PROGRAM NO. 1

AIM: Write a program to implement linear search, Binary search.

CODE:

```
#include
int main(){
int n, target, i;
printf("Enter size: ");
scanf("%d", &n);
int arr[n];
printf("Enter %d numbers: ", n);
for (i = 0; i < n; i++)
scanf("%d", &arr[i]);
printf("Enter number to find: ");
scanf("%d", &target);
for (i = 0; i < n; i++)
{ if (arr[i] == target) {
 printf("Found at index %d\n", i);
 return 0;
}
 }
 printf("Not found\n");
 return 0;
}
```

OUTPUT:

```
Enter size: 5
Enter 5 numbers: 5
6
3
76
4
Enter number to find: 76
Found at index 4
```

Code:

```
#include
int binarySearch(int arr[], int n, int key){
int low = 0, high = n - 1;
while (low <= high) {
int mid = (low + high) / 2;
 if (arr[mid] == key) return mid;
```

```
else if (arr[mid] < key) low = mid + 1;
else high = mid - 1;
}
return -1;
}
 int main() {
 int n, key, i;
printf("Enter size: ");
scanf("%d", &n);
int arr[n];
printf("Enter %d sorted numbers: ", n);
for (i = 0; i < n; i++)
scanf("%d", &arr[i]);
printf("Enter number to find: ");
scanf("%d", &key);
int result = binarySearch(arr, n, key);
printf(result == -1 ? "Not found\n" : "Found at index %d\n", result);
return 0;
}
```

Output:

```
Enter size: 4
Enter 4 sorted numbers: 2
4
6
9
Enter number to find: 6
Found at index 3
```

Program no. 2

AIM: Write a program to implement, Insertion Sort and Selection Sort

CODE : -

```
#include
int main() {
int n, i, pos, value;
printf("Enter size: ");
scanf("%d", &n);
int arr[n + 1];
printf("Enter %d numbers: ", n);
for (i = 0; i < n; i++)
scanf("%d", &arr[i]);
```

```
printf("Enter position (0 to %d) and value to insert: ", n);
scanf("%d %d", &pos, &value);
if (pos < 0 || pos > n) {
printf("Invalid position\n");
return 1;
}
 for (i = n; i > pos; i--) arr[i] = arr[i - 1];
arr[pos] = value;
n++;
printf("Array after insertion: ");
for (i = 0; i < n; i++)
printf("%d ", arr[i]);
return 0;
}
```

Output:

```
Enter size: 5
Enter 5 numbers: 4
3
1
8
6
Enter position (0 to 5) and value to insert: 3
11
Array after insertion: 4 3 1 11 8 6
```

Code:

```
#include
void selectionSort(int arr[], int n) {
for (int i = 0; i < n - 1; i++) {
int minIndex = i;
for (int j = i + 1; j < n; j++) {
if (arr[j] < arr[minIndex]) minIndex = j;
}
int temp = arr[i];
arr[i] = arr[minIndex];
arr[minIndex] = temp;
}
}
int main() {
int n, i;
```

```
printf("Enter size: ");
scanf("%d", &n);
int arr[n];
printf("Enter %d numbers: ", n);
for (i = 0; i < n; i++)
scanf("%d", &arr[i]);
selectionSort(arr, n);
printf("Sorted array: ");
for (i = 0; i < n; i++)
printf("%d ", arr[i]);
return 0;
}
```

Output:

Enter size: 5
Enter 5 numbers: 4
3
6
7
9
Sorted array: 3 4 6 7 9

Code:

```
#include
void bubbleSort(int arr[], int n) {
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (arr[j] > arr[j + 1]) {
int temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
}
}
}
}
int main() {
int n, i;
```

```
printf("Enter size: ");
scanf("%d", &n);
int arr[n];
printf("Enter %d numbers: ", n);
for (i = 0; i < n; i++)
scanf("%d", &arr[i]);
bubbleSort(arr, n);
printf("Sorted array: ");
for (i = 0; i < n; i++)
printf("%d ", arr[i]);
return 0;
}
```

Output:

Enter size: 5
Enter 5 numbers: 12
43
54
23
78
Sorted array : 12 23 43 54 78

PROGRAM NO. 3

AIM: Write a program to implement quick Sort with a given list of integers in ascending orders.

Code:

```
#include
void swap(int *a, int *b) {
int temp = *a;
*a = *b;
*b = temp; }
int partition(int arr[], int low, int high) {
int pivot = arr[high];
int i = low - 1;
for(int j = low; j < high; j++) {
if(arr[j] < pivot) {
```

```
i++;
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return i + 1;
}
void quickSort(int arr[], int low, int high) {
if(low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
int main() {
int arr[100], n, i;
printf("Enter number of elements: ");
scanf("%d", &n);
printf("Enter elements:\n");
for(i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
quickSort(arr, 0, n - 1);
printf("Sorted array:\n");
for(i = 0; i < n; i++) {
printf("%d ", arr[i]);
}
return 0;
}
```

Output:

Enter size: 5

Enter 5 numbers: 12

32

59

43

1

Sorted array:

1 12 32 43 59

PROGRAM NO. 4

AIM: Write a program to implement Heap Sort with a given list of integers in ascending orders.

Code:

```
#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);
}
}
int main() {
int arr[100], n, i;
printf("Enter number of elements: ");
scanf("%d", &n);
printf("Enter elements:\n");
for(i = 0; i < n; i++)
scanf("%d", &arr[i]);
heapSort(arr, n);
printf("Sorted array:\n");
```

```
for(i = 0; i < n; i++)
printf("%d ", arr[i]);
return 0;
}
```

Output:

Enter number of elements: 5

Enter elements:

14

12

33

23

45

Sorted array:

12 14 23 33 45

PROGRAM NO. 5

AIM: Write a program to implement Merge Sort with a given list of integers in ascending orders

Code:

```
#include
void merge(int arr[], int l, int m, int r) {
int n1 = m - l + 1;
int n2 = r - m;
int L[100],
R[100];
for(int i = 0; i < n1; i++)
L[i] = arr[l + i];
for(int j = 0; j < n2; j++)
R[j] = arr[m + 1 + j];
int 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);
}
}
int main() {
int arr[100], n, i;
printf("Enter number of elements: ");
scanf("%d", &n);
printf("Enter elements:\n");
for(i = 0; i < n; i++)
scanf("%d", &arr[i]);
mergeSort(arr, 0, n - 1);
printf("Sorted array:\n");
for(i = 0; i < n; i++)
printf("%d ", arr[i]);
return 0;
}
```

Output:

Enter number of elements: 5

Enter elements:

12

33
24
76
87
Sorted array:
12 24 33 76 87

PROGRAM NO. 6
AIM: Write a program to implement stack using Array.

Code:

```
#include
#define SIZE 100
int stack[SIZE], top = -1;
void push(int value) {
if(top == SIZE - 1)
printf("Stack Overflow\n");
else {
top++;
stack[top] = value;
}
}
void pop() {
if(top == -1)
printf("Stack Underflow\n");
else top--;
}
void display() {
if(top == -1)
printf("Stack is empty\n");
else {
for(int i = top; i >= 0; i--)
printf("%d ", stack[i]);
printf("\n");
}
}
int main() {
int choice, value;
while(1) {
```

```
printf("1.Push 2.Pop 3.Display 4.Exit\n");
scanf("%d", &choice);
if(choice == 1) {
printf("Enter value: ");
scanf("%d", &value);
push(value);
}
else if(choice == 2) {
pop();
}
else if(choice == 3) {
display();
}
else if(choice == 4)
{
break;
}
else {
printf("Invalid choice\n");
}
}
return 0;
}
```

Output:

```
1. Push  2. Pop  3. Display  4. Exit
1
Enter value: 50
1. Push  2. Pop  3. Display  4. Exit
1
Enter value: 30
1. Push  2. Pop  3. Display  4. Exit
1
Enter value: 90
1. Push  2. Pop  3. Display  4. Exit
3
90 30 50
1. Push  2. Pop  3. Display  4. Exit
2
1. Push  2. Pop  3. Display  4. Exit
3
```

30 50
1. Push 2. Pop 3. Display 4. Exit
4

PROGRAM NO. 7

AIM: Write a program to implement stack using Linked Lists.

Code:

```
#include
#include
struct Node {
int data; struct Node* next;
};
struct Node* top = NULL;
void push(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if(newNode == NULL) {
printf("Heap Overflow\n");
return;
}
newNode->data = value;
newNode->next = top;
top = newNode;
}
void pop() {
if(top == NULL) {
printf("Stack Underflow\n");
return;
}
struct Node* temp = top;
top = top->next;
free(temp);
}
void display()
{
if(top == NULL) {
printf("Stack is empty\n");
return;
}
struct Node* temp = top;
```

```
while(temp != NULL) {
printf("%d ", temp->data);
temp = temp->next;
}
printf("\n");
}
int main() {
int choice, value;
while(1) {
printf("1.Push 2.Pop 3.Display 4.Exit\n");
scanf("%d", &choice);
if(choice == 1) {
printf("Enter value: ");
scanf("%d", &value);
push(value);
}
else
if(choice == 2) {
pop();
}
else if(choice == 3) {
display();
}
else if(choice == 4) {
break;
}
else {
printf("Invalid choice\n");
 }
}
 return 0;
 }
```

Output:

```
1. Push  2. Pop  3. Display  4. Exit
1
Enter value: 10
1. Push  2. Pop  3. Display  4. Exit
1
```

```
Enter value: 30
1. Push  2. Pop  3. Display  4. Exit
1
Enter value: 20
1. Push  2. Pop  3. Display  4. Exit
1
Enter value: 5
1. Push  2. Pop  3. Display  4. Exit
3
5 20 30 10
1. Push  2. Pop  3. Display  4. Exit
1
Enter value: 100
1. Push  2. Pop  3. Display  4. Exit
3
100 5 20 30 10
1. Push  2. Pop  3. Display  4. Exit
2
1. Push  2. Pop  3. Display  4. Exit
3
5 20 30 10
1. Push  2. Pop  3. Display  4. Exit
4
```

Program no. 8

AIM: Write a program to implement Queue and circular queue using Array

Code-

```
#include
#define SIZE 5
int queue[SIZE], front = -1, rear = -1;
void enqueue(int value) {
if(rear == SIZE - 1)
printf("Queue Overflow\n");
else { if(front == -1) // If the queue is empty front = 0;
Rear++;
queue[rear] = value;
}
}
void dequeue() {
```

```
if(front == -1)
printf("Queue Underflow\n");
else {
printf("Dequeued: %d\n", queue[front]);
if(front == rear) // If only one element is left front = rear = -1;
else front++;
}
}
void display() {
if(front == -1)
 printf("Queue is empty\n");
else {
for(int i = front; i <= rear; i++)
printf("%d ", queue[i]);
printf("\n");
}
}
int main() {
int choice, value;
while(1) {
printf("1.Enqueue 2.Dequeue 3.Display 4.Exit\n");
scanf("%d", &choice);
if(choice == 1) {
printf("Enter value: ");
scanf("%d", &value);
enqueue(value);
}
else if(choice == 2) {
dequeue();
}
else if(choice == 3) {
display();
}
else if(choice == 4) {
break;
}
else {
printf("Invalid choice\n");
}
}
```

```
return 0;
}
Output
1.Enqueue 2.Dequeue 3.Display 4.Exit
1
Enter value: 54
1.Enqueue 2.Dequeue 3.Display 4.Exit
1
Enter value: 34
1.Enqueue 2.Dequeue 3.Display 4.Exit
1
Enter value: 12
1.Enqueue 2.Dequeue 3.Display 4.Exit
1
Enter value: 5
1.Enqueue 2.Dequeue 3.Display 4.Exit
3 54 34 12 5
1.Enqueue 2.Dequeue 3.Display 4.Exit
2
Dequeued: 54
1.Enqueue 2.Dequeue 3.Display 4.Exit
3 34 12 5
1.Enqueue 2.Dequeue 3.Display 4.Exit
4
```

Program no. 9

Aim-Write a program to implement Queue and circular queue Linked Lists

Code:

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

};

int data;
```

```
struct Node* next;

struct Node* front = NULL;

struct Node* rear = NULL;

void enqueue(int value) {

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = NULL;

if(rear == NULL) {

front = rear = newNode;

} else {
rear->next = newNode;

rear = newNode;

}

}

void dequeue() {

if(front == NULL)

printf("Queue Underflow\n");

else {

struct Node* temp = front;

front = front->next;

if(front == NULL) // If the queue is empty, reset rear to NULL rear
= NULL;
```

```
free(temp);

void display() {

if(front == NULL)

printf("Queue is empty\n");

else {

struct Node* temp = front;

while(temp != NULL) {

printf("%d ", temp->data);
temp = temp->next;

}

}
printf("\n");

int main(){

int choice, value;

while(1){

printf("1.Enqueue 2.Dequeue 3.Display 4.Exit\n");

scanf("%d", &choice);

if(choice == 1) {

printf("Enter value:");

scanf("%d", &value);

enqueue(value);

} else if(choice == 2) {
```

```
dequeue();

} else if(choice == 3) {

}

}

display();
} else if(choice == 4) {

Break;
} else {

}

printf("Invalid choice\n");

return 0;
```

Output

```
1.Enqueue 2.Dequeue 3.Display 4.Exit 1
Enter value: 54
1.Enqueue 2.Dequeue 3.Display 4.Exit 1
Enter value: 34
1.Enqueue 2.Dequeue 3.Display 4.Exit 1
Enter value: 12
1.Enqueue 2.Dequeue 3.Display 4.Exit 1
Enter value: 5
1.Enqueue 2.Dequeue 3.Display 4.Exit 3
54 34 12 5
1.Enqueue 2.Dequeue 3.Display 4.Exit
2
Dequeued: 54
1.Enqueue 2.Dequeue 3.Display 4.Exit
3
34 12 5
1.Enqueue 2.Dequeue 3.Display 4.Exit 4
```

Program no. 10

Aim-Write a program to implement a single link list and its operations

Code:
```
#include <stdio.h>
#include <stdlib.h>

struct Node {

};

int data;

struct Node* next;

struct Node* head = NULL;

void insert(int value) {

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); newNode->data

                = value;

newNode->next = head;

head = newNode;

}
void delete() {
if(head == NULL)

printf("List is empty\n"); else
                {

struct Node* temp = head; head =

                head->next;

free(temp);
```

```
}

}

void display() {

struct Node* temp = head;


}

if(temp == NULL) {

printf("List is empty\n");

                return;

}

while(temp != NULL) { }

printf("%d -> ", temp->data);

temp = temp->next;

printf("NULL\n");

int main() {

int choice, value;

while(1){

printf("1.Insert 2.Delete 3.Display 4.Exit\n");

                scanf("%d", &choice);

if(choice == 1) {
```

```
printf("Enter value: ");

scanf("%d", &value);

insert(value);

} else if(choice == 2) {

delete();

} else if(choice == 3) {

display();

} else if(choice == 4) {

break;
} else {

printf("Invalid choice\n");
}

}

return 0;
}
```

Output

```
1.Insert 2.Delete 3.Display 4.Exit 1
Enter value: 5
1.Insert 2.Delete 3.Display 4.Exit
1
Enter value: 10
1.Insert 2.Delete 3.Display 4.Exit
1
Enter value: 15
1.Insert 2.Delete 3.Display 4.Exit
```

3
15 -> 10 -> 5 -> NULL
1.Insert 2.Delete 3.Display 4.Exit
2
1.Insert 2.Delete 3.Display 4.Exit 3
10 -> 5 -> NULL
1.Insert 2.Delete 3.Display 4.Exit
4

Program no. 11

Aim-Write a program to implement double link list and circular and its operations

Code: #include <stdio.h>

#include <stdlib.h>

struct Node { int data;

};

struct Node* prev;

struct Node* next;

struct Node* head = NULL;

void insertEnd(int value) {

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); newNode->data =

value;

}

newNode->prev = NULL;

```
newNode->next = NULL;

if(head == NULL) {

head = newNode;

return;

}

struct Node* temp = head;

while(temp->next!= NULL) temp

  = temp->next;

temp->next = newNode;

newNode->prev = temp;

void deleteEnd(){ if(head == NULL)

  printf("List is empty\n");

else if(head->next == NULL) {

  free(head);

head = NULL;

} else {

struct Node* temp = head;

while(temp->next!= NULL)
temp = temp->next;

temp->prev->next = NULL;
```

```
  free(temp);
}
}
void display() {
if(head == NULL) {
printf("List is empty\n");
  return;
}
}
struct Node* temp = head;
  while(temp != NULL) { }
printf("%d <-> ", temp->data); temp =
  temp->next;
printf("NULL\n");
int main() {
int choice, value;
while(1){
printf("1.Insert 2.Delete 3.Display 4.Exit\n");
  scanf("%d", &choice);
if(choice == 1) {
printf("Enter value: ");
```

```
scanf("%d", &value);

insertEnd(value);

} else if(choice == 2) {

deleteEnd();

} else if(choice == 3) {

display();

}else if(choice == 4) {

break;

} else {

printf("Invalid choice\n");

}

}
return 0;

}
```

Output

```
1.Insert 2.Delete 3.Display 4.Exit 1
Enter value: 54
1.Insert 2.Delete 3.Display 4.Exit
1
Enter value: 14
1.Insert 2.Delete 3.Display 4.Exit
1
Enter value: 5
```

1.Insert 2.Delete 3.Display 4.Exit
3
54 <-> 14 <-> 5 <-> NULL
1.Insert 2.Delete 3.Display 4.Exit
2
1.Insert 2.Delete 3.Display 4.Exit
3
54 <-> 14 <-> NULL
1.Insert 2.Delete 3.Display 4.Exit
4