PROGRAM 1

Write a program to search an element from a list. Give user the option to perform Linear or Binary search. Use Template functions.

ALGORITHM

1. For Linear Search:

```
LinearSearch(arr, target):
for each element in arr with index i do:
    if arr[i] is equal to target then
        return i  //if Target found at index i
end for
return -1  //if Target not found in the array
```

2. Use a sorting algorithm to sort the array before applying Binary Search.
3. For Binary Search:

```
BinarySearch(arr, target):
left = 0
right = length of arr - 1
while left <= right do:
    mid = low + (high - low) / 2 //to avoid overflow
    if arr[mid] is equal to target then
        return mid //Target found at index mid
    else if arr[mid] < target then
        left = mid + 1 //Search the right half
    else
        right = mid - 1 //Search the left half
return -1 Target not found in the array
```

4. Give choice to user and call the functions

PROGRAM CODE

```
#include<iostream>
#include<vector>
using namespace std;
template <typename T>
int linearSearch(vector<T>& arr, T& target){
    for(int i=0;i<arr.size();i++){
        if(arr[i]==target)
            return i;
    }
    return -1;
}
template <typename T>
int binarySearch(vector<T>& arr, T& target){
    int hole, value;
    for(int i=0;i<arr.size();i++){
        value = arr[i];
        hole = i - 1;
        while(hole>=0 && arr[hole]>value){
            arr[hole+1] = arr[hole];
            hole = hole - 1;
        }
        arr[hole+1]=value;
```

```
        }
        int low = 0;
        int high = arr.size()-1;
        while(low<=high){
                int mid = low + (high-low)/2;
                if(arr[mid]==target)
                        return mid;
                else if(arr[mid]<target)
                        return low = mid + 1;
                else
                        return low = mid - 1;
        }
        return -1;
}
int main(){
        cout << "Enter the number of elements: ";
        int n;
        cin >> n;
        vector<int> List;
        cout << "Enter the elements of the lists: ";
        int element;
        for(int i=0;i<n;i++){
                cin >> element;
                List.push_back(element);
        }
        cout << "Enter the element to search for: ";
        int target;
        cin >> target;
        cout << "Choose Searching Algorithm:\n";
        cout << "1. Linear Search\n";
        cout << "2. Binary Search\n";
        int choice;
        cout << "Enter Choice: ";
        cin >> choice;
        int result;
        switch(choice){
                case 1:
                        result = linearSearch(List,target);
                        break;
                case 2:
                        result = binarySearch(List,target);
                        break;
                default:
                        cout << "Invalid Choice. Exit Program.";
                        return 1;
        }
        if(result != -1)
                cout << "Element " << target << " found at index " << result << endl;
        else
                cout << "Element " << target << " not found in the list." << endl;
        return 0;
}
```

OUTPUT

SET 1:

Enter the number of elements: 5
Enter the elements of the lists: 3 2 4 6 1
Enter the element to search for: 2
Choose Searching Algorithm:
1. Linear Search
2. Binary Search
Enter Choice: 1
Element 2 found at index 1

SET 2:

Enter the number of elements: 3
Enter the elements of the lists: 66 8 999
Enter the element to search for: 8
Choose Searching Algorithm:
1. Linear Search
2. Binary Search
Enter Choice: 2
Element 8 found at index 0

PROGRAM 2

WAP using templates to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

ALGORITHM

1. For Insertion Sort:

```
InsertionSort(arr):
n = length of arr
for i from 1 to n - 1 do:
    key = arr[i]
    j = i – 1 //Move elements of arr[0..i-1] that are greater than key to one position ahead of their current position
    while j >= 0 and arr[j] > key do:
        arr[j + 1] = arr[j]
        j = j - 1
    arr[j + 1] = key
```

2. For Selection Sort:

```
SelectionSort(arr):
n = length of arr
for i from 0 to n-2 do:
 //Assume the current index is the minimum
    minIndex = i
//Find the index of the minimum element in the unsorted part of the array
for j from i+1 to n-1 do:
        if arr[j] < arr[minIndex] then
            minIndex = j  //Swap the found minimum element with the first element in the unsorted part
swap arr[i] and arr[minIndex]
```

3. For Bubble Sort:

```
BubbleSort(arr):
n = length of arr
for i from 0 to n-1 do:
// Last i elements are already sorted, so we don't need to check them
    for j from 0 to n-i-1 do:
    // Swap if the element found is greater than the next element
        if arr[j] > arr[j+1] then
            swap arr[j] and arr[j+1]
```

4. Open main function and give user choice to perform sorting and call the functions according to choice.

PROGRAM CODE

```
#include<bits/stdc++.h>
#include<vector>
using namespace std;
template <typename T>
void insertionSort(vector<T>& arr){
    int hole, value;
    for(int i=1;i<arr.size();i++)
    {
        value = arr[i];
        hole = i - 1;
```

```
                while(hole>=0 && arr[hole]>value){
                        arr[hole+1] = arr[hole];
                        hole = hole-1;
                }
                arr[hole+1] = value;
        }
}
template <typename T>
void selectionSort(vector<T>& arr){
        int minDex, i;
        for(i=0;i<arr.size()-1;i++){
                minDex=i;
                for(int j=i+1;j<arr.size();j++){
                        if(arr[j]<arr[minDex])
                                minDex=j;
                }
                int temp = arr[minDex];
                arr[minDex]=arr[i];
                arr[i]=temp;
        }
}
template <typename T>
void bubbleSort(vector<T>& arr){
        for(int i=0;i<arr.size()-1;i++){
                for(int j=0;j<arr.size()-1;j++){
                        if(arr[j]>arr[j+1]){
                                int temp = arr[j];
                                arr[j]=arr[j+1];
                                arr[j+1]=temp;
                        }
                }
        }
}
template <typename T>
void printArray(vector<T>& arr){
        int i;
        for(i=0;i<arr.size();i++)
        {
                cout<<arr[i] << " ";
        }
        cout << endl;
}
int main(){
        cout << "Enter the number of elements: ";
        int n;
        cin >> n;
        vector<int> List;
        cout << "Enter the elements of the lists: ";
        int element;
        for(int i=0;i<n;i++){
                cin >> element;
                List.push_back(element);
```

```
        }
        cout << "Unsorted Array: ";
        printArray(List);
        cout << "Choose Sorting Algorithm:\n";
        cout << "1. Insertion Sort\n";
        cout << "2. Selection Sort\n";
        cout << "3. Bubble Sort\n";
        int choice;
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice) {
        case 1:
                insertionSort(List);
                break;
        case 2:
                selectionSort(List);
                break;
        case 3:
                bubbleSort(List);
                break;
        default:
                cout << "Invalid Choice. Exit Program.";
                return 1;
        }
        cout << "Sorted Array: ";
        printArray(List);
        return 0;
}
```

OUTPUT

SET 1:

```
Enter the number of elements: 5
Enter the elements of the lists: 2 6 1 88 9
Unsorted Array: 2 6 1 88 9
Choose Sorting Algorithm:
1. Insertion Sort
2. Selection Sort
3. Bubble Sort
Enter Choice: 1
Sorted Array: 1 2 6 9 88
```

SET 2:

```
Enter the number of elements: 4
Enter the elements of the lists: 33 6 999 1
Unsorted Array: 33 6 999 1
Choose Sorting Algorithm:
1. Insertion Sort
2. Selection Sort
3. Bubble Sort
Enter Choice: 2
Sorted Array: 1 6 33 999
```

SET 3:
Enter the number of elements: 3
Enter the elements of the lists: 23 11 898
Unsorted Array: 23 11 898
Choose Sorting Algorithm:
1. Insertion Sort
2. Selection Sort
3. Bubble Sort
Enter Choice: 3
Sorted Array: 11 23 898

PROGRAM 3

Implement Doubly Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

ALGORITHM

1. Class Node:
    1.1 Create a class Node with data, prev, and next pointers.
    1.2 Define a constructor to initialize the data and pointers.
    1.3 Define a destructor to free memory.
2. Function display:
    2.1 Input: head (pointer to the head of the linked list)
    2.2 Initialize a temporary pointer temp to head.
    2.3 While temp is not null:
    2.3.1 Print temp's data.
    2.3.2 Move temp to temp's next.
    2.4 End
3. Function getLength:
    3.1 Input: head (pointer to the head of the linked list)
    3.2 Initialize len to 0.
    3.3 Initialize a temporary pointer temp to head.
    3.4 While temp is not null:
        3.4.1 Increment len.
        3.4.2 Move temp to temp's next.
    3.5 Return len.
    3.6 End
4. Function insertAtHead:
    4.1 Input: head (pointer to the head of the linked list), tail (pointer to the tail of the linked list), data.
    4.2 If head is null:
        4.2.1 Create a new node with data.
        4.2.2 Set head and tail to the new node.
    4.3 Else:
        4.3.1 Create a new node with data.
        4.3.2 Set the new node's next to head and head's prev to the new node.
        4.3.3 Set head to the new node.
    4.4 End
5. Function insertAtTail:
    5.1 Input: tail (pointer to the tail of the linked list), head (pointer to the head of the linked list), data.
    5.2 If tail is null:
        5.2.1 Create a new node with data.
        5.2.2 Set head and tail to the new node.
    5.3 Else:
        5.3.1 Create a new node with data.
        5.3.2 Set tail's next to the new node and the new node's prev to tail.
        5.3.3 Set tail to the new node.
    5.4 End
6. Function insertAtPosition:
    6.1 Input: head (pointer to the head of the linked list), tail (pointer to the tail of the linked list), pos, data.
    6.2 If pos is less than 1 or greater than the length of the list plus 1:
        6.2.1 Print "Invalid position. Insertion failed."

6.3 If pos is 1:
6.3.1 Call insertAtHead with head, tail, and data.
6.4 Else if pos is the length of the list plus 1:
6.4.1 Call insertAtTail with tail, head, and data.
6.5 Else:
6.5.1 Initialize a temporary pointer temp to head.
6.5.2 Iterate pos - 2 times:
6.5.2.1 Move temp to temp's next.
6.5.3 Create a new node with data.
6.5.4 Set the new node's next to temp's next, temp's next's prev to the new node.
6.5.5 Set temp's next to the new node and the new node's prev to temp.
6.6 End
7. Function deleteNode:
7.1 Input: head (pointer to the head of the linked list), pos.
7.2 If pos is less than 1 or greater than the length of the list:
7.2.1 Print "Invalid position. Deletion failed."
7.3 If pos is 1:
7.3.1 Set temp to head.
7.3.2 Set temp's next's prev to null.
7.3.3 Set head to temp's next.
7.3.4 Set temp's next to null.
7.3.5 Free memory for temp.
7.4 Else:
7.4.1 Set cur to head.
7.4.2 Initialize prev to null.
7.4.3 Iterate pos - 1 times:
7.4.3.1 Set prev to cur.
7.4.3.2 Set cur to cur's next.
7.4.4 Set cur's prev's next to cur's next.
7.4.5 If cur's next is not null, set cur's next's prev to cur's prev.
7.4.6 Set cur's next to null.
7.4.7 Free memory for cur.
7.5 End
8. Function searchNode:
8.1 Input: head (pointer to the head of the linked list), target.
8.2 Set temp to head.
8.3 Initialize pos to 1.
8.4 While temp is not null:
8.4.1 If temp's data is equal to the target:
8.4.1.1 Return pos.
8.4.2 Move temp to temp's next.
8.4.3 Increment pos.
8.5 Return -1.
8.6 End
9. Function reverseList:
9.1 Input: head (pointer to the head of the linked list), tail (pointer to the tail of the linked list).
9.2 Set current to head, prevNode to null, nextNode to null.
9.3 While current is not null:
9.3.1 Set nextNode to current's next.
9.3.2 Set current's next to prevNode and current's prev to nextNode.

9.3.3 Set prevNode to current.
9.3.4 Set current to nextNode.
9.4 Set tail to head and head to prevNode.
9.5 End
10. Main program: Give the user choices for performing variousw functions as defined above.

PROGRAM CODE

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
class Node {
	public:
	   T data;
	   Node* prev;
	   Node* next;
	   Node(T data) : data(data), next(nullptr), prev(nullptr) {}
	   ~Node() {
		if (next != nullptr) {
			delete next;
			next = nullptr;
		}
		cout << "Memory freed with data " << data << endl;
	}
};
template <typename T>
void display(Node<T>*& head) {
	Node<T>* temp = head;
	while (temp != nullptr) {
		cout << temp->data << " ";
		temp = temp->next;
	}
	cout << endl;
}
template <typename T>
int getLength(Node<T>* head) {
	int len = 0;
	Node<T>* temp = head;
	while (temp != nullptr) {
		len++;
		temp = temp->next;
	}
	return len;
}
template <typename T>
void insertAtHead(Node<T>*& head, Node<T>*& tail, T data) {
	if (head == nullptr) {
		Node<T>* temp = new Node<T>(data);
		head = temp;
		tail = temp;
	} else {
```

```
                Node<T>* temp = new Node<T>(data);
                temp->next = head;
                head->prev = temp;
                head = temp;
        }
}
template <typename T>
void insertAtTail(Node<T>*& tail, Node<T>*& head, T data) {
        if (tail == nullptr) {
                Node<T>* temp = new Node<T>(data);
                tail = temp;
                head = temp;
        } else {
                Node<T>* temp = new Node<T>(data);
                tail->next = temp;
                temp->prev = tail;
                tail = temp;
        }
}
template <typename T>
void insertAtPosition(Node<T>*& head, Node<T>*& tail, int pos, T data) {
        if (pos < 1 || pos > getLength(head) + 1) {
                cout << "Invalid position. Insertion failed." << endl;
                return;
        }
        if (pos == 1) {
                insertAtHead(head, tail, data);
        } else if (pos == getLength(head) + 1) {
                insertAtTail(tail, head, data);
        } else {
                Node<T>* temp = head;
                for (int i = 1; i < pos - 1; ++i) {
                        temp = temp->next;
                }
                Node<T>* nodeToInsert = new Node<T>(data);
                nodeToInsert->next = temp->next;
                temp->next->prev = nodeToInsert;
                temp->next = nodeToInsert;
                nodeToInsert->prev = temp;
        }
}
template <typename T>
void deleteNode(Node<T>*& head, int pos) {
        if (pos < 1 || pos > getLength(head)) {
                cout << "Invalid position. Deletion failed." << endl;
                return;
        }
        if (pos == 1) {
                Node<T>* temp = head;
                temp->next->prev = nullptr;
                head = temp->next;
                temp->next = nullptr;
```

```
                delete temp;
        } else {
                Node<T>* cur = head;
                for (int i = 1; i < pos; ++i) {
                        cur = cur->next;
                }
                cur->prev->next = cur->next;
                if (cur->next != nullptr) {
                        cur->next->prev = cur->prev;
                }
                cur->next = nullptr;
                delete cur;
        }
}
template <typename T>
int searchNode(Node<T>* head, T target) {
        Node<T>* temp = head;
        int pos = 1;
        while (temp != nullptr) {
                if (temp->data == target) {
                        return pos;
                }
                temp = temp->next;
                pos++;
        }
        return -1;
}
template <typename T>
void reverseList(Node<T>*& head, Node<T>*& tail) {
        Node<T>* current = head;
        Node<T>* prevNode = nullptr;
        Node<T>* nextNode = nullptr;
        while (current != nullptr) {
                nextNode = current->next;
                current->next = prevNode;
                current->prev = nextNode;
                prevNode = current;
                current = nextNode;
        }
        tail = head;
        head = prevNode;
}
int main() {
        Node<int>* headInt = nullptr;
        Node<int>* tailInt = nullptr;
        Node<string>* headString = nullptr;
        Node<string>* tailString = nullptr;
        int choice;
        do {
                cout << "\nSelect an option:\n" << "1. Insert at Head\n"
```

```
<< "2. Insert at Tail\n"  << "3. Insert at Position\n" << "4. Delete a Node\
n"<< "5. Search for a Node\n" << "6. Reverse List\n" << "7. Display\n" <<
"0. Exit\n" << "Enter your choice: ";
cin >> choice;
switch (choice) {
        case 1: {
                int value;
                cout << "Enter the value to insert at the head: ";
                cin >> value;
                insertAtHead(headInt, tailInt, value);
                cout << "Inserted successfully." << endl;
                break;
        }
        case 2: {
                int value;
                cout << "Enter the value to insert at the tail: ";
                cin >> value;
                insertAtTail(tailInt, headInt, value);
                cout << "Inserted successfully." << endl;
                break;
        }
        case 3: {
                int pos;
                int value;
                cout << "Enter the position to insert at: ";
                cin >> pos;
                cout << "Enter the value to insert: ";
                cin >> value;
                insertAtPosition(headInt, tailInt, pos, value);
                cout << "Inserted successfully." << endl;
                break;
        }
        case 4: {
                int pos;
                cout << "Enter the position to delete from: ";
                cin >> pos;
                deleteNode(headInt, pos);
                cout << "Deleted successfully." << endl;
                break;
        }
        case 5: {
                int value;
                cout << "Enter the value to search for: ";
                cin >> value;
                int position = searchNode(headInt, value);
                if (position != -1) {
                        cout << "Value " << value << " found at position " <<
                        position << endl;
                } else {
                        cout << "Value " << value << " not found in the list."
                        << endl;
                        }
```

```
                    break;
                }
                case 6: {
                    cout << "Reversing the list." << endl;
                    reverseList(headInt, tailInt);
                    break;
                }
                case 7:
                    cout << "Linked List Contents: ";
                    display(headInt);
                    break;
                case 0:
                    cout << "Exiting program.\n";
                    break;
                default:
                    cout << "Invalid choice. Please try again.\n";
            }
        } while (choice != 0);
        return 0;
}
```

OUTPUT

```
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 1
Enter the value to insert at the head: 12
Inserted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 1
Enter the value to insert at the head: 31
Inserted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
```

6. Reverse List
7. Display
0. Exit
Enter your choice: 2
Enter the value to insert at the tail: 32
Inserted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 3
Enter the position to insert at: 2
Enter the value to insert: 55
Inserted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 7
Linked List Contents: 31 55 12 32
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 4
Enter the position to delete from: 2
Memory freed with data 55
Deleted successfully.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit

Enter your choice: 7
Linked List Contents: 31 12 32
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 5
Enter the value to search for: 3
Value 3 not found in the list.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 6
Reversing the list.
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 7
Linked List Contents: 32 12 31
Select an option:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete a Node
5. Search for a Node
6. Reverse List
7. Display
0. Exit
Enter your choice: 0
Exiting program.