1. Write a program to read 'N' numbers of elements into an array and also perform the following operation on an array

a. Add an element at the beginning of an array
b. Insert an element at given index of array
c. Update an element using a values and index
d. Delete an existing element

```
#include<stdio.h>
#include<stdlib.h>
int a[10],n,i,ch,ind=0;
void read_array();
void display();
void Update();
void insert_beg();
void insert();
void delete();
int main()
{
   while(1)
   {
      printf("1. Read the array\n");
      printf("2. Insert element into the array\n");
      printf("3. Display the array\n");
      printf("4. Insert element at beginning\n");
      printf("5. Update  the array\n");
      printf("6. Delete element in array\n");
      printf("7. exit\n");
      printf("Enter ur choice\n");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:read_array();break;
         case 2:insert(); break;
         case 3:display();break;
         case 4:insert_beg();break;
         case 5:Update();break;
         case 6:delete();break;
         case 7: exit(0);
      }

   }
   return 0;
}
void read_array()
{
   printf("enter the size of array\n");
   scanf("%d",&n);
   printf("enter the elements\n");
   for(i=0;i<n;i++){
      scanf("%d",&a[i]);
      ind++;
   }
}
void display(){ printf("The elements in array\n");
   for(i=0;i<ind;i++)
   printf("%d ",a[i]);
   printf("\n");
```

```
}
void Update(){
   int index,value;
   printf("enter the index and value\n");
   scanf("%d%d",&index,&value);
   a[index]=value;
   display();
}
void insert_beg(){
   int value;
   printf("Enter the element - \n");
   scanf("%d",&value);


   for(i = ind-1;i>=0;i--){
      a[i+1] = a[i];
   }
   a[0] = value;
   ind++;
   display();
}
void insert(){
   int index,value;
   printf("Enter the value and index\n ");
   scanf("%d%d",&value,&index);
   for(int i=ind;i>index;i--)
   {
      a[i]=a[i-1];
   }
   a[index]=value;
   display();
}
void delete(){
   int value;
   printf("Enter the element - \n");
   scanf("%d",&value);
   int index = -1;

   for(int i=0;i<10;i++){
      if(a[i] == value){
         index = i;
         break;
      }
   }
   if(index == -1){
      printf("Element Not found");
      return;
   }

   for(i=index+1;i<ind;i++){
      a[i-1] = a[i];
   }

   ind--;
   display();
}
```

2. Write Program to implement Single Linked List with insertion, deletion and traversal operations

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *ptr;
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
int main ()
{   while(1)
   {
      printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n
5.Delete from last\n6.Delete node after specified location\n7.Search for an element\n8.Show\n9.Exit\n");
      int ch;
      printf("enter ur choice ");
      scanf("%d",&ch);

      switch (ch)
      {
         case 1: beginsert (); break;
         case 2: lastinsert ();break;
         case 3: randominsert();break;
         case 4: begin_delete();break;
         case 5: last_delete();break;
         case 6: random_delete();break;
         case 7: search(); break;

         case 8: display();break;
         case 9: exit(0);
      }
   }
   return 0;
}
void beginsert()
{
   int item;
   ptr = (struct node *) malloc(sizeof(struct node *));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
      printf("\nEnter value\n");
      scanf("%d",&item);
      ptr->data = item;
      ptr->next = head;
```

```
      head = ptr;
      display();
   }

}
void lastinsert()
{
   struct node *temp;
   int item;
   ptr = (struct node*)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
      printf("\nEnter value?\n");
      scanf("%d",&item);
      ptr->data = item;
      if(head == NULL)
      {
         ptr -> next = NULL;
         head = ptr;
         display();
      }
      else
      {
         temp = head;
         while (temp -> next != NULL)
         {
            temp = temp -> next;
         }
         temp->next = ptr;
         ptr->next = NULL;
         display();
      }
   }
}
void randominsert()
{
   int i,loc,item;
   struct node *ptr, *temp;
   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
      printf("\nEnter element value ");
      scanf("%d",&item);
      ptr->data = item;
      printf("\nEnter the location  which you want to insert ");
      scanf("\n%d",&loc);
      temp=head;
      for(i=0;i<loc-1;i++)
      {
```

```
            temp = temp->next;
            if(temp == NULL)
            {
               printf("\ncan't insert\n");
               return;
            }

         }
         ptr ->next = temp ->next;
         temp ->next = ptr;
         display();
      }
   }
   void begin_delete()
   {
      if(head == NULL)
      {
         printf("\nList is empty\n");
      }
      else
      {
         ptr = head;
         head = ptr->next;
         free(ptr);
         printf("\nNode deleted from the begining ...\n");
         display();
      }
   }
   void last_delete()
   {
      struct node *ptr1;
      if(head == NULL)
      {
         printf("\nlist is empty");
      }
      else if(head -> next == NULL)
      {
         head = NULL;
         free(head);
         printf("\nOnly node of the list deleted ...\n");
         display();
      }

      else
      {
         ptr = head;
         while(ptr->next != NULL)
         {
            ptr1 = ptr;
            ptr = ptr ->next;
         }
         ptr1->next = NULL;
         free(ptr);
         printf("\nDeleted Node from the last ...\n");
         display();
      }
   }
```

```
void random_delete()
{
   struct node *ptr1;
   int loc,i;
   printf("\n Enter the location of the node after  you want to perform deletion \n");
   scanf("%d",&loc);
   ptr=head;
   for(i=0;i<=loc;i++)
   {
      ptr1 = ptr;
      ptr = ptr->next;

      if(ptr == NULL)
      {
         printf("\nCan't delete");
         return;
      }
   }
   ptr1 ->next = ptr ->next;
   free(ptr);
   printf("\nDeleted node %d ",loc+1);
   display();
}
void search()
{
   struct node *ptr;
   int item,i=0,flag=-1;
   ptr = head;
   if(ptr == NULL)
   {
      printf("\nEmpty List\n");
   }
   else
   {
      printf("\nEnter item to search?\n");
      scanf("%d",&item);
      while (ptr!=NULL)
      {
         if(ptr->data == item)
         {
            printf("item found at location %d ",i);
            flag=0;
         }
         i++;
         ptr = ptr -> next;
      }
      if(flag==-1)
      {
         printf("Item not found\n");
      }
   }
}
void display()
{
   struct node *ptr;
   ptr = head;
   if(ptr == NULL)
```

```
    {
        printf("Nothing to print");
    }
    else
    {
        while (ptr!=NULL)
        {
            printf("%d->",ptr->data);
            ptr = ptr -> next;
        }
        printf("NULL\n");
    }
}
```

3. Write Program to implement Circular doubly Linked List with insertion, deletion and traversal operations

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node *prev;
    struct Node *next;
};
struct CircularDoublyLinkedList{
    struct Node *head;
    struct Node *tail;
};
void insertAtBegin(struct CircularDoublyLinkedList *cdll,int data){
    struct Node *node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    if(!cdll->head){
        cdll->head = cdll->tail = node;
        cdll->tail->next = cdll->head;
        cdll->head->prev = cdll->tail;
    }
    else{
        cdll->head->prev = node;
        node->next = cdll->head;
        node->prev = cdll->tail;
        cdll->tail->next = node;
        cdll->head = node;
    }
}
void insertAtEnd(struct CircularDoublyLinkedList *cdll,int data){
    struct Node *node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    if(!cdll->head){
        cdll->head = cdll->tail = node;
        cdll->head->prev = node;
        node->next = cdll->head;
    }
    else{
        cdll->tail->next = node;
        node->prev = cdll->tail;
        node->next = cdll->head;
        cdll->head->prev = node;
        cdll->tail = node;
    }
}
```

```
void insertAtIndex(struct CircularDoublyLinkedList *cdll,int index,int data){

   if(index == 0){
      insertAtBegin(cdll,data);
   }
   else{
      struct Node *node = (struct Node*)malloc(sizeof(struct Node));
      node->data = data;
      int i = 0;
      struct Node* curr = cdll->head;
      while(i<index-1 && curr){
         i++;
         curr = curr->next;
      }
      if(curr){
         curr->next->prev = node;
         node->next = curr->next;
         curr->next = node;
         node->prev = curr;
         if(node->next == NULL){
            node->next = cdll->head;
            cdll->head->prev = node;
         }
      }
      else{
         printf("Index out of range");
      }
   }
}
void deleteAtBegin(struct CircularDoublyLinkedList *cdll)
{
   if(!cdll->head){
      printf("LinkedList is Empty");
   }
   else if(cdll->head == cdll->tail){
      cdll->head = cdll->tail = NULL;
   }
   else{
      struct Node *delNode = cdll->head;
      cdll->head = cdll->head->next;
      cdll->head->prev = cdll->tail;
      cdll->tail->next = cdll->head;
      cdll->head->prev = NULL;
      free(delNode);
   }
}
void deleteAtEnd(struct CircularDoublyLinkedList *cdll){
   if(!cdll->head){
      printf("LinkedList is Empty");
   }
   else if(!cdll->head->next){
      cdll->head = cdll->tail = NULL;
   }
   else{
      struct Node *delNode = cdll->tail;
      cdll->tail = cdll->tail->prev;
      cdll->tail->next = cdll->head;
```

```
        cdll->head->prev = cdll->tail;
        cdll->tail->next = NULL;
        free(delNode);
    }
}
void deleteAtIndex(struct CircularDoublyLinkedList* cdll,int index){
    if(!cdll->head){
        printf("LinkedList is Empty");
    }
    else if(index == 0){
        deleteAtBegin(cdll);
        return;
    }
    else{
        struct Node* curr = cdll->head;
        int i=0;
        while(i<index-1 && curr){
            i++;
            curr = curr->next;
        }
        if(curr && curr->next){
            struct Node* delNode = curr->next;
            curr->next = curr->next->next;
            if(!curr->next){
                cdll->tail = curr;
                cdll->tail->next = cdll->head;
                cdll->head->prev = cdll->tail;
            }
            else{
                curr->next->prev = curr;
            }
            free(delNode);
        }
        else{
            printf("Index out of range");
        }
    }
}
void displayForward(struct CircularDoublyLinkedList* cdll){
    if(!cdll->head){
        printf("Linked List is Empty\n");
        return;
    }
    struct Node* curr = cdll->head;
    while(curr!=cdll->tail){
        printf("%d->",curr->data);
        curr = curr->next;
    }
    printf("%d->NULL\n",cdll->tail->data);
}
void displayBackward(struct CircularDoublyLinkedList* cdll){
    if(!cdll->head){
        printf("Linked List is Empty\n");
        return;
    }
    struct Node* curr = cdll->tail;
    while(curr!=cdll->head){
```

```
      printf("%d->",curr->data);
      curr = curr->prev;
   }
   printf("%d->NULL\n",cdll->head->data);
}
int main(){
   struct CircularDoublyLinkedList cdll = {NULL,NULL};
   insertAtBegin(&cdll,10);
   displayForward(&cdll);
   displayBackward(&cdll);
   insertAtBegin(&cdll,20);
   displayForward(&cdll);
   displayBackward(&cdll);
   insertAtEnd(&cdll,30);
   displayForward(&cdll);
   displayBackward(&cdll);
   insertAtIndex(&cdll,1,40);
   displayForward(&cdll);
   displayBackward(&cdll);
   deleteAtBegin(&cdll);
   displayForward(&cdll);
   displayBackward(&cdll);
   deleteAtEnd(&cdll);
   displayForward(&cdll);
   displayBackward(&cdll);
   deleteAtIndex(&cdll,1);
   displayForward(&cdll);
   displayBackward(&cdll);
   deleteAtIndex(&cdll,0);
   displayForward(&cdll);
   displayBackward(&cdll);
}
```

4.Write Programs to implement the Stack operations using an array

```
#include<stdio.h>
#include<stdlib.h>
int a[15],ch,i,n,sp=-1,max=4;
void push();
void pop();
void display();
int main()
{
   while(1)
   {
      printf("\n1.push\n2.pop\n3.display\n4.exit\n");
      printf("enter the ur choice \n");
      scanf("%d",&ch);
      switch(ch){
         case 1:push();break;
         case 2:pop();break;
         case 3:display();break;
         case 4:exit(0);
      }
   }
   return 0;
}
void push(){
```

```
   if(sp==max)
   {
      printf("stack is full\n");
   }
   else{
      printf("enter element value\n");
      scanf("%d",&n);
      sp=sp+1;
      a[sp]=n;
      display();
   }
}
void pop(){
   if(sp==-1)
   printf("under flow");
   else
   {
      printf("deleted element is %d\n",a[sp]);
      sp=sp-1;
      display();
   }
}
void display(){
   printf("stack elements \n");
   for(i=sp;i>=0;i--)
   printf("%d ",a[i]);
}
```

5. Write a program using stacks to convert a given infix expression to postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_SIZE 100
int isOperator(char ch) {
   return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^');
}
int precedence(char op) {
   if (op == '^')
      return 3;
   else if (op == '*' || op == '/')
      return 2;
   else if (op == '+' || op == '-')
      return 1;
   else
      return -1;
}
void infixToPostfix(char* infix, char* postfix) {
   int i, j;
   char stack[MAX_SIZE];
   int top = -1;
   for (i = 0, j = 0; infix[i] != '\0'; i++) {
      if (isalnum(infix[i])) {
         postfix[j++] = infix[i];
      } else if (infix[i] == '(') {
         stack[++top] = '(';
```

```
    } else if (infix[i] == ')') {
        while (top != -1 && stack[top] != '(') {
            postfix[j++] = stack[top--];
        }
        top--; // Discard the '('
    } else { // infix[i] is an operator
        while (top != -1 && precedence(infix[i]) <= precedence(stack[top])) {
            postfix[j++] = stack[top--];
        }
        stack[++top] = infix[i];
    }
  }
  while (top != -1) {
    postfix[j++] = stack[top--];
  }
  postfix[j] = '\0';
}
int main() {
  char infix[MAX_SIZE];
  printf("Enter an infix expression: ");
  fgets(infix, MAX_SIZE, stdin);
  if (infix[strlen(infix) - 1] == '\n')
    infix[strlen(infix) - 1] = '\0';
  char postfix[MAX_SIZE];
  infixToPostfix(infix, postfix);
  printf("Postfix expression: %s\n", postfix);
  return 0;
}
```

6. Write Programs to implement the Stack operations using Liked List.

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();
struct node{
  struct node *next;
  int data;
};
struct node *head;
int ch;
int main(){
  while(1){
  printf("1. push element into stack\n");
  printf("2. pop element from stack\n");
  printf("3. Display the stack\n");
  printf("4. exit\n");

  printf("enter the ur choice\n");
  scanf("%d",&ch);
  switch(ch)
  {
    case 1: push();break;
    case 2: pop();break;
    case 3: display();break;
    case 4: exit(0);
  }
```

```
}
return 0;
}
void push(){
   int data;
   struct node *ptr=(struct node*)malloc(sizeof(struct node));
   if(ptr==NULL)
   {
      printf("overflow");
   }
   else{
      printf("Enter the value\n");
      scanf("%d",&data);
      if(head==NULL)
      {
         ptr->data=data;
         ptr->next=NULL;
         head=ptr;
      }
      else
      {
         ptr->data=data;
         ptr->next=head;
         head=ptr;
      }
      display();
   }
}
void pop(){
   int data;
   struct node *ptr;
   if(head==NULL)
   {
      printf("Underflow");
   }
   else{
      data=head->data;
      ptr=head;
      head=head->next;
      free(ptr);
      display();
   }
}
void display(){
   int i;
   struct node *ptr;
   ptr=head;
   if(ptr==NULL)
   {
      printf("Stack is empty\n");
   }
   else{
      printf("Stack elements\n");
      while(ptr!=NULL)
      {
         printf("%d-> ",ptr->data);
         ptr=ptr->next;
```

```
    }
    printf("NULL\n");
  }
}
```

7. Write Programs to implement the Queue operations using an array.

```
#include<stdio.h>
#include<stdlib.h>
int a[5],ch,i,n,front=-1,rear=-1,max=4;
void enqueue();
void dequeue1();
void display();
int main()
{
  while(1)
  {
    printf("\n1.insert\n2.delete\n3.display\n4.exit\n");
    printf("enter the ur choice \n");
    scanf("%d",&ch);
    switch(ch){
      case 1:enqueue();break;
      case 2:dequeue1();break;
      case 3:display();break;
      case 4:exit(0);
    }
  }
}
void enqueue(){
  if(rear==max)
  {
    printf("Queue is full\n");
  }
  else{
    printf("enter element value\n");
    scanf("%d",&n);
    rear=rear+1;
    a[rear]=n;
    display();
  }
}
void dequeue1(){
  if(front==max)
  printf("Queue is empty");
  else
  {
    front=front+1;
    printf("deleted element is %d\n",a[front]);
    display();
  }
}
void display(){
  printf("Queue elements \n");
  for(i=front+1;i<=rear;i++)
  printf("%d ",a[i]);
}
```

8. Write Programs to implement the Queue operations using Liked List.

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct node *next;
};
struct node *front;
struct node *rear;
void insert();
void delete();
void display();
int main()
{
   printf("1.insert\n2.delete\n3.display\n4.exit");
   while(1)
   {
      int ch;
      printf("Enter the ur choice\n");
      scanf("%d",&ch);
      switch (ch)
      {
      case 1: insert();break;
      case 2: delete();break;
      case 3: display();break;
      case 4: exit(0);
      }
   }
   return 0;
}
void insert()
{
   int value;
   printf("Enter element value\n");
   scanf("%d",&value);
   struct node * ptr;
   ptr = (struct node * ) malloc(sizeof(struct node));
   ptr -> data = value;
   ptr -> next = NULL;
   if ((front == NULL) && (rear == NULL))
    {
      front = rear = ptr;
      display();
   }
   else
    {
      rear -> next = ptr;
      rear = ptr;
      display();
   }

}
void delete()
 {
   if (front == NULL)
   {
      printf("Underflow\n");
   }
```

```
    else
    {
       struct node * temp = front;
       int temp_data = front -> data;
       front = front -> next;
       free(temp);
       display();
    }
}
void display()
{
   struct node * temp;
   if ((front == NULL) && (rear == NULL))
   {
      printf("Queue is Empty\n");
   }
   else
   {
      printf("The queue elements are \n");
      temp = front;
      while (temp)
      {
         printf("%d->", temp -> data);
         temp = temp -> next;
      }
      printf("NULL\n");
   }
}
```

9. Write a program for Binary Search Tree Traversals

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   struct node *left;
   int data;
   struct node *right;

}*root=NULL,*temp=NULL,*prev=NULL,*i=NULL;
void insert(int);
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);
int ch,n;
int main()
{
   printf("\n1.insert\n2.preorder\n3.inorder\n4.postorder\n5.exit\n");
   while(1)
   {
      printf(" enter ur choice\n");
      scanf("%d",&ch);
      if(ch==1)
      {
         printf("enter the number\n");
         scanf("%d",&n);
         insert(n);
      }
```

```
      if(ch==2)
      preorder(root);
      if(ch==3)
      inorder(root);
      if(ch==4)
      postorder(root);
      if(ch==5)
      exit(0);
   }
   return 0;
}
void insert(int n)
{
   temp=(struct node*)malloc(sizeof(struct node));
   temp->data=n;
   temp->left=temp->right=NULL;
   if(root==NULL)
   {
      root=temp;
   }
   else{
      i=prev=root;
      while(i!=NULL)
      {
         prev=i;
         if(n<=i->data)
         {
            i=i->left;
         }
         else
         {
            i=i->right;
         }
      }
      if(n<=prev->data)
      prev->left=temp;
      else
      prev->right=temp;
   }
}
void preorder(struct node *r)
{
   if(r!=NULL)
   {
      printf("%d-",r->data);
      preorder(r->left);
      preorder(r->right);
   }
}
void inorder(struct node *r)
{
   if(r!=NULL)
   {
      inorder(r->left);
      printf("%d-",r->data);
      inorder(r->right);
   }
```

```
}
void postorder(struct node *r)
{
   if(r!=NULL)
   {
      postorder(r->left);
      postorder(r->right);
      printf("%d-",r->data);

   }
```

10. Write a program to search an item in a given list using the following Searching Algorithms

a. Linear Search

b. Binary Search.

/* Linear Search*/

```
#include<stdio.h>
int linearSearch(int a[],int n,int target)
{
   int i;
   for(i=0;i<n;i++)
   {
      if(a[i]==target)
      {
         return i;
      }
   }
   return -1;
}
int main()
{
   int n;
   printf("enter size of arraya\n");
   scanf("%d",&n);
   int a[10];
   for(int i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   int taraget;
   printf("enter the taraget value\n");
   scanf("%d",&taraget);
   int result=linearSearch(a,n,taraget);
   if(result==-1)
   {
      printf("element is not found\n");
   }else
   {
      printf("element is found:%d",result);
   }
}
```

/*Binary Search.*/

```
#include<stdio.h>
int binarySearch(int a[],int l,int h,int x)
{
   if(h>l)
   {
      int mid=l+(h-l)/2;
      if(a[mid]==x)
```

```
      return mid;
      if(a[mid]>x)
      {
         return binarySearch(a,l,mid-1,x);
      }
         return binarySearch(a,mid+1,h,x);
   }
   return -1;
}
int a[10],n,x,i;
int main()
{
   printf("enter the size of arrya- ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   printf("enter the x value \n");
   scanf("%d",&x);
   int index = binarySearch(a, 0, n-1, x);

   if (index == -1) {
      printf("Element is not present in array");
   }
   else {
      printf("Element is present at index %d", index);
   }
}
```

11. Write a program for implementation of the following Sorting Algorithms

1. **Bubble Sort**
2. **Insertion Sort**
3. **Quick Sort**

/*Bubble Sort */

```
#include<stdio.h>
int i,j,n,a[10];
int bubble_sort(int a[],int n)
{
   for(i=0;i<n-1;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
         if(a[j]>a[j+1])
         {
            int temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
         }

      }
   }
}
int main()
{
   printf("enter the size of array\n");
   scanf("%d",&n);
   printf("enter the elements in array\n");
```

```
   for( i=0;i<n;i++)
   scanf("%d",&a[i]);
   bubble_sort(a,n);
   printf("Sorted array:");
   for(i=0;i<n;i++)
   printf("%d ",a[i]);
}
```

/*Insertion Sort */

```
#include<stdio.h>
int n,i,j,a[10],temp;
int insert(int a[],int n){
   for(i=0;i<n;i++){
   temp=a[i];
   j=i-1;
   while(j>=0 && temp<=a[j])
   {
      a[j+1]=a[j];
      j=j-1;
   }
   a[j+1]=temp;
   }
}
int main()
{
   printf("Enter the size of array\n");
   scanf("%d",&n);
   printf("Enter the array elements_\n");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   insert(a,n);
   printf("sorted array is ");
   for(i=0;i<n;i++)
   printf("%d ",a[i]);
}
```

/*Quick Sort*/

```
#include<stdio.h>
 int n,a[10],i,j;
 int partition(int a[],int start,int end)
 {
   int pivot=a[end];
   i=start-1;
   for(j=start;j<=end-1;j++)
   {
      if(a[j]<pivot)
      {
         i++;
         int t=a[i];
         a[i]=a[j];
         a[j]=t;
      }
   }
   int t=a[i+1];
   a[i+1]=a[end];
   a[end]=t;
   return (i+1);
```

```
}
int quick(int a[],int start,int end)
{
  if(start<end)
  {
    int p=partition(a,start,end);
    quick(a,start,p-1);
    quick(a,p+1,end);
  }
}
int main()
{
  printf("enter the size of array\n");
  scanf("%d",&n);
  printf("enter the elements in array\n");
  for( i=0;i<n;i++)
  scanf("%d",&a[i]);
  quick(a,0,n-1);
  printf("Sorted array:");
  for(i=0;i<n;i++)
  printf("%d ",a[i]);
}
// heap sort
#include <stdio.h>
void heapify(int[], int);
void build_maxheap(int heap[], int n){
  int i, j, c, r, t;
  for (i = 1; i < n; i++) {
    c = i;
    do {
      r = (c - 1) / 2;
      if (heap[r] < heap[c])
      { // to create MAX heap array
        t = heap[r];
        heap[r] = heap[c];
        heap[c] = t;
      }
      c = r;
    } while (c != 0);
  }
  printf("Heap array: ");
  for (i = 0; i < n; i++)
    printf("%d ", heap[i]);
  heapify(heap, n);
}
void heapify(int heap[], int n){
  int i, j, c, root, temp;
  for (j = n - 1; j >= 0; j--) {
    temp = heap[0];
    heap[0] = heap[j]; // swap max element with rightmost leaf element
    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) { // again rearrange to max heap array
```

```
        temp = heap[root];
        heap[root] = heap[c];
        heap[c] = temp;
      }
      root = c;
    } while (c < j);
  }
  printf("\nThe sorted array is: ");

  for (i = 0; i < n; i++)
    printf("%d ", heap[i]);
}
int main(){
  int n, i, j, c, root, temp,heap[10];
  printf("enter size of array: &d",n);
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
   scanf("%d",&heap[i]);
  }
  build_maxheap(heap, n);
}

// C recursive function to solve tower of hanoi puzzle
#include <stdio.h>
void towerOfHanoi(int n, char from, char to, char aux)
{
        if (n == 1)
        {
                printf("\n Move disk 1 from rod %c to rod %c", from, to);
                return;
        }
        towerOfHanoi(n-1, from, aux, to);
        printf("\n Move disk %d from rod %c to rod %c", n, from, to);
        towerOfHanoi(n-1, aux, to, from);
}

int main()
{
        int n = 4; // Number of disks
        towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
        return 0;
}
//matrix mulitiplication
  #include<stdio.h>
  #include<stdlib.h>
  int main(){
  int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
  system("cls");
  printf("enter the number of row=");
  scanf("%d",&r);
  printf("enter the number of column=");
  scanf("%d",&c);
  printf("enter the first matrix element=\n");
  for(i=0;i<r;i++)
  {
     for(j=0;j<c;j++)
```

```
   {
      scanf("%d",&a[i][j]);
   }
}
printf("enter the second matrix element=\n");
for(i=0;i<r;i++)
{
   for(j=0;j<c;j++)
   {
      scanf("%d",&b[i][j]);
   }
}

printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
   for(j=0;j<c;j++)
   {
      mul[i][j]=0;
      for(k=0;k<c;k++)
      {
         mul[i][j]+=a[i][k]*b[k][j];
      }
   }
}
//for printing result
for(i=0;i<r;i++)
{
   for(j=0;j<c;j++)
   {
      printf("%d\t",mul[i][j]);
   }
   printf("\n");
}
return 0;
}
```