Name : Rajeshwari Shivaji Jadhav

Class : MCA (III sem)

RollNo: 27

Practical 1:

Write a program for creating Max Heap using INSERT

```
#include<iostream>
using namespace std;
class InsertMaxHeap
{
    int n;
    int a[20];
public:
    void insert(int a[], int n);
    void get();
    void show();
};
void InsertMaxHeap::get()
{
    cout << "Enter how many element insert into heap : ";
    cin >> n;
    cout << "Enter heap element : \n";
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        insert(a, i);
    }
}
void InsertMaxHeap::insert(int a[], int n)
{
    int i, j, item;
    j = n;
    i = n / 2;
    item = a[n];
    while (i > 0 && a[i] < item)
    {
        a[j] = a[i];
        j = i;
        i = i / 2;
    }
    a[j] = item;
}
void InsertMaxHeap::show()
{
    cout << "Max heap using insert :\n";
    for (int i = 1; i <= n; i++)
    {
        cout << a[i];
        cout << "\t";
    }
}
int main()
{
    InsertMaxHeap obj;
    obj.get();
    obj.show();
    return 0;
}
// Output :
```

```
Enter how many element insert into heap:7
Enter heap element:
40 80 35 90 45 50 70
Max heap using insert:
90      80      70      40      45      35      50
```

```
//Write a program for creating Min Heap using INSERT


#include<iostream>
```

```
    using namespace std;
class InsertMinHeap
{
    int n;
    int a[20];
public:
    void insert(int a[], int n);
    void get();
    void show();
};
void InsertMinHeap::get()
{
    cout << "Enter how many element insert into heap:";
    cin >> n;
    cout << "Enter heap element:\n";
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        insert(a, i);
    }
}
void InsertMinHeap::insert(int a[], int n)
{
    int i, j, item;
    j = n;
    i = n / 2;
    item = a[n];
    while (i > 0 && a[i] > item)
    {
        a[j] = a[i];
        j = i;
        i = i / 2;
    }
    a[j] = item;
}
void InsertMinHeap::show()
{
    cout << "Min heap using insert:\n";
    for (int i = 1; i <= n; i++)
    {
        cout << a[i];
        cout << "\t";
    }
}
int main()
{
    InsertMinHeap obj;
    obj.get();
    obj.show();
    return 0;
}
```

Output :

```
Enter how many element insert into heap:5
Enter heap element:5 300 40 2 10
Min heap using insert: 2 5 40 300 10
```

Practical 2:-

Write a program for creating Max Heap using ADJUST/HEAPIFY

```
#include<iostream>
#include<conio.h>
using namespace std;
class AdjustMaxHeap
{
private:
    int a[10], n, i;
```

```
public:
    void Adjust(int a[], int i, int n);
    void Heapify(int a[], int n);
    void get();
    void show();
};
void AdjustMaxHeap::get()
{
    cout << "Enter the size of array : ";
    cin >> n;
    cout << "Enter " << n << " element : ";
    for (int b = 1; b <= n; b++)
    {
        cin >> a[b];
    }
    Heapify(a, n);
}
void AdjustMaxHeap::Heapify(int a[], int n)
{
    for (i = (n / 2); i >= 1; i--)
    {
        Adjust(a, i, n);
    }

}
void AdjustMaxHeap::Adjust(int a[], int i, int n)
{
    int j, item;
    j = 2 * i;
    item = a[i];
    while (j <= n)
    {
        if (j < n && a[j] < a[j + 1])
        {
            j = j + 1;
        }
        if (item >= a[j])
        {
            return;
        }
        else
        {
            a[j / 2] = a[j];
            j = 2 * j;
        }

    }
    a[j / 2] = item;

}
void AdjustMaxHeap::show()
{
    cout << "element after using adjust heapify : ";
    for (int c = 1; c <= n; c++)
    {
        cout << a[c] << "\t";
    }
}
int main()
{
    AdjustMaxHeap obj;
    obj.get();
    obj.show();
    return(0);
}
```

//Output:

```
Enter the size of array : 7

Enter 7 element : 40 80 35 90 45 50 70

element after using adjust heapify : 90      80      70      40      45      50      35
```

```
//Write a program for creating Min Heap using ADJUST/HEAPIFY
#include<iostream>
#include<conio.h>
using namespace std;
class AdjustMinHeap
{
private:
    int a[10], n, i;
public:
    void Adjust(int a[], int i, int n);
    void Heapify(int a[], int n);
    void get();
    void show();
};
void AdjustMinHeap::get()
{
    cout << "Enter the number of nodes : ";
    cin >> n;
    cout << "Enter " << n << " nodes : ";
    for (int b = 1; b <= n; b++)
    {
        cin >> a[b];
    }
    Heapify(a, n);
}
void AdjustMinHeap::Heapify(int a[], int n)
{
    for (i = (n / 2); i >= 1; i--)
    {
        Adjust(a, i, n);
    }

}
void AdjustMinHeap::Adjust(int a[], int i, int n)
{
    int j, item;
    j = 2 * i;
    item = a[i];
    while (j <= n)
    {
        if (j < n && a[j] > a[j + 1])
            j++;
        if (item <= a[j])
            break;
        a[j / 2] = a[j];
        j = 2 * j;
    }
    a[j / 2] = item;

}
void AdjustMinHeap::show()
{
    cout << "element after using adjust heapify : ";
    for (int c = 1; c <= n; c++)
    {
        cout << a[c] << "\t";
    }
}
int main()
{
    AdjustMinHeap obj;
    obj.get();
    obj.show();
    return 0;
}
```

Output :

```
Enter 7 nodes : 40 80 35 90 45 50 70
element after using adjust heapify : 35     45      40      90      80      50      70
```

Practical 3:-

Write a program to implement Union & find operation

```
#include<iostream>
using namespace std;
class Union
{
    int p[20][20], n, i, m;
public:
    void union1(int i, int j)
    {
        int x;
        x = p[1][i] = p[1][j];
        if (p[1][i] > p[1][j])
        {
            p[1][j] = p[0][j];
            p[1][j] = x;
        }
        else
        {
            p[1][j] = p[0][i];
```

```
                p[1][i] = x;

        }
}
void display()
{
        int i = 0;
        cout << "\nenter the size of first tree:";
        cin >> m;
        cout << "Enter the element of first tree:";
        while (i <= m - 1)
        {
                cin >> p[0][i];
                i++;
        }
        cout << "enetr the parent:";
        p[1][0] = -m;
        i = 1;
        while (i <= m - 1)
        {
                cin >> p[1][i];
                i++;
        }
        cout << "\nenter the size of second tree:";
        cin >> n;
        cout << "Enter the element of second tree:";
        while (i < m + n)
        {
                cin >> p[0][i];
                i++;
            }
            cout << "enetr the parent:";
            p[1][m] = -n;
            i = m + 1;
            while (i < m + n)
            {
                cin >> p[1][i];
                i++;
            }

            cout << "\n union of two tree..\n";
            i = 0;
            while (i < m + n)
            {
                cout << "elemnt:" << p[0][i] << "\n";
                cout << "parent" << p[1][i] << "\n";
                i++;
            }
        }
};
int main()
{
        Union u;
        u.union1(0, 2);
        u.display();
        return 0;
}
```

OutPut:

```
enter the size of first tree:4

Enter the element of first tree:2

3

4

6

enetr the parent:4

3

2


enter the size of second tree:5

Enter the element of second tree:5

6

7

8

9

enetr the parent:7

6

8

9


 union of two tree..

elemnt:2

parent-4

elemnt:3

parent4

elemnt:4

parent3
```

```
elemnt:6
parent2
elemnt:5
parent-5
elemnt:6
parent7
elemnt:7
parent6
elemnt:8
parent8
elemnt:9
parent9
```

find operation:-

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
using namespace std;
class Find
{
public:
	int i;
	int a[50][50], m, z;
	void getdata();
	void find();
	void display();
};
void Find::getdata()
```

```
{
    int i = 0;
    cout << "Enter size of tree";
    cin >> m;
    cout << "\n Enter the element of tree";
    while (i < m)
    {
        cout << "\nElement of node:";
        cin >> a[0][i];
        cout << "\n Enter thee paren node:";
        cin >> a[1][i];
        i++;
    }
    cout << "\n Enter the element whose root you want to find";
    cin >> z;
}
void Find::find()
{
    int i, j, flag = 0;
    for (j = 0; j < m; j++)
    {
        if (a[0][j] == z)
        {
            flag = 1;
            break;
        }
    }
    if (flag == 0)
    {
```

```
                std::cout << "\n element is not present";
        }
        else
        {
                cout << "\nElement is present";
                cout << "\n root=" << a[1][j];
        }
}
void Find::display()
{
        int i = 0;
        while (i < m)
        {
                cout << "\n Elemtn of node:" << a[0][i];
                cout << "\n Parent:" << a[1][i];
                i++;

        }
}
int main()
{
        Find f;
        f.getdata();
        f.find();
        cout << "\nAfter the element of the tree";
        f.display();
        return 0;
}
```

Output:-

```
Enter size of tree3


 Enter the element of tree
Element of node:45


 Enter thee paren node:80


Element of node:50


 Enter thee paren node:80


Element of node:77


 Enter thee paren node:50


 Enter the element whose root you want to find77


Element is present
 root=50
After the element of the tree
 Elemtn of node:45
 Parent:80
 Elemtn of node:50
 Parent:80
 Elemtn of node:77
 Parent:50
```

Practical 4:-

```
//Write a program to find minimum and maximum from a given array.
#include<iostream>
#include<cmath>
using namespace std;
class MaxMinDemo
{
    int A[16];
    int n;
    int fmax, fmin;
public:
    void getData()
    {
        cout << "enter the number of elements:";
        cin >> n;
        cout << "Enter the elements:";
        for (int i = 0; i < n; i++)
        {
            cin >> A[i];
        }
    }
    void MaxMin()
    {
        HMaxMin(0, n - 1, fmax, fmin, A);
    }
    void Display()
    {
        cout << "Given array is:";
        for (int i = 0; i <= n; i++)
        {
            cout << A[i] << " ";
        }
        cout << endl;
        cout << "\n Max:" << fmax << "\n Min:" << fmin;
    }
    void HMaxMin(int i, int j, int& fmax, int& fmin, int A[]);
};
void MaxMinDemo::HMaxMin(int i, int j, int& fmax, int& fmin, int A[])
{
    if (i == j)
    {
        fmax = fmin = A[i];
    }
    else if (i == j - 1)
    {
        if (A[i] > A[j])
        {
            fmax = A[i];
            fmin = A[j];
        }
        else
        {
            fmax = A[j];
            fmin = A[i];
        }
    }
    else
    {
        int mid = (i + j) / 2;
        int gmax, gmin;
        HMaxMin(i, mid, gmax, gmin, A);
        int hmax, hmin;
        HMaxMin(mid + 1, j, hmax, hmin, A);
        fmax = max(gmax, hmax);
        fmin = min(gmin, hmin);
    }
}
```

```
int main()
{
    MaxMinDemo o;
    o.getData();
    o.MaxMin();
    o.Display();
    return 0;
}
```

Output:

```
enter the number of elements:5
Enter the elements:34
67
8
33
29
Given array is:34 67 8 33 29 6422280

 Max:67
 Min:8
```

Practical 5:-

Write a program for searching element from given array using binary search.

```
#include<iostream>
using namespace std;
class BinarySearch
{
private:
    int a[20], n, r;
public:
    int binary(int a[], int n, int val);
    void get();
    void show(int r);
};
int BinarySearch::binary(int a[], int n, int val)
{
    int first = 0;
    int last = n - 1;
    int mid;
    while (first <= last)
    {
        mid = (first + last) / 2;
        if (a[mid] == val)
            return mid + 1;
        else if (val > a[mid])
            first = mid + 1;
        else
            last = mid - 1;
    }
    return 0;

}
void BinarySearch::show(int r)
{
    if (r == 0)
    {
        cout << "element not found.";
    }
    else
    {
        cout << "element found at position : " << r;
    }
}
void BinarySearch::get()
```

```
{
    int no;
    cout << "Enter no. of elements :";
    cin >> n;
    cout << "Enter only sorted element :  ";
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    cout << "Enter element to search : ";
    cin >> no;
    int result = binary(a, n, no);
    show(result);
}
int main()
{
    BinarySearch obj;
    obj.get();
    return 0;
}
```

Output:

```
Enter no. of elements :5
Enter only sorted element :    12
23
36
65
78
Enter element to search : 36
element found at position : 3
```

Practical 6:-

```
//Write a program for sorting from given array in ascending / descending order
```

```
// n = 1000, 2000, 300 find the exact time of execution.
```

HeapSort

```
#include<iostream>
#include<conio.h>
#include<chrono>
using namespace std;
using namespace std::chrono;
class HeapSort
{
public:
    int done, maxchild, temp;
    int A[1000];
    int i, n;
    void shiftdown(int[], int, int);
    void heapsort(int[], int);
    void getdata();
    void display();
};
void HeapSort::getdata()
{
    cout << "Enter size of array:";
    cin >> n;
    cout << "Enter the array elements=";
    for (int i = 0; i < n; i++)
    {
        cin >> A[i];
    }
}
void HeapSort::shiftdown(int A[], int root, int bottom)
{
    done = 0;
    while ((root * 2 + 1 <= bottom) && (!done))
    {
        if (root * 2 + 1 == bottom || A[root * 2 + 1] > A[root * 2 + 2])
        {
            maxchild = root * 2 + 1;
        }
        else
        {
            maxchild = root * 2 + 2;
        }
        if (A[root] < A[maxchild])
        {
            temp = A[root];
            A[root] = A[maxchild];
            A[maxchild] = temp;
            root = maxchild;
        }
        else
        {
            done = 1;
        }
    }
}
void HeapSort::heapsort(int A[], int ub)
{
    for (int i = (ub / 2.0) - 1; i >= 0; i--)
    {
        shiftdown(A, i, ub);
    }
    for (int i = ub; i >= 1; i--)
    {
        temp = A[0];
        A[0] = A[i];
        A[i] = temp;
        shiftdown(A, 0, i - 1);
```

```cpp
    }
}
void HeapSort::display()
{
    cout << "Elements you entered:";
    for (int i = 0; i < n; i++)
    {
        cout << A[i] << " ";

    }
    heapsort(A, n - 1);
    cout << "\nSorted element  in ascending order:";
    for (int i = 0; i < n; i++)
    {
        cout << A[i] << " ";
    }
    cout << endl;
    cout << "\nSorted element  in descending order:";
    for (int i = n; i >= 0; i--)
    {
        cout << A[i] << " ";
    }
    cout << endl;
}
int main()
{
    HeapSort h;
    h.getdata();
    auto start = high_resolution_clock::now();
    h.display();
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<seconds>(stop - start);
    cout << "\n Exact time of execution:" << duration.count() << "seconds\n" << endl;
```

```
}
```

Output:

```
Enter size of array:5

Enter the array elements=34

54

23

78

99

Elements you entered:34 54 23 78 99

Sorted element    in ascending order:23 34 54 78 99


Sorted element    in descending order: 99 78 54 34 23


  Exact time of execution:0seconds
```

Write a program for sorting given array in ascending/descending order using Merge sort.

```
#include<iostream>
using namespace std;
class MergeSortDemo
{
    int A[16];
    int n;
public:
    void getData()
    {
        cout << "Enter the number of elements:";
        cin >> n;
        cout << "Enter the element:";
        for (int i = 0; i < n; i++)
```

```cpp
        {
            cin >> A[i];
        }
    }

    void display()
    {
        cout << "Sorted elements in ascending order :";
        for (int i = 0; i < n; i++)
        {
            cout << A[i] << "\t";
        }
        cout << endl;

        cout << "Sorted elements in descending order :";
        for (int i = n; i >= 0; i--)
        {
            cout << A[i] << "\t";
        }
        cout << endl;
    }
    void merge(int A[], int temp[], int left, int mid, int right);
    void m_sort(int A[], int temp[], int left, int right);
    void mergeSort();
};
void MergeSortDemo::merge(int A[], int temp[], int left, int mid, int right)
{
    int t_pos, left_end, n, i;
    t_pos = left;
    left_end = mid - 1;
    n = right - left + 1;
    while (left <= left_end && mid <= right)
    {
        if (A[left] < A[mid])
        {
            temp[t_pos] = A[left];
            t_pos = t_pos + 1;
            left = left + 1;
        }
        else
        {
            temp[t_pos] = A[mid];
            t_pos = t_pos + 1;
            mid = mid + 1;
        }
    }
    while (left <= left_end)
    {
        temp[t_pos] = A[left];
        t_pos = t_pos + 1;
        left = left + 1;
    }
    while (mid <= right)
    {
        temp[t_pos] = A[mid];
        t_pos = t_pos + 1;
        mid = mid + 1;
    }
    for (i = 0; i < n; i++)
    {
        A[right] = temp[right];
        right--;
    }
}
void MergeSortDemo::m_sort(int A[], int temp[], int left, int right)
{
    int mid;
    if (right > left)
    {
        mid = (left + right) / 2;
        m_sort(A, temp, left, mid);
        m_sort(A, temp, mid + 1, right);
        merge(A, temp, left, mid + 1, right);
    }
}
void MergeSortDemo::mergeSort()
{
    int temp[10];
    m_sort(A, temp, 0, n - 1);
}
int main(int argc, char* argv[])
{
    MergeSortDemo o;
    o.getData();
    o.mergeSort();
    o.display();
}
```

```
Outuput:
Enter the number of elements : 5
Enter the element : 23 45 76 99 45
Sorted elements in ascending order : 23  45       45       76       99
Sorted elements in descending order : 99       76       45       45       23
```

```
//Write a program for sorting given array in ascending/descending order using Quick sort.
#include<iostream>
using namespace std;
class QuickSortDemo
{
    int A[16];
    int n;
public:
    void getData()
    {
        cout << "Enter the number of elements:";
        cin >> n;
        cout << "Enter the element:";
        for (int i = 0; i < n; i++)
        {
            cin >> A[i];
        }
    }
    void QuickSort()
    {
        QuickSort(A, 0, n - 1);
    }

    void display()
    {
        cout << "Sorted elements in ascending order :";
        for (int i = 0; i < n; i++)
        {
            cout << A[i] << "\t";
        }
        cout << endl;

        cout << "Sorted elements in descending order :";
        for (int i = n; i >= 0; i--)
        {
            cout << A[i] << "\t";
        }
        cout << endl;
    }
    int partition(int A[], int lb, int ub);
    void QuickSort(int A[], int lb, int ub);

};
int QuickSortDemo::partition(int A[], int lb, int ub)
{
    int temp;
    int start = lb, end = ub;
    int pivot = A[lb];
    while (start < end)
    {
        while (A[start] <= pivot)start++;
        while (A[end] > pivot)end--;
        if (start < end)
        {
            temp = A[start];
            A[start] = A[end];
            A[end] = temp;
        }
    }
    temp = A[lb];
    A[lb] = A[end];
    A[end] = temp;
    return end;
}
void QuickSortDemo::QuickSort(int A[], int lb, int ub)
{
    int loc;
    if (lb < ub)
    {
        loc = partition(A, lb, ub);
        QuickSort(A, lb, loc - 1);
        QuickSort(A, loc + 1, ub);
    }
}

int main(int argc, char* argv[])
{
    QuickSortDemo o;
    o.getData();
    o.QuickSort();
    o.display();
}
```

```
Output:
Enter the number of elements : 5
Enter the element : 23
45
65
77
34
Sorted elements in ascending order : 23  34      45      65      77
Sorted elements in descending order : 77      65      45      34      23
```

Practical 7:-

: Write a program for matrix multiplication using Strassen's matrix multiplication

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include<process.h>
using namespace std;
class matrix
{
	int i, j, a[10][10], b[10][10], c[10][10];
public:
    void menu();
    void read_matrix();
    void matrix_mul();
    void print();
};
void matrix::menu()
{
    int ch;
    cout << "\n 1:read two matrix";
    cout << "\n 2:matrix multiplication";
    cout << "\n 3:print result";
    cout << "\n 4:exit";
    do
    {
        cout << "\n enter choice: ";
        cin >> ch;
        switch (ch)
        {
        case 1:
            read_matrix();
            break;
        case 2:
            matrix_mul();
            cout << "Multiplication Is Successfully...";
            break;
        case 3:
            print();
            break;
        default:
            ch = 4;
        }
    } while (ch != 3);
}
```

```
void matrix::read_matrix()
{
    cout << "\n enter first matrix: \n";
    for (i = 1; i <= 2; i++)
    {
        for (j = 1; j <= 2; j++)
        {
            cin >> a[i][j];
        }
        cout << "\n";
    }
    cout << "\n enter second matrix: \n";
    for (i = 1; i <= 2; i++)
    {
        for (j = 1; j <= 2; j++)
        {
            cin >> b[i][j];
        }
        cout << "\n";
    }
}
void matrix::matrix_mul()
{
    int P = ((a[1][1] + a[2][2]) * (b[1][1] + b[2][2]));
    int Q = (a[2][1] + a[2][2]) * b[1][1];
    int R = a[1][1] * (b[1][2] - b[2][2]);
    int S = a[2][2] * (b[2][1] - b[1][1]);
    int T = (a[1][1] + a[1][2]) * b[2][2];
    int U = (a[2][1] - a[1][1]) * (b[1][1] + b[1][2]);
    int V = (a[1][2] - a[2][2]) * (b[2][1] + b[2][2]);
    c[1][1] = P + S - T + V;
    c[1][2] = R + T;
    c[2][1] = Q + S;
```

```cpp
	c[2][2] = P + R - Q + U;
}
void matrix::print()
{
	cout << "\n multiplication of two 2*2 matrix are: \n";
	for (i = 1; i <= 2; i++)
	{
		for (j = 1; j <= 2; j++)
		{
			cout << c[i][j] << "\t";

		}
		cout << "\n";
	}
}
int main()
{
	matrix m1;
	m1.menu();
	return 0;
}
```

OUTPUT:

```
1 : read two matrix
2 : matrix multiplication
3 : print result
4 : exit
enter choice : 1
enter first matrix :
1 1
1 1
enter second matrix :
1 1
1 1
```

```
enter choice : 2
Multiplication Is Successfully...
enter choice : 3
multiplication of two 2 * 2 matrix are :
2       2
2       2
```

Practical 8:-

Write a programme to find solution of knapsack instant

```
#include <iostream>
using namespace std;
class Knapsack
{
    float weight[20], profit[20], capacity;
    int num;
    float ratio[20], temp;
public:
    void getData()
    {
        int i;
        cout << "Enter the no. of objects : ";
        cin >> num;
        cout << "Enter the weight & profit of each objects : ";
        for (i = 0; i < num; i++)
        {
            cin >> weight[i];
            cin >> profit[i];
        }
        cout << "Enter the capacity of each kanpsack : ";
        cin >> capacity;
        for (i = 0; i < num; i++)
        {
            cout << weight[i];
        }
        for (i = 0; i < num; i++)
        {
            ratio[i] = profit[i] / weight[i];
        }
    }
    void knapsack()
    {
        sortData();
        hknapsack(num, weight, profit, capacity);
    }
    void sortData();
    void hknapsack(int n, float weight[], float profit[], float capacity);
};
void Knapsack::sortData()
{
    int i, j;
    for (i = 0; i < num; i++)
    {
        for (j = i + 1; j < num; j++)
        {
            if (ratio[i] < ratio[j])
            {
                temp = ratio[j];
                ratio[j] = ratio[i];
                ratio[i] = temp;
                temp = weight[j];
                weight[j] = weight[i];
                weight[i] = temp;
                temp = profit[j];
                profit[j] = profit[i];
                profit[i] = temp;
            }
        }
    }
}
void Knapsack::hknapsack(int n, float weight[], float profit[], float capacity)
{
    float x[20], tp = 0;
    int i, u;
    u = capacity;
    for (i = 0; i < n; i++)
        x[i] = 0.0;
    for (i = 0; i < n; i++)
    {
        if (weight[i] > u)
            break;
        else
        {
            x[i] = 1.0;
            tp = tp + profit[i];
            u = u - weight[i];
        }
    }
    if (i < n)
        x[i] = u / weight[i];
    tp = tp + (x[i] * profit[i]);
    cout << "\n the result vector is : ";
    for (i = 0; i < n; i++)
        cout << " " << x[i];
    cout << "\n Maximum profit is : " << tp;
}
int main()
{
    Knapsack ksd;
    ksd.getData();
    ksd.knapsack();
};
```

Output:

```
Enter the no.of objects : 4
Enter the weight & profit of each objects : 12 32
11 22
45 44
12 43
Enter the capacity of each kanpsack : 20
the result vector is : 1 0.666667 0 0
Maximum profit is : 64.3333
```

Practical 9:-

write a program to find the shortest path using single source pair shortest path

```
#include<iostream>
#include<conio.h>
using namespace std;
class shortest
{
private:
    int n, cost[20][20];
public:
    void getdata();
    void shortestpath(int v);

};
void shortest::getdata()
{
    cout << "Enter the number of the vertices:\n";
    cin >> n;
    cout << "\nEnter the Adjacent Matrix=\n";
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            cin >> cost[i][j];
        }
    }
}
void shortest::shortestpath(int v)
{
    int s[50], dist[50], i, j, d1, d2, u;
    for (i = 1; i <= n; i++)
    {
        s[i] = 0;
        dist[i] = cost[v][i];
    }
    s[v] = 1;
    dist[v] = 0;
    for (int num = 2; num <= n - 1; num++)
    {
        int min = 999;
        for (int i = 1; i <= n; i++)
        {
            if (dist[i] < min && s[i == 0])
            {
                u = i;
                min = dist[i];
            }

        }
        s[u] = 1;
        for (int j = 1; j <= n; j++)
        {
            if (s[j] = 0)
            {
                d1 = dist[j];
                d2 = dist[u] + cost[u][j];
                dist[j] = d1 < d2 ? d1 : d2;

            }
        }
    }
    for (int i = 1; i <= n; i++)
    {
        cout << "\n Distance of vertex " << v << " from vertex " << i << " is " << dist[i];
    }

}
int main()
{
    int v, i;
    shortest s;
    s.getdata();
    cout << " Enter the starting vertex : \n";
    cin >> v;
    s.shortestpath(v);
    return 0;
}
```

Output:

```
Enter the number of the vertices :
3

Enter the Adjacent Matrix =
2 3 4
5 6 7
```

```
8 9 1
Enter the starting vertex :
1

Distance of vertex 1 from vertex 1 is 0
Distance of vertex 1 from vertex 2 is 3
Distance of vertex 1 from vertex 3 is 4
```

Practical 10:-

```
//Write a program to find Minimum-Cost Spanning Trees (Prim’s & Kruskal’s algorithm).
//Prim’s Algorithm
#include<iostream>
#include<algorithm>
#include<conio.h>
using namespace std;
class Prims
{
public:
    void Cost();
};
void Prims::Cost()
{
    int a, b, u, n, v, i, j, ne = 1;
    int visited[10] = { 0 }, m, mincost = 0, cost[10][10];
    cout << "Enter the number of nodes:";
    cin >> n;
    cout << "Enter adjacency matrix:\n";
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
        {
            cin >> cost[i][j];
            if (cost[i][j] == 0)
                cost[i][j] = 999;
        }
    visited[1] = 1;
    cout << "\n";
    while (ne < n)
    {
        for (i = 1, m = 999; i <= n; i++)
            for (j = 1; j <= n; j++)
                if (cost[i][j] < m)
                    if (visited[i] == 0)
                        continue;
                    else
                    {
                        m = cost[i][j];
                        a = u = i;
                        b = v = j;
                    }
        if (visited[u] == 0 || visited[v] == 0)
        {
            cout << "\n" << ne++ << "edge";
            cout << "(" << a << "," << b << "" << ")" << "=";
            cout << m;
            mincost = mincost += m;
            visited[b] = 1;
        }
        cost[a][b] = cost[b][a] = 999;
    }
    cout << "\n minimum cost=" << mincost;
}
int main()
{
    Prims p;
    p.Cost();
    return 0;
```

```
}
```

Output:

```
Enter the number of nodes : 4
Enter adjacency matrix :
0 4 3 0
0 0 9 6
0 0 0 8
0 0 0 0


1edge(1, 3) = 3
2edge(1, 2) = 4
3edge(2, 4) = 6
minimum cost = 13
```

Krushkal

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int n;
class sprime
{
	int v, cost[10][10], i, j, s[10], e[10], near1[10], t[10][3], m, minedge, k, l, mincost;
	int jindex;
	float dist[10];
public:
	void get();
```

```
    void prime();
    void display();
};
void sprime::get()
{
    m = 1;
    minedge = 9999;
    cout << "\n enter adjancy vertices\n";
    cin >> n;
    cout << "\n enter adjancy matrix\n";
    for (i = 1;i <= n;i++)
        for (j = 1;j <= 1;j++)
            cin >> cost[i][j];
    if (cost[i][j] == -1)
    {
        cost[i][j] = 9999;
    }
    else
    {
        e[m] = cost[i][j];
        if (e[m] < minedge)
        {
            minedge = e[i];k = i;l = j;
        }
    }
}
void sprime::prime()
{
    t[1][1] = k;
```

```
t[1][2] = l;
mincost = cost[k][l];
for (i = 1;i <= n;i++)
{
    if (cost[i][1] < cost[i][k])
        near1[i] = l;
    else
        near1[i] = k;
}
near1[k] = near1[l] = 0;
int minj = 9999;
for (i = 2;i <= n;i++)
{
    minj = 9999;
    for (j = 1;j <= 1;j++)
    {
        if (near1[j] != 0)
        {
            if (cost[j][near1[j] < minj])
            {
                minj = cost[j][near1[j]];
                jindex = j;
            }
        }
    }
    t[i][1] = jindex;
    t[i][2] = near1[jindex];
    mincost = mincost + cost[jindex][near1[jindex]];
    near1[jindex] = 0;
```

```
        for (int k1 = 1;k1 <= n;k1++)
        {
            if (near1[k1] != 0 && cost[k1][near1[k1]] > cost[k1][jindex])
                near1[k1] = jindex;
        }
    }
    cout << "mincost" << mincost;
}
void sprime::display()
{
    cout << "\n minimum spanning tree path as follow\n";
    cout << t[1][1] << "__>" << t[1][2];
    for (i = 2;i <= n;i++)
    {
        cout << "__>";
        cout << t[i][j];
    }
}
int main()
{
    sprime d;
    d.get();
    d.prime();
    d.display();
    return 0;
}
    output
        Enter the no.of vertices 7
        Enter the adjacency matrix
```

```
0  28 − 1 − 1 − 1  10 − 1

28 − 1  16 − 1 − 1 − 1  14

− 1 16   0  12 − 1 − 1 − 1

− 1 − 1  12   0  22 − 1  18

− 1 − 1 − 1  22   0  25  24

10 − 1 − 1 − 1  25   0 − 1

− 1 14 − 1  18  25 − 1   0

Mincost 99

Minimum spanning tree path as follow

1-- > 1-- > 6-- > 5-- > 4-- > 3-- > --2 -- > 7
```

Practical 11:-

```
//write a program to find the shortest path using all pair path

#include<iostream>

#include<conio.h>

#include<stdio.h>

using namespace std;

class shortest

{

private:

    int i, j, n, cost[20][20], a[50][50];

public:

    void getdata();

    void allpair();

    void putdata();


};

void shortest::getdata()

{

    cout << "Enter the number of the vertices:\n";
    cin >> n;
    cout << "\nEnter the cost:";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> cost[i][j];
        }
    }
}
void shortest::allpair()
{
    int k, b;
    for (int i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            a[i][j] = cost[i][j];
        }
    }
    for (k = 0; k < n; k++)
    {
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                b = a[i][k] + a[k][j];
                a[i][j] = (a[i][j] < b) ? a[i][j] : b;
            }
        }
    }
}
void shortest::putdata()
{
    cout << "\n Shortest path distance:";
    cout << "\n";
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            cout << a[i][j] << "\t";

        }
        cout << "\n";
    }
}
int main()
{

    shortest obj;
    obj.getdata();
    obj.allpair();
    obj.putdata();
}
```

Output:

```
Enter the number of the vertices :
3
```

```
Enter the cost : 0 4 11 6 0 2  3 9 0

Shortest path distance :
0       4       6
5       0       2
3       7       0
```

Practical 12:-

```cpp
//Write a program to find Longest Common Subsequence.
#include<iostream>
#include<conio.h>
#include<string.h>
#include<stdio.h>
using namespace std;
class Longestcommon
{
public:
    char* a1, * b1, a[10], b[10];
    int c[10][10];
    void getdata();
    void LCS();
    void putdata(int, int);
    void sequence(int, int);
};
void Longestcommon::getdata()
{
    cout << "Enter first string:";
    cin >> a;
    cout << "Enter second string:";
    cin >> b;

};
void Longestcommon::LCS()
{
    int i, j, m, n;
    m = strlen(a);
    n = strlen(b);
    for (i = 0; i <= m; i++)
        c[i][0] = 0;
    for (j = 1; i <= n; j++)
        c[0][j] = 0;
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
            if (a[i - 1] == b[j - 1])
                c[i][j] = 1 + (c[i - 1][j - 1]);
            else
                c[i][j] = (c[i - 1][i] > c[i][j - 1] ? c[i - 1][j] : c[i][j - 1]);
    putdata(m, n);
    cout << "\n";
    sequence(m, n);
}

void Longestcommon::putdata(int m, int n)
{
    int i, j;
    cout << "the resultant matrix....\n";
    cout << "\t";
    for (i = 0; i <= m; i++)
    {
        cout << "\t" << a[i];
    }
    cout << "\n";
    for (i = 0; i <= m; i++)
    {
        if (i > 0)
            cout << "\n" << b[i - 1];
        for (j = 0; j <= n; j++);
        {
            cout << "\t" << c[i][j];
        }
        cout << "\n";
    }
    cout << "Longest common subsequence....\n";
}
void Longestcommon::sequence(int m, int n)
{
    if (c[m][n] == 0)
        return;
    if (c[m][n] == c[m - 1][n])
        sequence(m - 1, n);
    else  if (c[m][n] == c[m][n - 1])
        sequence(m, n - 1);
    else
    {
        sequence(m - 1, n - 1);
        cout << "\t" << a[m - 1];
    }
}
int main()
{
    Longestcommon l;
    l.getdata();
    l.LCS();
    return 0;
}
```

```
Output:
Enter first string : ABAB
Enter second string : AABA
the resultant matrix....
A       B       A       B
- 1

A       1877944186

A       2

B       1975679874

A       0
Longest common subsequence....
A       B       A
```

Practical 13:-

```
//Write a program to implement breadth first search
```

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int a[10][10], i, j, v[10], n, q[15], f, b, r;
class bfsl
{
public:
    void getdata();
    int bfs(int vl);
    void display();
};
void bfsl::getdata()
{
    std::cout << "Enter the no. ofn vertices:\t";
    cin >> n;
    std::cout << "Enter the matrix:=";
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            cin >> a[i][j];
}
int bfsl::bfs(int vl)
{
    int w;
    v[vl] = 1;
    std::cout << vl;
    f = r = 0;
    while (1)
    {
        for (w = 1; w <= n; w++)

        {

            if (a[vl][w] == 1)

            {

                if (v[w] == 0)

                {

                    if ((f == 0) && (r == 0))

                        f = r = 1;

                    else

                        r++;

                    q[r] = w;

                    v[w] = 1;

                    std::cout << "\t" << w;


                }

            }

        }

        if ((f == 0) && (r == 0))

            return 0;

        vl = q[f];

        if (f == r)

            f = r = 0;

        else

            f++;

    }

}

void bfsl::display()

{

    std::cout << "Sequenced of node in bfs is:=";
    bfs(1);
    cout << "\n";
    for (i = 1; i <= n; i++)
        v[i] = 0;
}
int main()
{
    bfsl b;
    b.getdata();
    b.display();
    return 0;
}
```

Output:

```
Enter the no.ofn vertices : 5
Enter the matrix : = 0 1 1 0 0
0 0 0 1 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
Sequenced of node in bfs is : = 1  2       3       4       5
```

```
//Write a program to implement depth first search.
#include<iostream>
#include<conio.h>
#include<stdio.h>

using namespace std;
```

```
int a[10][10], j, i, v[10], n, q[15], f, b, r;

class dfs1
{
public:
    void getdata();
    int dfs(int i);
    void display();
};
void dfs1::getdata()
{
    cout << "Enter the vertices : ";
    cin >> n;
    cout << "Enter the matrix : ";
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            cin >> a[i][j];
}
int dfs1::dfs(int i)
{
    int w;
    v[i] = 1;
    cout << i;
    for (w = 1; w <= n; w++)
    {
        if (a[i][w] == 1)
        {
            if (v[w] == 0)
                dfs(w);
        }
    }
    return(0);
}
void dfs1::display()
{
    cout << "\n Sequence of node in dfs are : ";
    dfs(1);
    cout << "\n";
    for (i = 1; i <= n; i++)
        v[i] = 0;
}
int main()
{
    dfs1 b;
    b.getdata();
    b.display();
    return 0;
}
```

Output:

```
Enter the vertices : 5
Enter the matrix : 0 1 1 0 0
0 0 0 1 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
Sequence of node in dfs are : 12453
```

Practical 14:-

```
//Write a program to implement breadth first traversal.
#include<stdio.h>
#include<conio.h>
#include<iostream>
using namespace std;
class BFT
{
private:
    int matrix[50][50], n;
    int visited[50];
    int q[50], front, rear;
public:
    void getdata();
    void bft(int v);

};
void BFT::getdata()
{
    int i, j;
    front = rear = 1;
    cout << "\n Enter the number of the nodes";
    cin >> n;
    cout << "\n Enter the matrix";
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            cin >> matrix[i][j];
        }
    }
    for (i = 1; i <= n; i++)
    {
        visited[i] = 0;

    }
}
void BFT::bft(int v)
{
    int i, t;
    for (i = 1; i <= n; i++)
    {
        visited[i] = 0;
    }
    int u = v;
    t = '\0';
    visited[v] = 1;
    cout << v << "\t";
    do
    {
        for (int w = 1; w <= n; w++)
        {
            if (matrix[u][w] == 1)
            {
                if (visited[w] == 0)
                {
                    q[rear++] = w;
                    visited[w] = 1;
                    t = t + (u, w);
                    cout << w << "\t";

                }
            }
        }
        if (front == rear)
        {
            break;
        }
        u = q[front++];
    } while (1);
}
int main()
{
    int v;
    BFT b;
    b.getdata();
    cout << "\n Enter the starting node:";
    cin >> v;
    cout << "\n Visited node using BFT:";
    b.bft(v);
    return 0;
}
```

Output:

```
Enter the number of the nodes4


Enter the matrix1 0 1 0
0 0 1 1
```

```
0 1 0 1
1 1 0 0
```

```cpp
//Write a program to implement depth first traversal.

#include<iostream>
#include<conio.h>
#include<process.h>

using namespace std;
int Visited[20], v, a[20][20], n, i;
class DFS
{
public:
    void getdata();
    void dfs(int);
    void dft();
}d;
void DFS::getdata()
{
    int i, j;
    cout << "\n Enter the vertices : ";
    cin >> n;
    cout << "\n Enter the Adjacency matrix";
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            cin >> a[i][j];
        }
    }
}
void DFS::dfs(int v)
{
    Visited[v] = 1;
    cout << v << "\t";
    for (int w = 1; w <= n; w++)
    {
        if (a[v][w] == 1)
            if (Visited[w] == 0)
                dfs(w);
    }
}
void DFS::dft()
{
    for (int i = 1; i <= n; i++)
        Visited[i] = 0;
    for (int i = 1; i < n; i++)
        if (Visited[i] == 0)
            dfs(i);
}
int main()
{
    d.getdata();
    cout << "\n DFS order of nodes is  :  ";
    d.dft();
    return 0;
}
```

Output:

```
Enter the vertices : 8

Enter the Adjacency matrix0 1 1 0 0 0 0 0
1 0 0 1 1 0 0 0
0 0 0 0 0 1 1 0
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 0 1 1 1 1 0

DFS order of nodes is : 1    2       4       8       5       6       3       7
```

Practical 15:-

```cpp
// Write a program to find all solutions for 8-queen problem using backtracking.
#include <iostream>
#include <vector>
using namespace std;

bool isSafe(vector<vector<int>>& board, int row, int col, int N) {
    for (int i = 0; i < col; i++)
        if (board[row][i])
            return false;

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
```

```
            return false;

    for (int i = row, j = col; i < N && j >= 0; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

bool solveNQueens(vector<vector<int>>& board, int col, int N,
vector<vector<vector<int>>>& solutions) {
    if (col == N) {
        solutions.push_back(board);
        return true;
    }

    bool res = false;
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col, N)) {
            board[i][col] = 1;
            res = solveNQueens(board, col + 1, N, solutions) || res;
            board[i][col] = 0;
        }
    }

    return res;
}

void printSolution(vector<vector<int>>& board) {
```

```cpp
    int N = board.size();
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

int main() {
    int N = 8;

    vector<vector<int>> board(N, vector<int>(N, 0));
    vector<vector<vector<int>>> solutions;

    solveNQueens(board, 0, N, solutions);

    int numSolutions = solutions.size();
    cout << "Total solutions: " << numSolutions << endl;

    for (int i = 0; i < numSolutions; i++) {
        cout << "Solution " << i + 1 << ":\n";
        printSolution(solutions[i]);
    }

    return 0;
}
```