanCodes(char data[], int freq[], int size){
    struct MinHeapNode* root = buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);
}

int main()
{
    char arr[] = {'a', 'c', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'y'};
    int freq[] = {2,4,7,1,1,2,3,4,2,4,6,1,1,2,5,2,1,1,1};
    int size = sizeof(arr)/sizeof(arr[0]);
    HuffmanCodes(arr, freq, size);
    return 0;
}
```

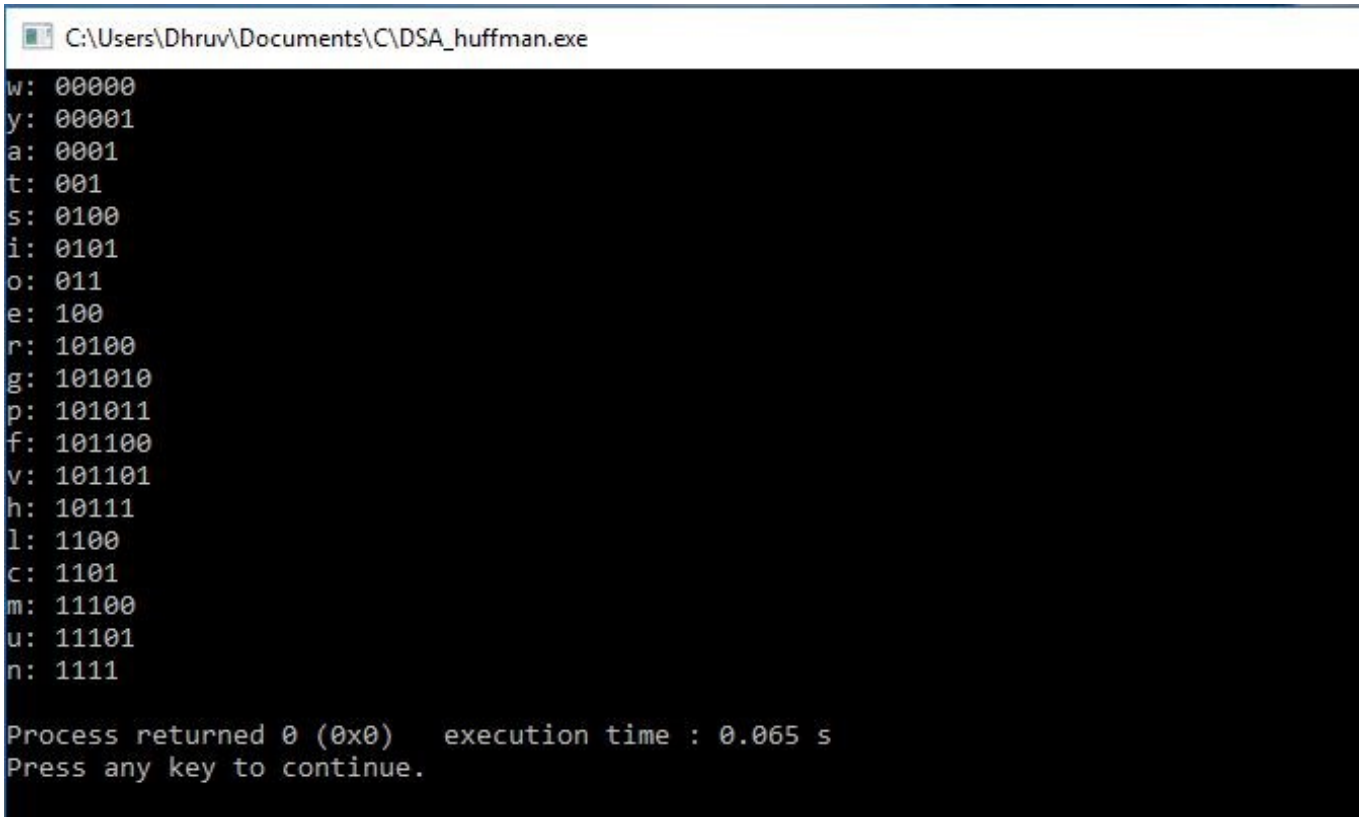
OUTPUT

The above letter and corresponding frequency array has been taken from the following sentence :

“welcome to vellore institute of technology chennai campus”

Thus the corresponding letter frequencies are :

a – 2, c – 4, e – 7, f – 1, g – 1, h – 2, i – 3, l – 4, m – 2, n – 4, o – 6, p – 1, r – 1, s – 2, t – 5, u – 2, v – 1, w – 1, y – 1


C:\Users\Dhruv\Documents\C\DSA_huffman.exe
w: 00000
y: 00001
a: 0001
t: 001
s: 0100
i: 0101
o: 011
e: 100
r: 10100
g: 101010
p: 101011
f: 101100
v: 101101
h: 10111
l: 1100
c: 1101
m: 11100
u: 11101
n: 1111
Process returned 0 (0x0) execution time : 0.065 s
Press any key to continue.

DHRUV GARG 16 BCE 1190

EXPERIMENT 9: BACKTRACKING ALGORITHM

DHRUV GARG 16 BCE 1190

VIRTUAL LAB EXERCISE: N QUEEN PROBLEM

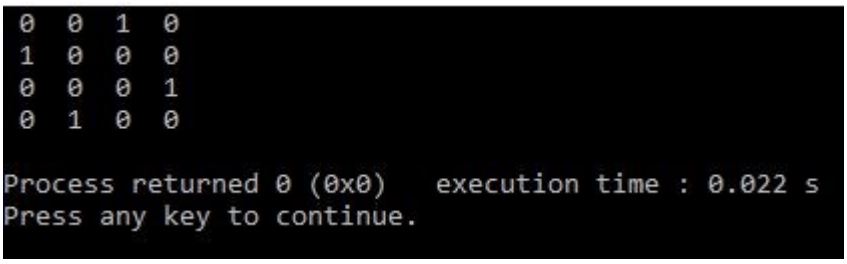
CODE

```
#define N 4
#include<stdio.h>
#include<stdbool.h>
void printSolution(int board[N][N])
{
   int i, j;
   for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++)
      printf(" %d ", board[i][j]);
   printf("\n");
   }
}
bool isSafe(int board[N][N], int row, int col) {
   int i, j;
   for (i = 0; i < col; i++)
      if (board[row][i])
      return false;
   for (i=row, j=col; i>=0 && j>=0; i--, j--)
      if (board[i][j])
      return false;
   for (i=row, j=col; j>=0 && i<N; i++, j--)
      if (board[i][j])
      return false;
   return true;
}
bool solveNQUtil(int board[N][N], int col) {
```

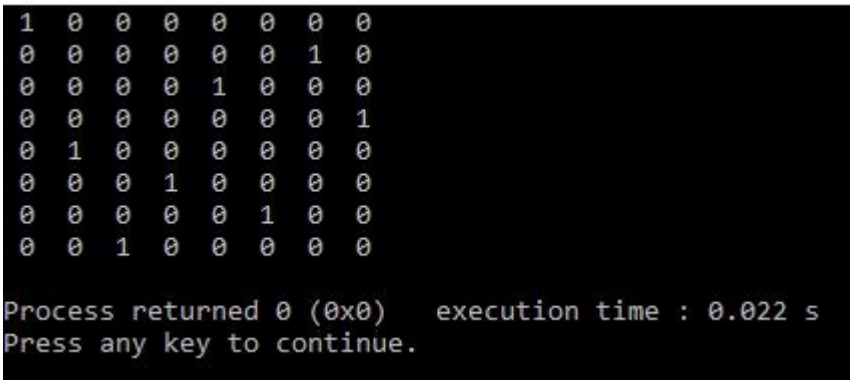
```
    int i;
    if (col >= N)
        return true;
        for (i = 0; i < N; i++) {
        if ( isSafe(board, i, col) ) {
            board[i][col] = 1;
            if ( solveNQUtil(board, col + 1) )
        return true;
    board[i][col] = 0; // BACKTRACK
    }
    }
    return false;
}

bool solveNQ() {
    int board[N][N] = { {0, 0, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}};
    if( solveNQUtil(board, 0) == false ) {
        printf("Solution does not exist");
    return false;
    }
    printSolution(board);
    return true;
}
int main() {
    solveNQ();
    return 0;}
```

OUTPUT

FOUR QUEEN



EIGHT QUEEN



VIRTUAL LAB EXERCISE: SUM OF SUBSETS

CODE

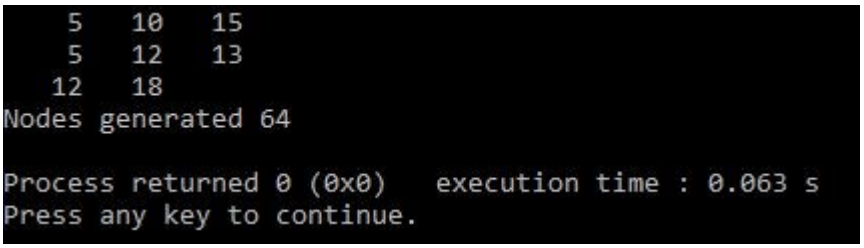
```
#include <stdio.h>

#include <stdlib.h>

#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))

static int total_nodes;

void printSubset(int A[], int size)

{

   int i;
```

```
   for(i = 0; i < size; i++)
   {
      printf("%*d", 5, A[i]);
   }
   printf("\n");
}
void subset_sum(int s[], int t[],int s_size, int t_size,int sum, int ite,int const target_sum)
{
   total_nodes++;
   if( target_sum == sum ) {
      printSubset(t, t_size);
      subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
      return;
   }
   else
   {
      int i;
      for(i = ite; i < s_size; i++ ) {
         t[t_size] = s[i];
         subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
      }
   }
}
void generateSubsets(int s[], int size, int target_sum) {
   int *tuplet_vector = (int *)malloc(size * sizeof(int));
   subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
   free(tuplet_vector);
}
int main() {
```

```
    int weights[] = {5, 10, 12, 13, 15, 18};
    int size = ARRAYSIZE(weights);
    generateSubsets(weights, size, 30);
    printf("Nodes generated %d\n", total_nodes);
    return 0;
}
```

OUTPUT

weights = {5, 10, 12, 13, 15, 18}, required sum = 30



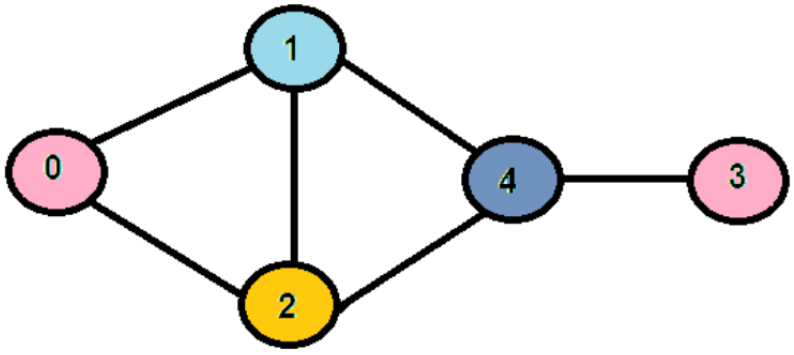
VIRTUAL LAB EXERCISE: GRAPH COLOURING

CODE

```
#include<stdio.h>
int G[50][50],x[50];  //G:adjacency matrix,x:colors
void next_color(int k){
  int i,j;
  x[k]=1;  //coloring vertex with color1
  for(i=0;i<k;i++){ //checking all k-1 vertices-backtracking
    if(G[i][k]!=0 && x[k]==x[i])  //if connected and has same color
      x[k]=x[i]+1;  //assign higher color than x[i]   }}
int main(){
```

```
int n,e,i,j,k,l;
printf("Enter no. of vertices : ");
scanf("%d",&n);
printf("Enter no. of edges : ");
scanf("%d",&e);
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    G[i][j]=0;
printf("Enter indexes where value is 1-->\n");
for(i=0;i<e;i++){
  scanf("%d %d",&k,&l);
  G[k][l]=1;
  G[l][k]=1;
 }
for(i=0;i<n;i++)
  next_color(i);  //coloring each vertex
printf("Colors of vertices -->\n");
for(i=0;i<n;i++)  //displaying color of each vertex
  printf("Vertex[%d] : %d\n",i+1,x[i]);
return 0;
}
```

OUTPUT



```
Enter no. of vertices : 5
Enter no. of edges : 6
Enter indexes where value is 1-->
0 1
0 2
1 2
1 4
2 4
4 3
Colors of vertices -->
Vertex[1] : 1
Vertex[2] : 2
Vertex[3] : 3
Vertex[4] : 1
Vertex[5] : 2
```

VIRTUAL LAB EXERCISE: HAMILTONIAN CYCLE

CODE

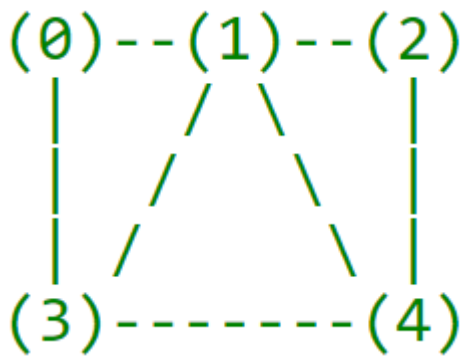
```
#include<stdio.h>

#define V 5

#include<stdbool.h>

#include <malloc.h>

void printSolution(int path[]);

bool isSafe(int v, bool graph[V][V], int path[], int pos)

{

   int i;

   if (graph [ path[pos-1] ][ v ] == 0)

      return false;

   for (i = 0; i < pos; i++)

      if (path[i] == v)

         return false;

   return true;

}

bool hamCycleUtil(bool graph[V][V], int path[], int pos)

{
```

```
    int v;
    if (pos == V)
    {
       if ( graph[ path[pos-1] ][ path[0] ] == 1 )
          return true;
       else
         return false;
    }
    for (v = 1; v < V; v++)
    {
       if (isSafe(v, graph, path, pos))
       {
          path[pos] = v;
          if (hamCycleUtil (graph, path, pos+1) == true)
             return true;
          path[pos] = -1;
       }}
    return false;
}bool hamCycle(bool graph[V][V])
{
    int i;
    int *path =  malloc(sizeof(int[V]));
    for (i = 0; i < V; i++)
       path[i] = -1;
    path[0] = 0;
    if ( hamCycleUtil(graph, path, 1) == false )
    {
       printf("\nSolution does not exist");
       return false;
```

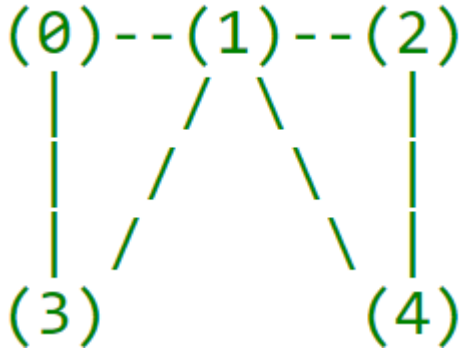
```
    }
    printSolution(path);
    return true;
}
void printSolution(int path[])
{
    int i;
    printf ("Solution Exists:"
            " Following is one Hamiltonian Cycle \n");
    for (i = 0; i < V; i++)
       printf(" %d ", path[i]);    printf(" %d ", path[0]);
    printf("\n");
}
int main(){
   bool graph1[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 1},
                        {0, 1, 1, 1, 0},
                       };
    hamCycle(graph1);
    bool graph2[V][V] = {{0, 1, 0, 1, 0},
                        {1, 0, 1, 1, 1},
                        {0, 1, 0, 0, 1},
                        {1, 1, 0, 0, 0},
                        {0, 1, 1, 0, 0},
                       };
    hamCycle(graph2);
    return 0;}
```

OUTPUT

Example 1 Hamiltonian Graph



Example 2 Non-Hamiltonian Graph



RESULT

```
Solution Exists: Following is one Hamiltonian Cycle
 0  1  2  4  3  0

Solution does not exist
```