MANGALORE UNIVERSITY

MASTER OF SCIENCE
IN
COMPUTER SCIENCE

23CSP101: ADVANCED DATA STRUCTURES LAB

SUBMITTED
BY
I SEMESTER MSC
Computer Science Students

SUBMITTED
TO
Mr. Prakasha M.
Department of Computer
Science

Lecturers, In-charge:

1.

2.

Mangalore University

Dept. of Post-Graduate Studies and Research in Computer Science

Mangalagangothri - 574199

ADVANCED DATA STRUCTURES PROGRAMS

PART-A

PART-B

1. Write a C++ program to find the Subsets of a given set S = {S1, S2, …, Sn} of ‘n’ positive integers whose sum is equal to a given positive integer ‘d’.

```
#include <bits/stdc++.h>
using namespace std;

bool flag = 0;
void PrintSubsetSum(int i, int n,  vector<int> arr, int targetSum, vector<int>& subset)
{
        if (targetSum == 0) {
        flag = 1;
        cout << "[ ";
        for (int i = 0; i < subset.size(); i++) {
                cout << subset[i] << " ";
        }
        cout << "]" << endl;
        return;
        }

        if (i == n)
                return;

        if (arr[i] <= targetSum)
        {
                subset.push_back(arr[i]);
                PrintSubsetSum(i + 1, n, arr, targetSum - arr[i],subset);

                subset.pop_back();
        }

        PrintSubsetSum(i + 1, n, arr, targetSum, subset);
}

int main()
{
        int n, data, sum;
        vector<int> arr, subArr;
        cout << "Enter number of elements : ";
        cin >> n;
        cout << "Enter " << n << " elements" << endl;
        for(int i = 0; i < n; i++)
        {
                cin >> data;
```

```
		arr.push_back(data);
	}
	cout << "Enter a sum :";
	cin >> sum;
	cout << "Subset:" << endl;
	PrintSubsetSum(0, n, arr, sum, subArr);
	if(!flag)
		cout << "Subset not found" << endl;
}
```

<u>Output :</u>

```
Enter number of elements : 6
3
4
2
1
8
6
Enter a sum :9
Subset:
[ 3 4 2 ]
[ 3 6 ]
[ 2 1 6 ]
[ 1 8 ]
```

2. Write a C++ program to store k keys into an array of size n at the location computed using a hash function, loc = key % n, where k<=n and key takes values from [1 to m], m > n. Handle the collision using Linear probing technique.

```
#include<iostream>
using namespace std;

void hashInsert(int *arr, int size, int element)
{
    int n = 0;
    int position = element % size;
    while(arr[position] != INT_MIN && n != size)
    {
        position = (position + 1) % size;
        n++;
    }
    if(n != size)
        arr[position] = element;
    else
        cout << "Table is full. We cannot insert values anymore." << endl;
}

void hashDisplay(int arr[], int size)
{
    cout << "Index\tValue" << endl;
    for(int i = 0; i < size; i++)
    {
        cout << i << "\t";
        if(arr[i] == INT_MIN)
            cout << " " << endl;
        else
            cout << arr[i] << endl;
    }
}

int main()
{
    int size, choice, element;
    cout << "Enter the size of hash table :: ";
    cin >> size;
```

```
        int arr[size];

        for(int i =0; i < size; i++)
                arr[i] = INT_MIN;

        while(1)
        {
                cout << "1. Insert\n2. Display\n3. Exit" << endl;
                cout << "Enter your choice :: ";
                cin >> choice;
                switch(choice)
                {
                        case 1:
                                cout << "Enter a value to insert :: ";
                                cin >> element;
                                hashInsert(arr, size, element);
                                break;
                        case 2:
                                hashDisplay(arr, size);
                                break;
                        case 3:
                                exit(0);
                                break;
                        default:
                                cout << "Invalid choice. Please eneter valid choice" <<
                        endl;
                }
        }
}
```

Output:

```
Enter the size of hash table :: 11
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 20
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 30
```

```
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 2
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 13
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 25
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 24
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 10
1. Insert
2. Display
3. Exit
Enter your choice :: 1
Enter a value to insert :: 9
1. Insert
2. Display
3. Exit
Enter your choice :: 2
Index   Value
0     9
1
2     2
3     13
4     25
5     24
6
7
```

```
8       30
9       20
10      10
1. Insert
2. Display
3. Exit
Enter your choice :: 3
```

3. Write a C++ program to implement Merge Sort technique using Divide and Conquer method.

```
#include <iostream>
using namespace std;

void Merge(int *a, int low, int high, int mid)
{
	int i, j, k, temp[high - low + 1];
	i = low;
	k = 0;
	j = mid + 1;

	while (i <= mid && j <= high)
	{
		if (a[i] < a[j])
		{
			temp[k] = a[i];
			k++;
			i++;
		}
		else
		{
			temp[k] = a[j];
			k++;
			j++;
		}
	}

	while (i <= mid)
	{
		temp[k] = a[i];
		k++;
		i++;
	}

	while (j <= high)
	{
		temp[k] = a[j];
		k++;
		j++;
	}
	for (i = low; i <= high; i++)
```

```
        a[i] = temp[i - low];
}

void MergeSort(int *a, int low, int high)
{
    int mid;

    if (low < high)
    {
        mid = (low + high) / 2;
        MergeSort(a, low, mid);
        MergeSort(a, mid + 1, high);
        Merge(a, low, high, mid);
    }
}

int main()
{
    int n, i;

    cout << "\nEnter the number of elements : ";
    cin >> n;

    int arr[n];

    cout << "Enter " << n << " elements " << endl;
    for (i = 0; i < n; i++)
        cin >> arr[i];

    MergeSort(arr, 0, n - 1);

    cout << "\nSorted Data: ";
    for (i = 0; i < n; i++)
        cout << " " << arr[i];

    return 0;
}
```

Output:

```
Enter the number of elements : 8
Enter 8 elements
5
8
12
0
3
1
6
7

Sorted Data:  0 1 3 5 6 7 8 12
```

4. Find the minimum cost spanning tree of a given weighted undirected graph using Prim's Algorithm.

```
#include <bits/stdc++.h>
using namespace std;

#define V 5

int minKey(int key[], bool mstSet[])
{

        int min = INT_MAX, min_index;

        for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
                min = key[v], min_index = v;

        return min_index;
}

void printMST(int parent[], int graph[V][V])
{
        cout << "Edge \tWeight\n";
        for (int i = 1; i < V; i++)
                cout << parent[i] << " - " << i << " \t"<< graph[i][parent[i]] << " \n";
}

void primMST(int graph[V][V])
{
        int parent[V];
        int key[V];
        bool mstSet[V];

        for (int i = 0; i < V; i++)
        {
                key[i] = INT_MAX;
                mstSet[i] = false;
        }

        key[0] = 0;

        parent[0] = -1;
        for (int count = 0; count < V - 1; count++)
        {
                int u = minKey(key, mstSet);
                mstSet[u] = true;
                for (int v = 0; v < V; v++)
                {
                        if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                                parent[v] = u, key[v] = graph[u][v];
                }
        }
        printMST(parent, graph);
}

int main()
{
        int graph[V][V] = { { 0, 2, 0, 6, 0 },
                                { 2, 0, 3, 8, 5 },
                                { 0, 3, 0, 0, 7 },
                                { 6, 8, 0, 0, 9 },
                                { 0, 5, 7, 9, 0 } };

        primMST(graph);

        return 0;
}
```

Output:

```
Edge   Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

5. Write a C++ program to implement Depth First Search (DFS) for an undirected graph.

```
#include <bits/stdc++.h>
using namespace std;

class Graph
{
public:
	map<int, bool> visited;
	map<int, list<int> > adj;

	void addEdge(int v, int w);

	void DFS(int v);
};

void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w);
}

void Graph::DFS(int v)
{
	stack<int> stack;
	stack.push(v);

	list<int>::iterator i;
	while (!stack.empty())
	{
		v = stack.top();
		stack.pop();
		if (!visited[v])
		{
			cout << v << " ";
			visited[v] = true;
		}
		for (i = adj[v].begin(); i != adj[v].end(); ++i)
		{
			if (!visited[*i])
				stack.push(*i);
		}
	}
}

int main()
{
        int e, u, v;
        Graph g;
        cout << "Enter no of edges : ";
        cin >> e;
        for(int i = 0; i < e; i++)
        {
                cout<<"Enter from"<<endl;
                cin>>u;
                cout<<"Enter To"<<endl;
                cin>>v;
                g.addEdge(u,v);
                g.addEdge(v,u);
        }

        cout << "Enter the starting index : ";
        int startIndex;
        cin >> startIndex;
        cout << "Following is Depth First Traversal" << endl;
        g.DFS(startIndex);

        return 0;
}
```

Output:

```
Enter no of edges : 5
Enter from
0
Enter To
1
Enter from
2
Enter To
0
Enter from
0
Enter To
3
Enter from
```

```
2
Enter To
3
Enter from
4
Enter To
2
Enter the starting index : 0
Following is Depth First Traversal
0 3 2 4 1
```

6. Write a C++ program to implement insertion on Min Heap.

```
#include<iostream>
using namespace std;

void minHeapify(int* arr, int n, int i)
{
        int largest = i;
        int leftChild = 2 * i;
        int rightChild = 2 * i + 1;

        if(leftChild <= n && arr[leftChild] < arr[largest])
                largest = leftChild;

        if(rightChild <= n && arr[rightChild] < arr[largest])
                largest = rightChild;

        if(largest != i)
        {
                swap(arr[i], arr[largest]);
                minHeapify(arr, n, largest);
        }
}

int main()
{
        int n;

        cout << "Enter the number of elements : ";
        cin >>n;
        int numArr[n+1];
        numArr[0] = 0;

        cout << "Enter " << n << " elements" << endl;
        for (int i = 1; i <= n; i++)
                cin >> numArr[i];
        cout << "Original Array" << endl;
        for(int i = 1; i <= n; i++)
                cout << numArr[i] << " ";
        cout << endl;

        for(int i = n / 2; i > 0; i--)
                minHeapify(numArr, n, i);
```

```
	cout << "Min heap" << endl;
	for(int i = 1; i <= n; i++)
		cout << numArr[i] << " ";
	cout << endl;
}
```

Output:

```
Enter the number of elements : 7
Enter 7 elements
12
15
7
6
10
5
17
Original Array
12 15 7 6 10 5 17
Min heap
5 6 7 15 10 12 17
```

7. Write a C++ program to Sort element using Max Heap Sort.

```
#include<iostream>
using namespace std;

void maxHeapify(int* arr, int n, int i)
{
        int largest = i;
        int leftChild = 2 * i;
        int rightChild = 2 * i + 1;

        if(leftChild <= n && arr[leftChild] > arr[largest])
                largest = leftChild;

        if(rightChild <= n && arr[rightChild] > arr[largest])
                largest = rightChild;

        if(largest != i)
        {
                swap(arr[i], arr[largest]);
                maxHeapify(arr, n, largest);
        }
}

void heapSort(int* arr, int n)
{
        for(int i = n / 2; i > 0; i--)
                maxHeapify(arr, n, i);

        for(int i = n; i > 0; i--)
        {
                swap(arr[1], arr[i]);
                maxHeapify(arr, i - 1, 1);
        }
}

int main()
{
        int n;

        cout << "Enter the number of elements : ";
        cin >>n;
        int numArray[n+1];
        numArray[0] = 0;

        cout << "Enter " << n << " elements" << endl;
        for (int i = 1; i <= n; i++)
                cin >> numArray[i];
        cout << "Original Array" << endl;
        for(int i = 1; i <= n; i++)
                cout << numArray[i] << " ";
        cout << endl;

        for(int i = n / 2; i > 0; i--)
                maxHeapify(numArray, n, i);

        cout << "Max heap" << endl;
        for(int i = 1; i <= n; i++)
                cout << numArray[i] << " ";
        cout << endl;

        heapSort(numArray, n);
        cout << "Sorted Array : ";
        for(int i = 1; i <= n; i++)
                cout << numArray[i] << " ";
        cout << endl;
}
```

Output:

```
Enter the number of elements : 6
Enter 6 elements
4
8
12
9
3
1
Original Array
4 8 12 9 3 1
Max heap
12 9 4 8 3 1
Sorted Array : 1 3 4 8 9 12
```

8. Write a C++ program to find Minimum and Maximum element from an unsorted array using Divide and Conquer method.

```
#include <iostream>
using namespace std;

void MinMax(int arr[], int low, int high, int &min, int &max)
{
	if (low == high)
	{
		if (max < arr[low])
			max = arr[low];

		if (min > arr[high])
			min = arr[high];
		return;
	}

	if (high - low == 1)
	{
		if (arr[low] < arr[high])
		{
			if (min > arr[low])
				min = arr[low];
			if (max < arr[high])
				max = arr[high];
		}
		else
		{
			if (min > arr[high])
				min = arr[high];
			if (max < arr[low])
				max = arr[low];
		}
		return;
	}
	int mid = (low + high) / 2;
	MinMax(arr, low, mid, min, max);
	MinMax(arr, mid + 1, high, min, max);
}


int main()
```

```
{
        int i, n;
        cout<<"Enter the number of elements : ";
        cin>>n;
        int arr[n];

        for( i = 0; i < n; i++ )
        {
                cout<<"Enter the element : ";
                cin>>arr[i];
        }

        int max = arr[0], min = arr[0];
        MinMax(arr, 0, n - 1, min, max);
        cout<<"The minimum array element is "<<min<<endl;
        cout<<"The maximum array element is "<<max;
}
```

Output:

```
Enter the number of elements : 6
Enter the element : 4
Enter the element : 7
Enter the element : 1
Enter the element : 8
Enter the element : 9
Enter the element : 5
The minimum array element is 1
The maximum array element is 9
```

9. Write a C++ program for implementing Singly Linked list.

```
#include<iostream>
using namespace std;

struct Node
{
   int value;
   struct Node * next;
};

class LinkedList
{
public:
        LinkedList()
        {
                head = tail = NULL;
        }

        Node* createNode(int value)
        {
                Node *node = new Node;
                node->value = value;
                node->next = NULL;
                return node;
        }

        void insertFirst(int value)
        {
                Node *newNode = createNode(value);
                if(head == NULL)
                {
                        head = newNode;
                        tail = newNode;
                }
                else
```

```
		{
			newNode->next = head;
			head = newNode;
		}
		display();
	}

	void insertLast(int value)
	{
		Node *newNode = createNode(value);
		if(head == NULL)
		{
			head = newNode;
			tail = newNode;
		}
		else
		{
			tail->next = newNode;
			tail = newNode;
		}
		display();
	}

	void insertAtPosition(int value, int pos)
	{
		Node *newNode = createNode(value);
		Node *current = head;
		Node *prev = head;
		if(head == NULL)
		{
			head = newNode;
			tail = newNode;
		}
		else if(pos == 1)
		{
			insertFirst(value);
		}
```

```
    else
    {
        for( int i = 1; i < pos && current != NULL; i++)
        {
            prev = current;
            current = current->next;
        }

        if(current == NULL)
        {
            cout << "Position not found" << endl;
        }
        else
        {
            newNode->next = current;
            prev->next = newNode;
            display();
        }
    }
}

void deleteFirst()
{
    Node *current = head;
    if(head == NULL)
    {
        cout << "Linked List is empty" << endl;
    }
    else if(head == tail)
    {
        head = tail = NULL;
        delete head;
    }
    else
    {
        head = head->next;
        delete current;
```

```
		}
		display();
	}

	void deleteLast()
	{
		Node *current = head;
		Node *prev = NULL;
		if(head == NULL)
		{
			cout << "Linked List is empty" << endl;
		}
		else if(head == tail)
		{
			delete head;
			head = tail = NULL;
		}
		else
		{
			while(current->next != NULL)
			{
				prev = current;
				current = current->next;
			}
			tail = prev;
			delete current;
			prev->next = NULL;
		}
		display();
	}

	void deleteAtPosition(int pos)
	{
		Node *current = head;
		Node *prev = NULL;
		if(head == NULL)
		{
			cout << "Linked List is empty" << endl;
		}

		else if(pos == 1)
		{
			deleteFirst();
		}
		else
		{
			for( int i = 1; i < pos && current != NULL; i++)
			{
				prev = current;
				current = current->next;
			}
			if(current == NULL)
				cout << "Position not found" << endl;
			else
			{
				prev->next = current->next;
				delete current;
			}
			display();
		}
	}

	void display()
	{
		Node *current = head;

		if(head == NULL)
			cout << "Linked List is empty" << endl;
		else
		{
			while (current != NULL)
			{
				cout << current->value << "->";
				current = current->next;
```

```
                }
                cout << "NULL" << endl;
            }
        }

private:
        Node *head, *tail;
};

int main()
{
        int choice, data, pos;
        LinkedList ll;

        while(1)
        {
                cout << "1. Insert First" << endl;
                cout << "2. Insert Last" << endl;
                cout << "3. Insert At Desired Position" << endl;
                cout << "4. Delete First" << endl;
                cout << "5. Delete Last" << endl;
                cout << "6. Delete At Desired Position" << endl;
                cout << "7. Display" << endl;
                cout << "8. Exit" << endl;
                cout << "Enter your choice : ";
                cin >> choice;
                switch(choice)
                {
                        case 1:
                                cout << "Enter the value : ";
                                cin >> data;
                                ll.insertFirst(data);
                                break;
                        case 2:
                                cout << "Enter the value : ";
                                cin >> data;
                                ll.insertLast(data);
                                break;
                        case 3:
                                cout << "Enter the value : ";
                                cin >> data;
                                cout << "Enter the position : ";
                                cin >> pos;
                                ll.insertAtPosition(data, pos);
                                break;
                        case 4:
                                ll.deleteFirst();
                                break;
                        case 5:
                                ll.deleteLast();
                                break;
                        case 6:
                                cout << "Enter the position : ";
                                cin >> pos;
                                ll.deleteAtPosition(pos);
                                break;
                        case 7:
                                ll.display();
                                break;
                        case 8:
                                exit(0);
                        default:
                                cout<< "Invalid choice" << endl;
                                break;
                }
        }
}
```

Output:

```
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
```

```
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 1
Enter the value : 20
20->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 2
Enter the value : 30
20->30->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 3
Enter the value : 10
Enter the position : 2
20->10->30->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
```

```
8. Exit
Enter your choice : 2
Enter the value : 50
20->10->30->50->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 4
10->30->50->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 6
Enter the position : 2
10->50->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 5
10->NULL
1. Insert First
2. Insert Last
```

```
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 7
10->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 8
```

10. Write a C++ program for implementing Doubly Linked list.

```
#include<iostream>
using namespace std;

struct Node
{
	int value;
	struct Node * rLink;
	struct Node *lLink;
};

class LinkedList
{
public:
	LinkedList()
	{
		head = tail = NULL;
	}

	Node* createNode(int value)
	{
		Node *node = new Node;
		node->value = value;
		node->lLink = NULL;
		node->rLink = NULL;
		return node;
	}

	void insertFirst(int value)
	{
		Node *newNode = createNode(value);
		if(head == NULL)
		{
			head = newNode;
			tail = newNode;
		}
		else
		{
			head->lLink = newNode;
			newNode->rLink = head;
			head = newNode;
		}
		display();
	}

	void insertLast(int value)
	{
		Node *newNode = createNode(value);
		if(head == NULL)
		{
			head = newNode;
			tail = newNode;
		}
		else
		{
			newNode->lLink = tail;
			tail->rLink = newNode;
			tail = newNode;
		}
		display();
	}

	void insertAtPosition(int value, int pos)
	{
		Node *newNode = createNode(value);
		Node *current = head;
		Node *prev = NULL;
		if(head == NULL)
		{
			head = newNode;
			tail = newNode;
		}
		else if(pos == 1)
		{
			insertFirst(value);
		}
		else
		{
			for( int i = 1; i < pos && current != NULL; i++)
			{
				prev = current;
				current = current->rLink;
			}

			if(current == NULL)
			{
				cout << "Position not found" << endl;
			}
			else
			{
				newNode->lLink = prev;
				newNode->rLink = current;
				prev->rLink = newNode;
				current->lLink = newNode;
				display();
			}
		}
	}

	void deleteFirst()
	{
		Node *current = head;
		if(head == NULL)
		{
			cout << "Linked List is empty" << endl;
		}
		else if(head == tail)
		{
			head = tail = NULL;
			delete head;
		}
		else
		{
			head->rLink->lLink = NULL;
			head = head->rLink;
			delete current;
		}
		display();
	}

	void deleteLast()
	{
		Node *current = head;
		Node *prev = NULL;
		if(head == NULL)
		{
			cout << "Linked List is empty" << endl;
		}
		else if(head == tail)
		{
			delete head;
			head = tail = NULL;
		}
		else
		{
			while(current->rLink != NULL)
			{
				prev = current;
				current = current->rLink;
			}
			tail = prev;
			delete current;
			prev->rLink = NULL;
		}
		display();
	}
```

```
void deleteAtPosition(int pos)
{
        Node *current = head;
        Node *prev = NULL;
        if(head == NULL)
        {
                cout << "Linked List is empty" << endl;
        }

        else if(pos == 1)
        {
                deleteFirst();
        }
        else
        {
                for( int i = 1; i < pos && current != NULL; i++)
                {
                        prev = current;
                        current = current->rLink;
                }
                if(current == NULL)
                {
                        cout << "Position not found" << endl;
                }
                else
                {
                        current->rLink->lLink = prev;
                        prev->rLink = current->rLink;
                        delete current;
                }
                display();
        }
}

void display()
{
        Node *current = head;
```

```
                if(head == NULL)
                {
                        cout << "Linked List is empty" << endl;
                }
                else
                {
                        while (current != NULL)
                        {
                                cout << current->value << "->";
                                current = current->rLink;
                        }
                        cout << "NULL" << endl;
                   }
        }

private:
        Node *head, *tail;
};

int main()
{
        int choice, data, pos;
        LinkedList ll;

        while(1)
        {
                cout << "1. Insert First" << endl;
                cout << "2. Insert Last" << endl;
                cout << "3. Insert At Desired Position" << endl;
                cout << "4. Delete First" << endl;
                cout << "5. Delete Last" << endl;
                cout << "6. Delete At Desired Position" << endl;
                cout << "7. Display" << endl;
                cout << "8. Exit" << endl;
                cout << "Enter your choice : ";
                cin >> choice;
                switch(choice)
```

```
{
        case 1:
                cout << "Enter the value : ";
                cin >> data;
                ll.insertFirst(data);
                break;
        case 2:
                cout << "Enter the value : ";
                cin >> data;
                ll.insertLast(data);
                break;
        case 3:
                cout << "Enter the value : ";
                cin >> data;
                cout << "Enter the position : ";
                cin >> pos;
                ll.insertAtPosition(data, pos);
                break;
        case 4:
                ll.deleteFirst();
                break;
        case 5:
                ll.deleteLast();
                break;
        case 6:
                cout << "Enter the position : ";
                cin >> pos;
                ll.deleteAtPosition(pos);
                break;
        case 7:
                ll.display();
                break;
        case 8:
                exit(0);
        default:
                cout<< "Invalid choice" << endl;
                break;
```

```
                    }
          }
}
```

Output:

```
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 1
Enter the value : 10
10->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 2
Enter the value : 30
10->30->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 3
Enter the value : 20
Enter the position : 2
10->20->30->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 4
20->30->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 5
20->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 6
Enter the position : 1
Linked List is empty
1. Insert First
2. Insert Last
3. Insert At Desired Position
```

```
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 1
Enter the value : 20
20->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 7
20->NULL
1. Insert First
2. Insert Last
3. Insert At Desired Position
4. Delete First
5. Delete Last
6. Delete At Desired Position
7. Display
8. Exit
Enter your choice : 8
```

11. Write a C++ program to split the linked list into two halves such that the element ‘e’ should be the first element of second list.

```
#include<iostream>
using namespace std;

struct Node
{
	int value;
	struct Node * next;
};

class LinkedList
{
public:
	LinkedList()
	{
		head = NULL;
		tail = NULL;
	}

	void createNode(int value)
	{
		Node *temp = new Node;
		temp->value = value;
		temp->next = NULL ;

		if (head == NULL)
		{
			head = temp;
			tail = temp;
		}
		else
		{
			tail->next = temp;
			tail = temp;
		}
```

```
}

void printList()
{
        Node * current = head;

        while (current != NULL)
        {
                cout << current->value << "->";
                current = current->next;
        }
        cout << "NULL" << endl;
}

int search(int data)
{
        Node *current = head;
        int position = 0;
        while(current != NULL)
        {
                position++;
                if(current->value == data)
                        return position;
                else
                        current = current->next;}
        }
        return -1;
}

void split(LinkedList list, int data)
{
        Node *current = head;
        Node *prev = head;
        LinkedList list2;
        while(current != NULL)
        {
                if(current->value == data)
```

```
            {
                while(current != NULL)
                {
                    list2.createNode(current->value);
                    current = current->next;
                }

                list.tail = prev;
                list.tail->next = NULL;
            }
            else
            {
                prev = current;
                current = current->next;
            }
        }

        cout << "Linked List 1 : ";
        list.printList();
        cout << "Linked List 2 : ";
        list2.printList();
    }

private:
    Node *head;
    Node *tail;
};

int main()
{
    int n, data,splitElement,position;
    LinkedList ll;
    cout << "Enter number of nodes : ";
    cin >> n;
    for(int  i = 0; i < n; i++)
    {
        cout << "Enter the data for node " << i + 1 << " : ";
```

```
            cin >> data;
            ll.createNode(data);
        }

        cout << "Linked List" << endl;
        ll.printList();

        cout << "Enter the data to split : " << endl;
        cin >> splitElement;
        position = ll.search(splitElement);
        if(position > 0)
        {
            cout << "Element " << splitElement << " found in position " << position
            << endl;
            ll.split(ll, splitElement);
        }
        else
            cout << "Element " << splitElement << " not found" << endl;
}
```

<u>Output:</u>

```
Enter number of nodes : 5
Enter the data for node 1 : 10
Enter the data for node 2 : 20
Enter the data for node 3 : 30
Enter the data for node 4 : 40
Enter the data for node 5 : 50
Linked List
10->20->30->40->50->NULL
Enter the data to split :
20
Element 20 found in position 2
Linked List 1 : 10->NULL
Linked List 2 : 20->30->40->50->NULL
```

12. Write a C++ program to split the linked list into two halves such that the element ‘e’ should be the last element of first list.

```
#include<iostream>
using namespace std;

struct Node
{
        int value;
        struct Node * next;
};

class LinkedList
{
public:
        LinkedList()
        {
                head = NULL;
                tail = NULL;
        }

        void createNode(int value)
        {
                Node *temp = new Node;
                temp->value = value;
                temp->next = NULL ;

                if (head == NULL)
                {
                        head = temp;
                        tail = temp;
                }
                else
                {
                        tail->next = temp;
                        tail = temp;
                }
```

```
}

void printList()
{
        Node * current = head;
        while (current != NULL)
        {
                cout << current->value << "->";
                current = current->next;
        }
        cout << "NULL" << endl;
}

int search(int data)
{
        Node *current = head;
        int position = 0;
        while(current != NULL)
        {
                position++;
                if(current->value == data)
                        return position;
                else
                        current = current->next;
        }
        return -1;
}

void split(LinkedList &list, int data)
{
        Node *current = head;
        Node *prev = head;
        LinkedList list2;
        while(current != NULL)
        {
                if(current->value == data)
                {
```

```
                current = current->next;
                while(current != NULL)
                {
                    list2.createNode(current->value);
                    current = current->next;
                }

                list.tail = prev->next;
                list.tail->next = NULL;
            }
            else
            {
                prev = current;
                current = current->next;
            }
        }
        cout << "Linked List 1 : ";
        list.printList();
        cout << "Linked List 2 : ";
        list2.printList();
    }

private:
    Node *head;
    Node *tail;
};

int main()
{
    int n, data,splitElement,position;
    LinkedList ll;
    cout << "Enter number of nodes : ";
    cin >> n;
    for(int  i = 0; i < n; i++)
    {
        cout << "Enter the data for node " << i + 1 << " : ";
        cin >> data;
        ll.createNode(data);
    }

    cout << "Linked List" << endl;
    ll.printList();

    cout << "Enter the data to split : " << endl;
    cin >> splitElement;
    position = ll.search(splitElement);
    if(position > 0)
    {
        cout << "Element " << splitElement << " found in position " << position
        << endl;
        ll.split(ll, splitElement);
    }
    else
        cout << "Element " << splitElement << " not found" << endl;
}
```

<u>Output:</u>

```
Enter number of nodes : 6
Enter the data for node 1 : 10
Enter the data for node 2 : 20
Enter the data for node 3 : 30
Enter the data for node 4 : 40
Enter the data for node 5 : 50
Enter the data for node 6 : 60
Linked List
10->20->30->40->50->60->NULL
Enter the data to split :
40
Element 40 found in position 4
Linked List 1 : 10->20->30->40->NULL
Linked List 2 : 50->60->NULL
```

13. Construct a Binary Search Tree (BST) and insert an element into a binary search tree. Display the tree using in order, preorder and post order traversal methods.

```
#include<iostream>
using namespace std;

struct Node
{
        int value;
        struct Node * left;
struct Node * right;
};

class BinarySearchTree
{
public:
        Node* createNode(int value)
        {
                Node *node = new Node;
                node->value = value;
                node->left = node->right = NULL;
                return node;
        }
        Node* insert(Node* node, int value)
        {
                if(node == NULL
                        return createNode(value);
                if(value <= node->value)
                        node->left = insert(node->left, value);
                else if(value > node->value)
                        node->right = insert(node->right, value);
                return node;
        }
        void preorder(Node* node)
        {
                if(node != NULL)
                {
                        cout << node->value << " ";
                        preorder(node->left);
                        preorder(node->right);
                }
        }
        void inorder(Node* node)
        {
                if(node != NULL)
                {
                        inorder(node->left);
                        cout << node->value << " ";
                        inorder(node->right);
                }
        }
        void postorder(Node* node)
        {
                if(node != NULL)
                {
                        postorder(node->left);
                        postorder(node->right);
                        cout << node->value << " ";
                }
        }
};

int main()
{
        int n;
        Node * root = NULL;
        BinarySearchTree bst;
        cout << "Enter number of elements : ";
        cin >> n;
        cout << "Enter " << n << " elements" << endl;
        for(int i = 0; i < n; i++)
        {
                int value;
```

```
                    cin >> value;
                    root = bst.insert(root, value);
          }
          cout<< "Preorder traversal : ";
          bst.preorder(root);
          cout << endl;
          cout<< "Inorder traversal : ";
          bst.inorder(root);
          cout << endl;
          cout<< "Postorder traversal : ";
          bst.postorder(root);
          cout << endl;
}
```

Output:

```
Enter number of elements : 6
Enter 6 elements
5
7
6
1
8
2
Preorder traversal : 5 1 2 7 6 8
Inorder traversal : 1 2 5 6 7 8
Postorder traversal : 2 1 6 8 7 5
```

14. Write a C++ program for solving the N-Queen’s Problem using backtracking.

```
#include <bits/stdc++.h>
using namespace std;
bool isPlace(vector<int> &board, int current, int col)
{
        for (int j = 0; j < current; j += 1)
        {
                if (board[j] == col || abs(board[j] - col) == abs(j - current))
                        return false;
        }
        return true;
}
void prettyPrint(vector<int> &board)
{
        for (int i = 0; i < board.size(); i += 1)
        {
                cout << endl;
                for (int j = 0; j < board.size(); j += 1)
                {
                        if (board[i] == j)
                                cout << "Q" << " ";
                        else
                                cout <<  "* ";
                }
        }
        cout << endl;
}
void nqueen(vector<int> &board, int current, int queens, int &count)
{
        for (int col = 0; col < board.size(); col += 1)
        {
                if (isPlace(board, current, col))
                {
                        board[current] = col;
                        if (current == queens-1)
                        {
                                prettyPrint(board);
                                count++;
                        }
                        else
                                nqueen(board, current + 1, queens, count);
                }
        }
}
int main()
{
        int queens;
        cout << "Enter the number of queens to consider : ";
        cin >> queens;
        vector<int> board(queens, 0);
        int count = 0;
        nqueen(board, 0, queens, count);
        cout<<"\n\nThere are "<<count<<" possible solutions.";
        return 0;
}
```

Output:

```
Enter the number of queens to consider : 4
* Q * *
* * * Q
Q * * *
* * Q *

* * Q *
Q * * *
* * * Q
* Q * *
There are 2 possible solutions.
```

15. Write a C++ program to count Number of connected components in an undirected graph.

```
#include <bits/stdc++.h>
using namespace std;
class Graph
{
        int V;
        list<int>* adj;
        void DFSUtil(int v, bool visited[]);
        public:
        Graph(int V);
        void addEdge(int v, int w);
        int NumberOfconnectedComponents();
};
int Graph::NumberOfconnectedComponents()
{
        bool* visited = new bool[V];
        int count = 0;
        for (int v = 0; v < V; v++)
        visited[v] = false;
        for (int v = 0; v < V; v++)
        {
                if (visited[v] == false)
                {
                        DFSUtil(v, visited);
                        count += 1;
                }
        }
        return count;
}
void Graph::DFSUtil(int v, bool visited[])
{
        visited[v] = true;
        list<int>::iterator i;
        for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
```

```
                    DFSUtil(*i, visited);
}
Graph::Graph(int V)
{
          this->V = V;
          adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
          adj[v].push_back(w);
          adj[w].push_back(v);
}

int main()
{
          Graph g(7);
          g.addEdge(0, 1);
          g.addEdge(2, 4);
          g.addEdge(2, 3);
          g.addEdge(3, 4);
          g.addEdge(6, 5);

          cout << "Number of connected componets :: " <<
          g.NumberOfconnectedComponents();
          return 0;
}
```

Output:

```
Number of connected componets :: 3
```

16. Write a C++ program to implement Breadth First Search (BFS) for an undirected graph.

```
#include <bits/stdc++.h>
using namespace std;
class Graph
{
public:
        map<int, bool> visited;
        map<int, list<int> > adj;
        void addEdge(int v, int w);
        void BFS(int v);
};

void Graph::addEdge(int v, int w)
{
        adj[v].push_back(w);
}

void Graph::BFS(int v)
{
        list<int> queue;
        queue.push_back(v);
        visited[v] = true;
        list<int>::iterator i;
        while (!queue.empty())
        {
                int currVertex = queue.front();
                cout << currVertex << " ";
                queue.pop_front();

                for (i = adj[currVertex].begin(); i != adj[currVertex].end(); ++i)
                {
                        if (!visited[*i])
                        {
                                visited[*i] = true;
                        queue.push_back(*i);
                    }
                }
            }
        }

        int main()
        {
            int e, u, v;
            Graph g;
            cout << "Enter no of edges : ";
            cin >> e;
            for(int i = 0; i < e; i++)
            {
                cout<<"Enter from"<<endl;
                cin>>u;
                cout<<"Enter To"<<endl;
                cin>>v;
                g.addEdge(u,v);
                g.addEdge(v,u);
            }

            cout << "Enter the starting index : ";
            int startIndex;
            cin >> startIndex;
            cout << "Following is Breadth First Traversal" << endl;
            g.BFS(startIndex);

            return 0;
        }
```

Output:

```
Enter no of edges : 5
Enter from
1
Enter To
2
Enter from
1
Enter To
5
Enter from
5
Enter To
7
Enter from
3
Enter To
7
Enter from
1
Enter To
7
Enter the starting index : 1
Following is Breadth First Traversal
1 2 5 7 3
```

17. Find the minimum cost spanning tree of a given undirected graph using Kruskal’s Algorithm.

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> iPair;

class Graph
{
        int V, E;
        vector< pair<int, iPair> > edges;

public:
        Graph(int V, int E)
        {
                this->V = V;
                this->E = E;
        }

        void addEdge(int u, int v, int w)
        {
                edges.push_back({w, {u, v}});
        }

        int kruskalMST();
};

class DisjointSets
{
        int *parent, *rank;
        int n;
public:
        DisjointSets(int n)
        {
                this->n = n;
                parent = new int[n + 1];
                rank = new int[n + 1];
        for (int i = 0; i <= n; i++)
        {
            rank[i] = 0;
            parent[i] = i;
        }
    }

    int find(int u)
    {
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    void merge(int x, int y)
    {
        x = find(x), y = find(y);
        if (rank[x] > rank[y])
            parent[y] = x;
        else
            parent[x] = y;

        if (rank[x] == rank[y])
            rank[y]++;
    }
};

int Graph::kruskalMST()
{
    int mst_wt = 0;
    sort(edges.begin(), edges.end());
    DisjointSets ds(V);
    vector< pair<int, iPair> >::iterator it;

    for (it = edges.begin(); it != edges.end(); it++)
    {
        int u = it->second.first;
        int v = it->second.second;
        int set_u = ds.find(u);
        int set_v = ds.find(v);
        if (set_u != set_v)
        {
            cout << u << " - " << v << endl;
            mst_wt += it->first;
            ds.merge(set_u, set_v);
        }
    }
    return mst_wt;
}

int main()
{
    int V = 9, E = 14;
    Graph g(V, E);
    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    cout << "Edges of MST are \n";
    int mst_wt = g.kruskalMST();
    cout << "\nWeight of MST is " << mst_wt;
    return 0;
}
```

Output:

```
Edges of MST are
6 - 7
2 - 8
5 - 6
0 - 1
2 - 5
2 - 3
0 - 7
3 - 4

Weight of MST is 37
```