PROGRAM 5

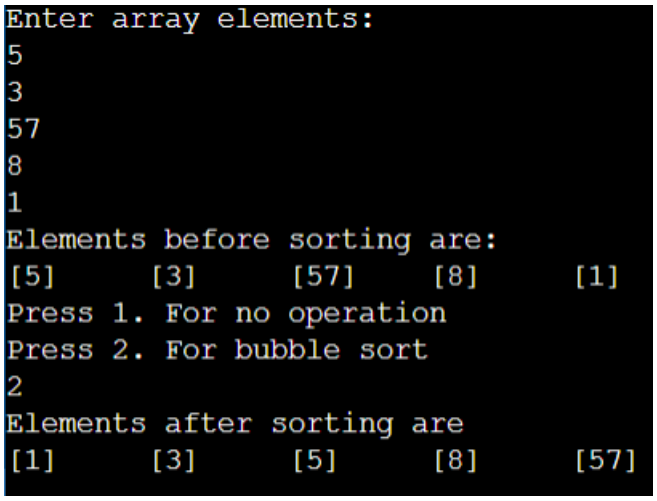
WAP to perform Bubble sort

INPUT:

```
#include<stdio.h>
int main()
{
 int arr[10];
 int size,i,j,temp;
 printf("Enter the size of array:\n");
 scanf("%d",&size);
 printf("Enter array elements:\n");
 for(i=0;i<size;i++)
 {
 scanf("%d",&arr[i]);
 }
 printf("Elements before sorting are:\n");
 for(i=0;i<size;i++)
 {
 printf("[%d]\t",arr[i]);
 }
 int x;
 printf("\nPress 1. For no operation\nPress 2. For bubble sort\n");
 scanf("%d",&x);
 switch(x)
 {
 case 1:
 {
 printf("No sorting is performed");
 }
 case 2:
 {
 for(int i=0;i<size-1;i++)
```

```
{
for(j=0;j<size-1;j++)
{
if(arr[j]>arr[j+1])
{
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
}
}
}
printf("Elements after sorting are\n");
for(i=0;i<size;i++)
{
printf("[%d]\t",arr[i]);
}
}
}
}
```

OUTPUT:



PROGRAM 6

WAP to find 2nd largest element in an array

INPUT:

```
#include<stdio.h>
int main()
{
 int arr[10],lar_el=0,Second_lar_el=0;
 int size,i;
 printf("Enter the size of array:");
 scanf("%d",&size);
 printf("Enter the array elements:\n");
 for(i=0;i<size;i++)
 {
 scanf("%d",&arr[i]);
 }
for(i=0;i<size;i++)
{
 if(lar_el<arr[i])
 {
 Second_lar_el=lar_el;
 lar_el=arr[i];
 }
 else if(Second_lar_el<arr[i])
 {
 Second_lar_el=arr[i];
 }
}
printf("The largest element is:%d\nSecond Largest element is:%d",lar_el,Second_lar_el);
}
```

OUTPUT:

```
Enter the size of array:5
Enter the array elements:
2
6
4
3
7
The largest element is:7
Second Largest element is:6
```

PROGRAM 7

WAP to calculate length of a string

INPUT:

```
#include<stdio.h>
int main()
{
 char a[20];
 int i,length=0;
 printf("Enter the string:");
 scanf("%s",&a);
 for(i=0;a[i]!='\0';i++)
 {
 length++;
 }
 printf("Length of String is: %d",i);
}
```

OUTPUT:

```
Enter the string:Asmitha
Length of String is: 7
```

PROGRAM 8

WAP to do stack operation using array

INPUT:

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
```

```
{
   //clrscr();
   top=-1;
   printf("\n Enter the size of STACK[MAX=100]:");
   scanf("%d",&n);
   printf("\n\t STACK OPERATIONS USING ARRAY");
   printf("\n\t--------------------------------");
   printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
   do
   {
      printf("\n Enter the Choice:");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
         case 2:
         {
            pop();
            break;
         }
         case 3:
         {
            display();
            break;
         }
         case 4:
         {
            printf("\n\t EXIT POINT ");
            break;
```

```
        }
        default:
        {
           printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
        }

      }
   }
   while(choice!=4);
   return 0;
}
void push()
{
   if(top>=n-1)
   {
      printf("\n\tSTACK is over flow");

   }
   else
   {
      printf(" Enter a value to be pushed:");
      scanf("%d",&x);
      top++;
      stack[top]=x;
   }
}
void pop()
{
   if(top<=-1)
   {
      printf("\n\t Stack is under flow");
   }
   else
```

```
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }

}
```

OUTPUT:

```
Enter the size of STACK[MAX=100]:6

        STACK OPERATIONS USING ARRAY
       --------------------------------
        1.PUSH
        2.POP
        3.DISPLAY
        4.EXIT
Enter the Choice:3

The STACK is empty
Enter the Choice:2

        Stack is under flow
Enter the Choice:1
Enter a value to be pushed:78

Enter the Choice:3

The elements in STACK

78
Press Next Choice
Enter the Choice:1
Enter a value to be pushed:4

Enter the Choice:3

The elements in STACK

4
78
Press Next Choice
Enter the Choice:4

        EXIT POINT
```

PROGRAM 9

WAP for tower of Hanoi

INPUT:

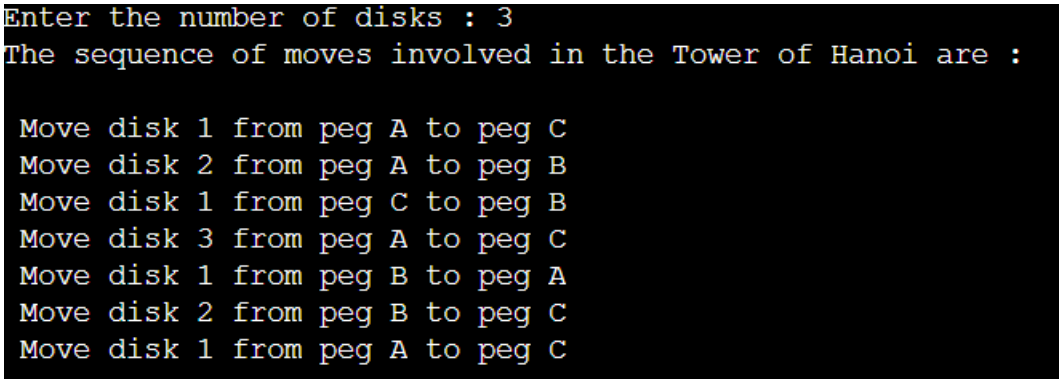
```
#include <stdio.h>


void towers(int, char, char, char);


int main()
{
 int num;


 printf("Enter the number of disks : ");
 scanf("%d", &num);
 printf("The sequence of moves involved in the Tower of Hanoi are :\n");
 towers(num, 'A', 'C', 'B');
 return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
 if (num == 1)
 {
 printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
 return;
 }
 towers(num - 1, frompeg, auxpeg, topeg);
 printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
 towers(num - 1, auxpeg, topeg, frompeg);
}
```

OUTPUT:



PROGRAM 10 WAP on queue operations INPUT: #include #include #define MAX 6 void insert(); void remov(); void display(); int queue[MAX], rear=-1, front=-1, item; int main() { int ch; do { printf("\n\n1. Insert\n2. Delete\n3. Display\n4. Exit\n"); printf("\nEnter your choice:"); scanf("%d", &ch); switch(ch) { case 1: insert(); break; case 2: remov(); break; case 3: display(); break; case 4: exit(0); default: printf("\n\nInvalid entry. Please try again...\n"); } } while(ch<=4); } void insert() { if(rear == MAX-1) printf("\nQueue is full."); else { printf("\n\nEnter ITEM:"); scanf("%d", &item); if (rear == -1 && front == -1) { rear = 0; front = 0; } else rear++; queue[rear] = item; printf("\n\nItem inserted: %d", item); } } void remov() { if(front == -1) printf("\n\nQueue is empty."); else { item = queue[front]; if (front == rear) { front = -1; rear = -1; } else front++; printf("\n\nItem deleted: %d", item); } } void display() { int i; if(front == -1) printf("\n\nQueue is empty."); else { printf("\n\n"); for(i=front; i<=rear; i++) printf("%d", queue[i]); } }

PROGRAM 11 WAP for circular queue operations INPUT: #include # define MAX 5 int cqueue_arr[MAX]; int front = -1; int rear = -1; void insert(int item) { if((front == 0 && rear == MAX-1) || (front == rear+1)) { printf("Queue Overflow n"); return; } if(front == -1) { front = 0; rear = 0; } else { if(rear == MAX-1) rear = 0; else rear = rear+1; } cqueue_arr[rear] = item ; } void deletion() { if(front == -1) { printf("Queue Underflown"); return ; } printf("\nElement deleted from queue is : %dn",cqueue_arr[front]); if(front == rear) { front = -1; rear=-1; } else { if(front == MAX-1) front = 0; else front = front+1; } } void display() { int front_pos = front,rear_pos = rear; if(front == -1) { printf("Queue is emptyn"); return; } printf("Queue elements :n"); if(front_pos <= rear_pos) while(front_pos <= rear_pos) { printf("%d ",cqueue_arr[front_pos]); front_pos++; } else { while(front_pos <= MAX-1) { printf("%d ",cqueue_arr[front_pos]); front_pos++; } front_pos = 0; while(front_pos <= rear_pos) { printf("%d ",cqueue_arr[front_pos]); front_pos++; } } printf("n"); } int main() { int choice,item; do { printf("\n1.Insert"); printf("\n2.Delete"); printf("\n3.Display"); printf("\n4.Quit"); printf("\nEnter your choice : "); scanf("%d",&choice); switch(choice) { case 1 : printf("Input the element for insertion in queue : "); scanf("%d", &item); insert(item); break; case 2 : deletion(); break; case 3: display(); break; case 4: break; default: printf("Wrong choicen"); } }while(choice!=4); return 0; }

PROGRAM 12 WAP for creating and traversing a linked list INPUT: #include #include struct node { int data; struct node *next; } *head; void createList (int n); void traverseList (); int main () { int n; printf ("Enter the total number of nodes: "); scanf ("%d", &n); createList (n); printf ("\nData in the list \n"); traverseList (); return 0; } void createList (int n) { struct node *newNode, *temp; int data, i; head = (struct node *) malloc (sizeof (struct node)); if (head == NULL) { printf ("Unable to allocate memory."); exit (0); } printf ("Enter the data of node 1: "); scanf ("%d", &data); head->data = data; head->next = NULL; temp = head; for (i = 2; i <= n; i++) { newNode = (struct node *) malloc (sizeof (struct node)); if (newNode == NULL) { printf ("Unable to allocate memory."); break; } printf ("Enter the data of node %d: ", i); scanf ("%d", &data); newNode->data = data; newNode->next = NULL; temp->next = newNode; temp = temp->next; } } void traverseList () { struct node *temp; if (head == NULL) { printf ("List is empty."); return; } temp = head; while (temp != NULL) { printf ("Data = %d\n", temp->data); temp = temp->next; } }

PROGRAM 13 WAP for singly linked list INPUT: include #include #include struct node { int data; struct node *next; }*start=NULL,*q,*t; int main() { int ch; void insert_beg(); void insert_end(); int insert_pos(); void display(); void delete_beg(); void delete_end(); int delete_pos(); while(1) { printf("\n\n---- Singly Linked List(SLL) Menu ----"); printf("\n1.Insert\n2.Display\n3.Delete\n4.Exit\n\n"); printf("Enter your choice(1-4):"); scanf("%d",&ch); switch(ch) { case 1: printf("\n---- Insert Menu ----"); printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specified position\n4.Exit"); printf("\n\nEnter your choice(1-4):"); scanf("%d",&ch); switch(ch) { case 1: insert_beg(); break; case 2: insert_end(); break; case 3: insert_pos(); break; case 4: exit(0); default: printf("Wrong Choice!!"); } break; case 2: display(); break; case 3: printf("\n---- Delete Menu ----"); printf("\n1.Delete from beginning\n2.Delete from end\n3.Delete from specified position\n4.Exit"); printf("\n\nEnter your choice(1-4):"); scanf("%d",&ch); switch(ch) { case 1: delete_beg(); break; case 2: delete_end(); break; case 3: delete_pos(); break; case 4: exit(0); default: printf("Wrong Choice!!"); } break; case 4: exit(0); default: printf("Wrong Choice!!"); } } return 0; } void insert_beg() { int num; t=(struct node*)malloc(sizeof(struct node)); printf("Enter data:"); scanf("%d",&num); t->data=num; if(start==NULL) //If list is empty { t->next=NULL; start=t; } else { t->next=start; start=t; } } void

insert_end() { int num; t=(struct node*)malloc(sizeof(struct node)); printf("Enter data:"); scanf("%d",&num); t->data=num; t->next=NULL; if(start==NULL) //If list is empty { start=t; } else { q=start; while(q->next!=NULL) q=q->next; q->next=t; } } int insert_pos() { int pos,i,num; if(start==NULL) { printf("List is empty!!"); return 0; } t=(struct node*)malloc(sizeof(struct node)); printf("Enter data:"); scanf("%d",&num); printf("Enter position to insert:"); scanf("%d",&pos); t->data=num; q=start; for(i=1;inext==NULL) { printf("There are less elements!!"); return 0; } q=q->next; } t->next=q->next; q->next=t; return 0; } void display() { if(start==NULL) { printf("List is empty!!"); } else { q=start; printf("The linked list is:\n"); while(q!=NULL) { printf("%d->",q->data); q=q->next; } } } void delete_beg() { if(start==NULL) { printf("The list is empty!!"); } else { q=start; start=start->next; printf("Deleted element is %d",q->data); free(q); } } void delete_end() { if(start==NULL) { printf("The list is empty!!"); } else { q=start; while(q->next->next!=NULL) q=q->next; t=q->next; q->next=NULL; printf("Deleted element is %d",t->data); free(t); } } int delete_pos() { int pos,i; if(start==NULL) { printf("List is empty!!"); return 0; } printf("Enter position to delete:"); scanf("%d",&pos); q=start; for(i=1;inext==NULL) { printf("There are less elements!!"); return 0; } q=q->next; } t=q->next; q->next=t->next; printf("Deleted element is %d",t->data); free(t); return 0; }

PROGRAM 14 WAP for circular linked list operations INPUT: #include #include typedef struct Node { int info; struct Node *next; }node; node *front=NULL,*rear=NULL,*temp; void create(); void del(); void display(); int main() { int chc; do { printf("\nMenu\n\t 1 to create the element : "); printf("\n\t 2 to delete the element : "); printf("\n\t 3 to display the queue : "); printf("\n\t 4 to exit from main : "); printf("\nEnter your choice : "); scanf("%d",&chc); switch(chc) { case 1: create(); break; case 2: del(); break; case 3: display(); break; case 4: return 1; default: printf("\nInvalid choice :"); } }while(1); return 0; } void create() { node *newnode; newnode=(node*)malloc(sizeof(node)); printf("\nEnter the node value : "); scanf("%d",&newnode->info); newnode->next=NULL; if(rear==NULL) front=rear=newnode; else { rear->next=newnode; rear=newnode; } rear->next=front; } void del() { temp=front; if(front==NULL) printf("\nUnderflow :"); else { if(front==rear) { printf("\n%d",front->info); front=rear=NULL; } else { printf("\n%d",front->info); front=front->next; rear->next=front; } temp->next=NULL; free(temp); } } void display() { temp=front; if(front==NULL) printf("\nEmpty"); else { printf("\n"); for(;temp!=rear;temp=temp->next) printf("\n%d address=%u next=%u\t",temp->info,temp,temp->next); printf("\n%d address=%u next=%u\t",temp->info,temp,temp->next)

PROGRAM 15 WAP on Binary search tree operations INPUT: #include #include struct btnode { int value; struct btnode *l; struct btnode *r; }*root = NULL, *temp = NULL, *t2, *t1; void delete1(); void insert(); void delete(); void inorder(struct btnode *t); void create(); void search(struct btnode *t); void preorder(struct btnode *t); void postorder(struct btnode *t); void search1(struct btnode *t,int data); int smallest(struct btnode *t); int largest(struct btnode *t); int flag = 1; void main() { int ch; printf("\nOPERATIONS ---"); printf("\n1 - Insert an element into tree\n"); printf("2 - Delete an element from the tree\n"); printf("3 - Inorder Traversal\n"); printf("4 - Preorder Traversal\n"); printf("5 - Postorder Traversal\n"); printf("6 - Exit\n"); while(1) { printf("\nEnter your choice : "); scanf("%d", &ch); switch (ch) { case 1: insert(); break; case 2: delete(); break; case 3: inorder(root); break; case 4: preorder(root); break; case 5: postorder(root); break; case 6: exit(0); default : printf("Wrong choice, Please enter correct choice "); break; } } } void insert() { create(); if (root == NULL) root = temp; else search(root); } void create() { int data; printf("Enter data of node to be inserted : "); scanf("%d", &data); temp = (struct btnode *)malloc(1*sizeof(struct btnode)); temp-

>value = data; temp->l = temp->r = NULL; } void search(struct btnode *t) { if ((temp->value > t->value) && (t->r != NULL)) search(t->r); else if ((temp->value > t->value) && (t->r == NULL)) t->r = temp; else if ((temp->value < t->value) && (t->l != NULL)) search(t->l); else if ((temp->value < t->value) && (t->l == NULL)) t->l = temp; } void inorder(struct btnode *t) { if (root == NULL) { printf("No elements in a tree to display"); return; } if (t->l != NULL) inorder(t->l); printf("%d -> ", t->value); if (t->r != NULL) inorder(t->r); } void delete() { int data; if (root == NULL) { printf("No elements in a tree to delete"); return; } printf("Enter the data to be deleted : "); scanf("%d", &data); t1 = root; t2 = root; search1(root, data); } void preorder(struct btnode *t) { if (root == NULL) { printf("No elements in a tree to display"); return; } printf("%d -> ", t->value); if (t->l != NULL) preorder(t->l); if (t->r != NULL) preorder(t->r); } void postorder(struct btnode *t) { if (root == NULL) { printf("No elements in a tree to display "); return; } if (t->l != NULL) postorder(t->l); if (t->r != NULL) postorder(t->r); printf("%d -> ", t->value); } void search1(struct btnode *t, int data) { if ((data>t->value)) { t1 = t; search1(t->r, data); } else if ((data < t->value)) { t1 = t; search1(t->l, data); } else if ((data==t->value)) { delete1(t); } } void delete1(struct btnode *t) { int k; if ((t->l == NULL) && (t->r == NULL)) { if (t1->l == t) { t1->l = NULL; } else { t1->r = NULL; } t = NULL; free(t); return; } else if ((t->r == NULL)) { if (t1 == t) { root = t->l; t1 = root; } else if (t1->l == t) { t1->l = t->l; } else { t1->r = t->l; } t = NULL; free(t); return; } else if (t->l == NULL) { if (t1 == t) { root = t->r; t1 = root; } else if (t1->r == t) t1->r = t->r; else t1->l = t->r; t == NULL; free(t); return; } else if ((t->l != NULL) && (t->r != NULL)) { t2 = root; if (t->r != NULL) { k = smallest(t->r); flag = 1; } else { k =largest(t->l); flag = 2; } search1(root, k); t->value = k; } } int smallest(struct btnode *t) { t2 = t; if (t->l != NULL) { t2 = t; return(smallest(t->l)); } else return (t->value); } int largest(struct btnode *t) { if (t->r != NULL) { t2 = t; return(largest(t->r)); } else return(t->value); } OUTPUT

PROGRAM 16 WAP on Tree traversal- Inorder, Postorder, Preorder INPUT: #include #include struct node { int item; struct node* left; struct node* right; }; // Inorder traversal void inorderTraversal(struct node* root) { if (root == NULL) return; inorderTraversal(root->left); printf("%d ->", root->item); inorderTraversal(root->right); } // preorderTraversal traversal void preorderTraversal(struct node* root) { if (root == NULL) return; printf("%d ->", root->item); preorderTraversal(root->left); preorderTraversal(root->right); }// postorderTraversal traversal void postorderTraversal(struct node* root) { if (root == NULL) return; postorderTraversal(root->left); postorderTraversal(root->right); printf("%d ->", root->item); }// Create a new Node struct node* createNode(value) { struct node* newNode = malloc(sizeof(struct node)); newNode->item = value; newNode->left = NULL; newNode->right = NULL; return newNode; } // Insert on the left of the node struct node* insertLeft(struct node* root, int value) { root->left = createNode(value); return root->left; } // Insert on the right of the node struct node* insertRight(struct node* root, int value) { root->right = createNode(value); return root->right; } int main() { struct node* root = createNode(1); insertLeft(root, 12); insertRight(root, 9); insertLeft(root->left, 5); insertRight(root->left, 6); printf("Inorder traversal \n"); inorderTraversal(root); printf("\nPreorder traversal \n"); preorderTraversal(root); printf("\nPostorder traversal \n"); postorderTraversal(root); }

PROGRAM 17 WAP on insertion sort INPUT: #include #include void insertionSort(int arr[], int n) { int i, key, j; for (i = 1; i < n; i++) { key = arr[i]; j = i - 1; while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j]; j = j - 1; }arr[j + 1] = key; }} void printArray(int arr[], int n) { int i; for (i = 0; i < n; i++) printf("%d ", arr[i]); printf("\n"); } int main() { int arr[] = {12, 11, 13, 5, 6}; int n = sizeof(arr) / sizeof(arr[0]); insertionSort(arr, n); printArray(arr, n); return 0; }

PROGRAM 18 WAP on selection sort #include void swap (int *xp, int *yp) { int temp = *xp; *xp = *yp; *yp = temp; } void selectionSort (int arr[], int n) { int i, j, min_idx; for (i = 0; i < n - 1; i++) { min_idx = i; for (j = i + 1; j < n; j++) if (arr[j] < arr[min_idx]) min_idx = j; swap (&arr[min_idx], &arr[i]); } } void printArray (int arr[], int size) { int i; for (i = 0; i < size; i++) printf ("%d ", arr[i]); printf ("\n"); } int main () { int arr[] = { 64, 25, 12, 22, 11 }; int n = sizeof (arr) / sizeof (arr[0]); selectionSort (arr, n); printf ("Sorted array: \n"); printArray (arr, n); return 0; }

PROGRAM 19 WAP on heap sort #include void swap(int* a, int* b) { int temp = *a; *a = *b; *b = temp; } void heapify(int arr[], int N, int i) { int largest = i; int left = 2 * i + 1; int right = 2 * i + 2; if (left < N && arr[left] > arr[largest]) largest = left; if (right < N && arr[right] > arr[largest]) largest = right; if (largest != i) { swap(&arr[i], &arr[largest]); heapify(arr, N, largest); } } void heapSort(int arr[], int N) { for (int i = N / 2 - 1; i >= 0; i--) heapify(arr, N, i); for (int i = N - 1; i >= 0; i--) { swap(&arr[0], &arr[i]); heapify(arr, i, 0); } } void printArray(int arr[], int N) { for (int i = 0; i < N; i++) printf("%d ", arr[i]); printf("\n"); } int main() { int arr[] = { 12, 11, 13, 5, 6, 7 }; int N = sizeof(arr) / sizeof(arr[0]); heapSort(arr, N); printf("Sorted array is\n"); printArray(arr, N); }

PROGRAM 20 WAP on merge sort #include #include void merge(int arr[], int l, int m, int r) { int i, j, k; int n1 = m - l + 1; int n2 = r - m; int L[n1], R[n2]; for (i = 0; i < n1; i++) L[i] = arr[l + i]; for (j = 0; j < n2; j++) R[j] = arr[m + 1 + j]; i = 0; j = 0; k = l; while (i < n1 && j < n2) { if (L[i] <= R[j]) { arr[k] = L[i]; i++; } else { arr[k] = R[j]; j++; } k++; } while (i < n1) { arr[k] = L[i]; i++; k++; } while (j < n2) { arr[k] = R[j]; j++; k++; } } void mergeSort(int arr[], int l, int r) { if (l < r) { int m = l + (r - l) / 2; mergeSort(arr, l, m); mergeSort(arr, m + 1, r); merge(arr, l, m, r); } }void printArray(int A[], int size) { int i; for (i = 0; i < size; i++) printf("%d ", A[i]); printf("\n"); } int main() { int arr[] = {12, 11, 13, 5, 6, 7}; int arr_size = sizeof(arr) / sizeof(arr[0]); printf("Given array is \n"); printArray(arr, arr_size); mergeSort(arr, 0, arr_size - 1); printf("\nSorted array is \n"); printArray(arr, arr_size); return 0; }

PROGRAM 21 WAP on quick sort #include void swap(int *a, int *b) { int t = *a; *a = *b; *b = t; } int partition(int array[], int low, int high) { int pivot = array[high]; int i = (low - 1); for (int j = low; j < high; j++) { if (array[j] <= pivot) { i++; swap(&array[i], &array[j]); } } swap(&array[i + 1], &array[high]); return (i + 1); } void quickSort(int array[], int low, int high) { if (low < high) { int pi = partition(array, low, high); // recursive call on the left of pivot quickSort(array, low, pi - 1); quickSort(array, pi + 1, high); } } void printArray(int array[], int size) { for (int i = 0; i < size; ++i) { printf("%d ", array[i]); } printf("\n"); } int main() { int data[] = {8, 7, 2, 1, 0, 9, 6}; int n = sizeof(data) / sizeof(data[0]); printf("Unsorted Array\n"); printArray(data, n); quickSort(data, 0, n - 1); printf("Sorted array in ascending order: \n"); printArray(data, n); }

PROGRAM 22 WAP on shell sort #include void shellSort(int array[], int n) { for (int interval = n / 2; interval > 0; interval /= 2) { for (int i = interval; i < n; i += 1) { int temp = array[i]; int j; for (j = i; j >= interval && array[j - interval] > temp; j -= interval) { array[j] = array[j - interval]; } array[j] = temp; } } } void printArray(int array[], int size) { for (int i = 0; i < size; ++i) { printf("%d ", array[i]); } printf("\n"); } int main() { int data[] = {9, 8, 3, 7, 5, 6, 4, 1}; int size = sizeof(data) / sizeof(data[0]); shellSort(data, size); printf("Sorted array: \n"); printArray(data, size); }