**Assignment-5**

**EXPERIMENT – 6: Write a program to implement Heap Sort algorithm.**

#include <stdio.h>

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);

}

}

int main() {

int arr[30];

int n;

printf("\nEnter the size of the array:");

scanf("%d",&n);

printf("\nEnter the elements of the array:-\n");

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

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

heapSort(arr, n);

printf("\nSorted array in ascending order: \n");

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

printf("%d ", arr[i]);

}

printf("\n");

printf("\nSorted array in descending order: \n");

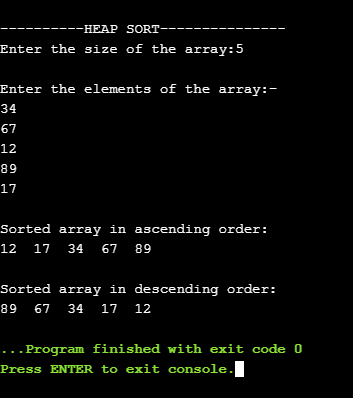
for (int i = n-1; i >= 0; i--) {

printf("%d ", arr[i]);

}

}

**OUTPUT-**

****

**EXPERIMENT – 7: Write a program to demonstrate the Merge Sort algorithm using linked list.**

#include<iostream>

using namespace std;

struct node

{

int data;

node \*next;

};

node\* NewNode(int d)

{

struct node \*temp = new node;

temp->data = d;

temp->next = NULL;

return temp;

}

node\* AddToList(node \*tail, int data)

{

struct node \*newnode;

newnode = NewNode(data);

if(tail == NULL)

{

tail = newnode;

}

else

{

tail->next = newnode;

tail = tail->next;

}

return tail;

}

node\* Merge(node\* h1, node\* h2)

{

node \*t1 = new node;

node \*t2 = new node;

node \*temp = new node;

if(h1 == NULL)

return h2;

if(h2 == NULL)

return h1;

t1 = h1;

while (h2 != NULL)

{

t2 = h2;

h2 = h2->next;

t2->next = NULL;

if(h1->data > t2->data)

{

t2->next = h1;

h1 = t2;

t1 = h1;

continue;

}

flag:

if(t1->next == NULL)

{

t1->next = t2;

t1 = t1->next;

}

else if((t1->next)->data <= t2->data)

{

t1 = t1->next;

goto flag;

}

else

{

temp = t1->next;

t1->next = t2;

t2->next = temp;

}

}

return h1;

}

void MergeSort(node \*\*head)

{

node \*first = new node;

node \*second = new node;

node \*temp = new node;

first = \*head;

temp = \*head;

if(first == NULL || first->next == NULL)

{

return;

}

else

{

while(first->next != NULL)

{

first = first->next;

if(first->next != NULL)

{

temp = temp->next;

first = first->next;

}

}

second = temp->next;

temp->next = NULL;

first = \*head;

}

MergeSort(&first);

MergeSort(&second);

\*head = Merge(first, second);

}

int main()

{

int n, i, num;

struct node \*head = new node;

struct node \*tail = new node;

head = NULL;

tail = NULL;

cout<<"-----------MERGE SORT USING LINKED LIST---------------";

cout<<"\nEnter the number of Nodes:";

cin>>n;

cout<<"\nEnter information to be stored in nodes"<<":-\n";

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

{

cin>>num;

tail = AddToList(tail, num);

if(head == NULL)

head = tail;

}

MergeSort(&head);

cout<<"\nSorting in Ascending Order:-";

while(head != NULL)

{

cout<<" "<<head->data;

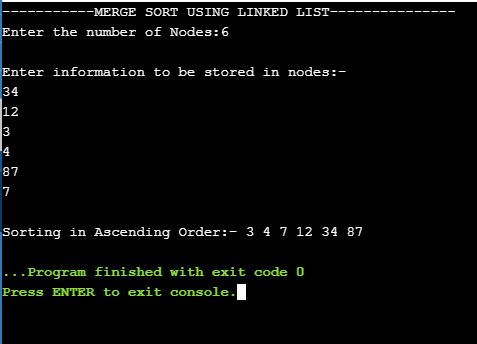
head=head->next;

}

return 0;

}

**OUTPUT-**



**EXPERIMENT – 8: Write a program to demonstrate the Quick Sort algorithm using linked list.**

#include <iostream>

#include <cstdio>

using namespace std;

struct Node

{

int data;

struct Node \*next;

};

void insert(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void print(struct Node \*node)

{

while (node != NULL)

{

cout<<node->data<<" ";

node = node->next;

}

cout<<"\n";

}

struct Node \*getTail(struct Node \*cur)

{

while (cur != NULL && cur->next != NULL)

cur = cur->next;

return cur;

}

struct Node \*partition(struct Node \*head, struct Node \*end,

struct Node \*\*newHead, struct Node \*\*newEnd)

{

struct Node \*pivot = end;

struct Node \*prev = NULL, \*cur = head, \*tail = pivot;

while (cur != pivot)

{

if (cur->data < pivot->data)

{

if ((\*newHead) == NULL)

(\*newHead) = cur;

prev = cur;

cur = cur->next;

}

else

{

if (prev)

prev->next = cur->next;

struct Node \*tmp = cur->next;

cur->next = NULL;

tail->next = cur;

tail = cur;

cur = tmp;

}

}

if ((\*newHead) == NULL)

(\*newHead) = pivot;

(\*newEnd) = tail;

return pivot;

}

struct Node \*quickSortRecur(struct Node \*head, struct Node \*end)

{

if (!head || head == end)

return head;

Node \*newHead = NULL, \*newEnd = NULL;

struct Node \*pivot = partition(head, end, &newHead, &newEnd);

if (newHead != pivot)

{

struct Node \*tmp = newHead;

while (tmp->next != pivot)

tmp = tmp->next;

tmp->next = NULL;

newHead = quickSortRecur(newHead, tmp);

tmp = getTail(newHead);

tmp->next = pivot;

}

pivot->next = quickSortRecur(pivot->next, newEnd);

return newHead;

}

void quickSort(struct Node \*\*headRef)

{

(\*headRef) = quickSortRecur(\*headRef, getTail(\*headRef));

return;

}

int main()

{

struct Node \*start = NULL;

int t,num;

cout<<"--------------QUICK SORT USING LINKED LIST--------------";

cout<<"\nEnter the number of Nodes:";

cin>>t;

cout<<"\nEnter the Information to be stored in Nodes:-\n";

while(t--){

cin>>num;

insert(&start, num);

}

quickSort(&start);

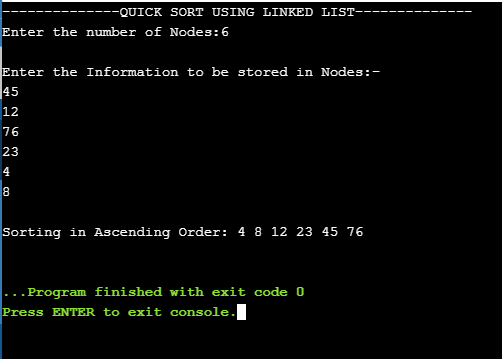
cout <<"\nSorting in Ascending Order: ";

print(start);

return 0;

}

**OUTPUT-**

****

**EXPERIMENT – 9: Write a program to demonstrate the insertion, deletion and searching in hash table.**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int ht[10]={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

int count,l,x,h\_x,i,h1\_x,k,key;

void Insert();

void Display();

void Search();

void menu();

void del();

void main()

{

printf("\nHashing techniques using Division Method");

menu();

}

void menu()

{

printf("\n\nMenu:-");

printf("\n1.Insert\n2.Display\n3.Search\n4.Delete\n5.Exit\n");

printf("\nEnter your choice:");

scanf("%d",&l);

switch(l)

{

case 1:{

Insert();

}break;

case 2:{

Display();

}break;

case 3:{

Search();

}break;

case 4:{

del();

}break;

case 5:{

exit(0);

}break;

default:

{

printf("\nInvalid Choice");

}

}

}

void Insert()

{

printf("\nCase1:-\tInsertion:-");

printf("\nEnter the element you want to insert:");

scanf("%d",&x);

h\_x=x%10;

printf("\nh\_x=%d",h\_x);

if(ht[h\_x]==-1)

{

ht[h\_x]=x;

printf("\nValue successfully inserted in the hash table.");

}

else

{

printf("\nCollision.");

}

menu();

}

void Display()

{

printf("\nCase2:-\tDisplay:-");

printf("\nDisplaying HASH TABLE");

for(k=0;k<10;k++)

{

printf("\n%d",ht[k]);

}

menu();

}

void Search()

{

printf("\nCase3:-\tSearch:-");

printf("\nEnter the element you want to search in the hash table:");

scanf("%d",&key);

h\_x=key%10;

if(ht[h\_x]==key)

{

printf("\nElement %d found at location %d",key,h\_x);

}

else

{

printf("\nElement not found.");

}

menu();

}

void del(){

printf("\nCase4:-\tDelete:-");

printf("\nEnter the element you want to delete:");

scanf("%d",&x);

h\_x=x%10;

if(ht[h\_x] == x)

{

printf("\nThis element %d has been removed.\n", x);

ht[h\_x]=-1;

}

else {

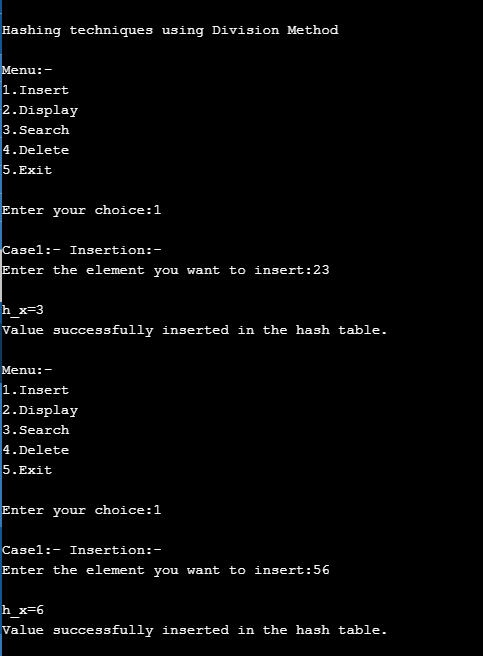
printf("\nThis element does not exist.\n");

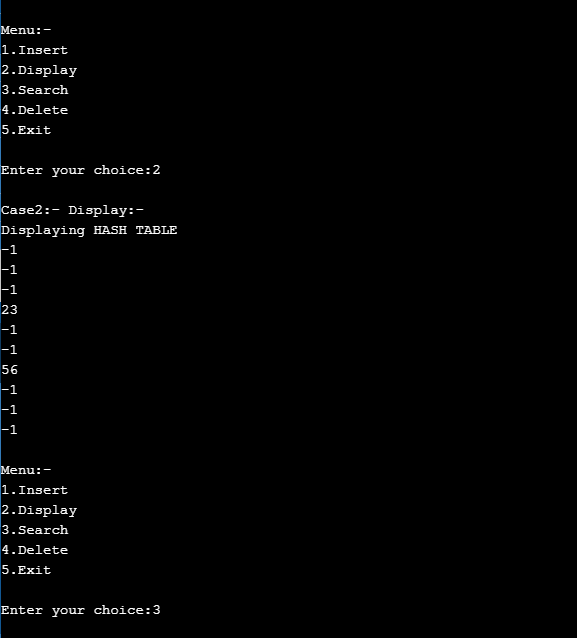
}

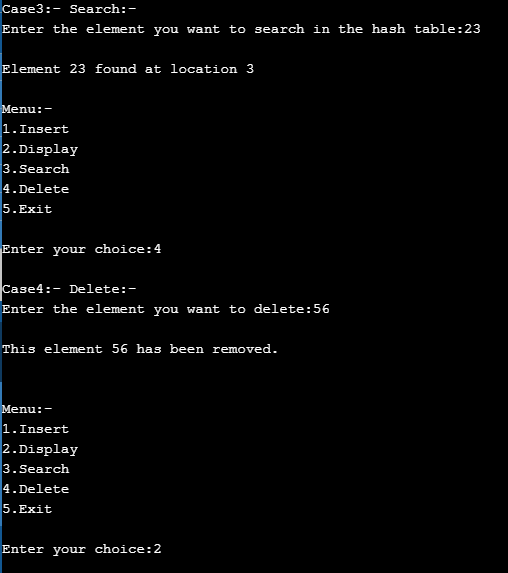
menu();

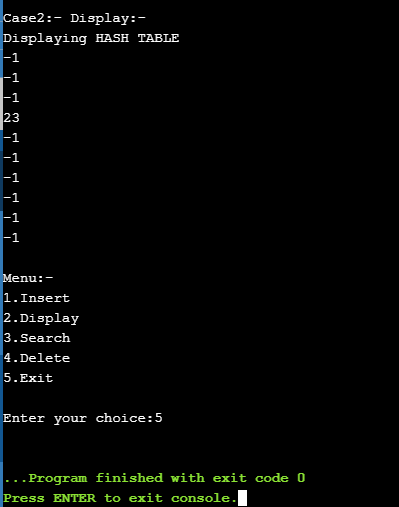
}

**OUTPUT-**

****

****

****

****