ADA LAB FILE

J.C. BOSE UNIVERSITY OF SCIENCE AND TECHNOLOGY, YMCA

Submitted To: **Dr. Lalit Sir**

Submitted By: **Mukesh Aggarwal**

Class: **MCA 2nd Semester**

Roll No.:**24001602043**

INDEX

PROGRAM:

1. Write a program to implement Selection Sort.

```
#include <iostream>
#include <cstdlib>
#include <chrono>

using namespace std;
void selection(int a[], int n) {
    int i, j, p;
    for (j = 0; j < n - 1; j++) {
        p = j;
        for (i = j + 1; i < n; i++) {
            if (a[i] < a[p]) {
                p = i;
            }
        }
        swap(a[j], a[p]);
    }
    cout << "Sorted array is: ";
    for (int i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
}
int main() {
    const int n = 100;
    int a[n];
    srand(time(0)); // Seed the random number generator
```

```
    for (int i = 0; i < n; i++) {
        a[i] = rand() % n + 1; // random values between 1 and n
    }
    auto start = chrono::high_resolution_clock::now();
    selection(a, n);
    auto end = chrono::high_resolution_clock::now();

    chrono::duration<double, milli> elapsed = end - start;
    cout << "Time taken to sort the array: " << elapsed.count() << " ms" << endl;

    return 0;
}
```

OUTPUT: -

```
[Running] cd "c:\Users\mukes\OneDrive\Desktop\New folder\" && g++ tempCodeRunnerFile.cpp -o
tempCodeRunnerFile && "c:\Users\mukes\OneDrive\Desktop\New folder\"tempCodeRunnerFile
Sorted array is: 3 4 4 6 7 7 7 10 11 12 13 13 15 16 18 21 21 22 23 23 25 26 27 28 29 29 30 31 31 31 32
33 34 34 34 34 37 38 41 41 42 42 43 43 43 45 47 48 49 51 51 52 52 53 54 56 58 58 61 63 64 65 66 68 68
70 70 70 72 72 73 75 77 81 81 82 82 84 84 85 85 86 86 86 87 87 88 89 90 90 91 92 92 93 93 94 96 98 98
98
Time taken to sort the array: 1.009 ms

[Done] exited with code=0 in 1.32 seconds
```

2. Write a program to implement Insertion Sort.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <chrono>

using namespace std;

void insertion(int a[], int n)
{
    // Fill array with random values between 1 and 100
    for (int i = 0; i < n; i++)
    {
        a[i] = rand() % 100 + 1;
    }
    // Insertion Sort
    for (int j = 1; j < n; j++)
    {
        int key = a[j];
        int i = j - 1;
        while (i >= 0 && a[i] > key)
        {
            a[i + 1] = a[i];
            i--;
        }
        a[i + 1] = key;
    }
    // Print sorted array
    cout << "Sorted array is:\n";
```

```
    for (int i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
}
int main()
{
    const int n = 1000;
    int a[n];

    srand(time(0));  // Seed random number generator

    auto start = chrono::high_resolution_clock::now();
    insertion(a, n);
    auto end = chrono::high_resolution_clock::now();

    chrono::duration<double, milli> elapsed = end - start;
    cout << "\nTime taken to sort the array: " << elapsed.count() << " ms" << endl;

    return 0;
}
```

OUTPUT: -

```
[Running] cd "c:\Users\mukes\OneDrive\Desktop\New folder\" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "c:\Users\mukes\OneDrive\Desktop\New
folder\"tempCodeRunnerFile
Sorted array is:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 7 7 7 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 10 10 10 10 10 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 13 13 14 14
14 14 14 14 14 15 15 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 19 19 19
19 19 20 20 20 20 20 20 20 21 21 21 21 21 21 22 22 22 22 22 22 22 22 23 23 23 23 23 23 23 24 24 24 24 24 24 24 24 24 24 24 24 24 25 25 25 25 25 25 25 25
25 25 25 26 26 26 26 26 26 26 26 26 27 27 27 27 27 27 27 27 27 28 28 28 28 28 28 28 28 29 29 29 29 29 29 29 29 29 29 29 29 29 30 30 30 30 30 30 30 30 31
31 31 31 31 31 31 31 31 32 32 32 32 32 32 32 32 32 32 32 32 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 34 34 34 34 34 34 34 34 34 34 34 35 35 35 35 35 35
35 35 35 35 35 36 36 36 36 36 36 36 36 36 36 37 37 37 37 37 37 37 38 38 38 38 38 38 38 38 38 38 38 38 38 38 39 39 39 39 39 39 39 40 40 40 40 41 41 41 41 41 41
41 41 41 41 41 41 41 42 42 42 42 42 42 42 42 43 43 43 43 43 43 43 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 45 45 45 45 45 45 45 45 45 45 46 46 46 46 46 46
46 47 47 47 47 47 47 47 47 47 47 47 48 48 48 48 48 48 48 48 49 49 49 49 49 49 49 49 50 50 50 50 50 50 50 50 50 51 51 51 51 51 51 51 51 52 52 52 52 52 52 52 52
52 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 54 54 54 54 54 54 54 54 54 54 54 55 55 55 55 55 55 55 55 55 55 55 55 55 55 56 56 56 56 56 56 56 56 56 56 56 56
57 57 57 57 57 57 57 57 57 57 57 57 58 58 58 58 58 58 58 58 59 59 59 59 59 59 59 59 59 59 59 59 59 60 60 60 60 60 60 60 60 60 60 60 60 61 61 61 61 61 61 61 61
61 61 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 63 63 63 63 63 63 64 64 64 64 64 64 65 65 65 65 65 65 65 65 65 66 66 66 66 66 66 66 66 66 66 66
66 66 66 66 67 67 67 67 67 67 67 67 68 68 68 68 68 68 68 68 68 68 68 69 69 69 69 69 69 70 70 70 70 70 70 70 70 70 71 71 71 71 71 71 71 71 71 72 72 72 72 72 72
73 73 73 73 73 73 73 73 73 74 74 74 74 74 74 74 74 74 74 75 75 75 75 75 75 75 76 76 76 76 76 76 76 76 76 77 77 77 77 77 77 77 77 77 77 77 77 77 77 78 78 78
78 78 78 78 78 78 78 79 79 79 79 79 79 79 79 79 79 79 79 79 79 79 79 79 80 80 80 80 80 80 80 80 80 80 81 81 81 81 81 81 81 81 81 82 82 82 82 82 83 83 83 83 83 83
83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85 86 86 86 86 86 86 86 87 87 87 87 88 88 88 88 88 89 89 89 89 89
89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 91 91 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 93 93 93 93 94 94 94
94 94 94 94 94 94 94 94 94 94 95 95 95 95 95 95 95 95 95 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 98 98 98 98 98 98 98 98 98 98 98 98 98 98 99 99 99
99 99 99 99 100 100 100 100 100 100 100 100 100 100 100 100

Time taken to sort the array: 2.996 ms

[Done] exited with code=0 in 5.994 seconds
```

3. Write a program to implement Merge Sort.

```
#include <iostream>
#include <cstdlib>
#include <chrono>
#include <limits>
using namespace std;
void merge(int a[], int p, int q, int r)
{
   int n1 = q - p + 1;
   int n2 = r - q;
   int l[n1 + 1];
   int rig[n2 + 1];
   for (int i = 0; i < n1; i++)
   {
      l[i] = a[p + i];
   }
   for (int j = 0; j < n2; j++)
   {
      rig[j] = a[q + 1 + j];
   }
   l[n1] = numeric_limits<int>::max();
   rig[n2] = numeric_limits<int>::max();
   int i = 0, j = 0;
   for (int k = p; k <= r; k++)
   {
      if (l[i] <= rig[j])
      {
```

```
            a[k] = l[i];
            i++;
        }
        else
        {
            a[k] = rig[j];
            j++;
        }
    }
}
void merge_sort(int a[], int p, int r)
{
    if (p < r)
    {
        int q = (p + r) / 2;
        merge_sort(a, p, q);
        merge_sort(a, q + 1, r);
        merge(a, p, q, r);
    }
}
int main()
{
    const int n = 100;
    int a[n];

    for (int i = 0; i < n; i++)
    {
        a[i] = rand() % n + 1;
```

```
    }
    auto start = chrono::high_resolution_clock::now();
    for (int i = 0; i < 1000; i++)
    {
        merge_sort(a, 0, n - 1);
    }
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> elapsed = end - start;
    cout << "Sorted array is: ";
    for (int i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
    cout << "Time taken to sort the array over 1000 loops: " <<
elapsed.count() << " ms" << endl;
    return 0;
}
```

OUTPUT: -

```
[Running] cd "c:\Users\mukes\OneDrive\Desktop\New folder\" && g++ tempCodeRunnerFile.cpp -o
tempCodeRunnerFile && "c:\Users\mukes\OneDrive\Desktop\New folder\"tempCodeRunnerFile
Sorted array is: 1 2 3 4 5 6 6 7 7 9 12 12 13 17 17 19 19 22 23 24 24 24 25 27 27 28 28 30 30 30 30 32
34 35 36 36 37 38 38 39 39 40 41 41 42 42 42 42 43 43 43 45 45 46 47 48 48 49 49 51 54 54 55 57 58 59
60 62 63 63 65 65 65 67 68 68 69 70 70 71 72 74 77 79 79 82 83 83 85 89 91 91 92 92 93 94 95 96 96 100
Time taken to sort the array over 1000 loops: 8.379 ms

[Done] exited with code=0 in 2.095 seconds
```

4. Write a program to implement Quick Sort.

```
#include <iostream>
#include <cstdlib>
#include <chrono>

using namespace std;

int partition(int a[], int p, int r)
{
    int x = a[r];
    int i = p - 1;
    for (int j = p; j < r; j++)
    {
        if (a[j] <= x)
        {
            i++;
            swap(a[i], a[j]);
        }
    }
    swap(a[i + 1], a[r]);
    return i + 1;
}
void quick_sort(int a[], int p, int r)
{
    if (p < r)
    {
        int q = partition(a, p, r);
```

```
        quick_sort(a, p, q - 1);
        quick_sort(a, q + 1, r);
    }
}
int main()
{
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    int a[n];

    srand(time(0));

    for (int i = 0; i < n; i++)
    {
        a[i] = rand() % n + 1;
    }
    auto start = chrono::high_resolution_clock::now();
    quick_sort(a, 0, n - 1);
    auto end = chrono::high_resolution_clock::now();

    chrono::duration<double, nano> elapsed = end - start;

    cout << "\nSorted array is:\n";
    for (int i = 0; i < n; i++)
    {
        cout << a[i] << " ";
```

```
    }
    cout << "\n\nTime taken to sort the array: " << elapsed.count() << " nanoseconds" << endl;


    return 0;
}
```

OUTPUT: -

```
PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the size of the array: 300

Sorted array is:
3 3 5 5 5 8 8 8 9 10 12 14 15 15 16 17 21 21 22 22 23 24 28 30 31 32 32 34 35 38 40 42 43 43 43 44 45 45 46 47 48 48 49 49
49 50 52 52 52 54 55 55 57 57 59 59 61 63 64 64 64 65 67 69 70 71 72 73 73 74 77 77 78 79 79 79 83 83 87 89 89 89 92 95 96
97 97 97 98 99 99 101 101 102 103 105 106 106 107 108 110 111 113 114 115 115 116 116 116 121 122 124 125 125 126 127 128 1
29 130 131 134 136 136 137 137 137 138 140 142 143 144 144 144 145 146 147 147 149 150 151 151 151 152 154 154 154 154 155
155 156 156 159 163 163 163 163 164 166 168 168 168 169 169 169 170 172 173 174 174 175 176 177 177 177 177 179 179 180 180
 182 182 183 183 184 185 185 186 189 190 190 191 193 193 193 196 196 196 199 200 201 202 203 204 205 207 208 209 210 210 21
0 210 210 213 214 214 215 216 216 219 219 220 220 221 222 223 226 226 227 230 231 232 238 238 239 239 240 242 243 245 246 2
46 246 247 247 248 248 248 249 249 250 250 251 251 251 253 253 254 254 261 261 261 263 264 266 267 268 268 269 269 270 272
273 273 275 276 276 277 278 279 279 280 280 284 287 287 290 290 290 290 290 292 293 294 294 295 296 296 297 298 299

Time taken to sort the array: 1.006e+006 nanoseconds
PS C:\Users\mukes\OneDrive\Desktop\New folder>
```

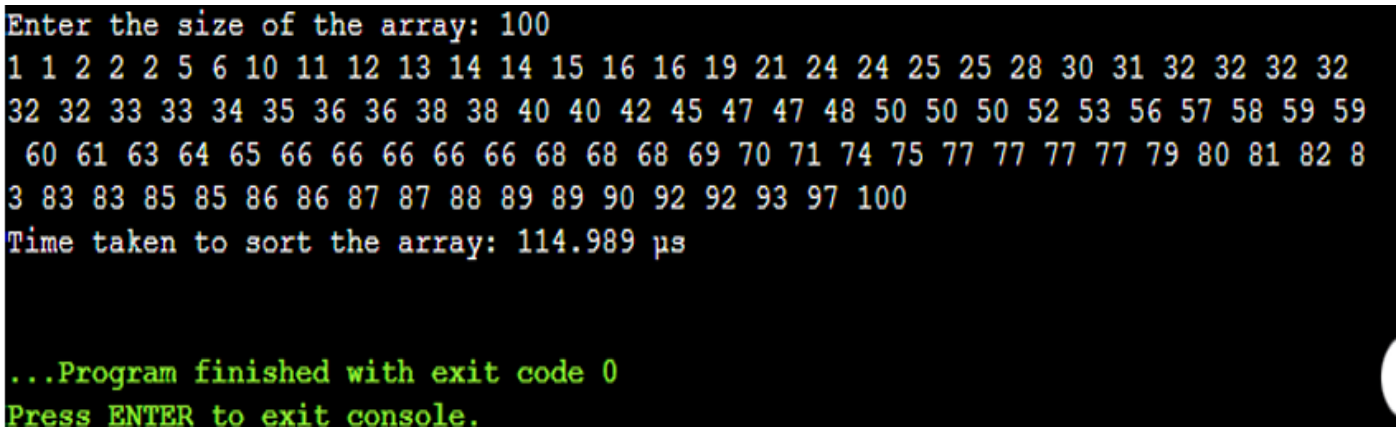
5. Write a program to implement Heap Sort.

```
#include <iostream>
#include <cstdlib>
#include <chrono>
#include <random>
using namespace std;
void max_heapify(int a[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && a[l] > a[largest])
        largest = l;

    if (r < n && a[r] > a[largest])
        largest = r;

    if (largest != i) {
        swap(a[i], a[largest]);
        max_heapify(a, n, largest);
    }
}
void build_max_heap(int a[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        max_heapify(a, n, i);
}
void heap_sort(int a[], int n) {
```

```
    build_max_heap(a, n);
    for (int i = n - 1; i > 0; i--) {
        swap(a[0], a[i]);
        max_heapify(a, i, 0);
    }
}
int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    int a[n];

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(1, n);

    for (int i = 0; i < n; i++)
        a[i] = dis(gen);

    auto start = chrono::high_resolution_clock::now();
    heap_sort(a, n);
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double, micro> elapsed = end - start;
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << "\nTime taken to sort the array: " << elapsed.count()
<< " µs" << endl;
    return 0;
```

}

OUTPUT: -


Enter the size of the array: 100
1 1 2 2 2 5 6 10 11 12 13 14 14 15 16 16 19 21 24 24 25 25 28 30 31 32 32 32 32
32 32 33 33 34 35 36 36 38 38 40 40 42 45 47 47 48 50 50 50 52 53 56 57 58 59 59
60 61 63 64 65 66 66 66 66 66 68 68 68 69 70 71 74 75 77 77 77 77 79 80 81 82 8
3 83 83 85 85 86 86 87 87 88 89 89 90 92 92 93 97 100
Time taken to sort the array: 114.989 µs
...Program finished with exit code 0
Press ENTER to exit console.

6. Write a program to implement BFS Traversal.

```
#include <iostream>
#include <queue>
#include <vector>
#include <limits>

using namespace std;
class Graph {
    int V; // Number of vertices
    vector<vector<int>> adj; // Adjacency list

public:
    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u); // Assuming an undirected graph
    }
    void BFS(int s) {
        vector<string> color(V, "WHITE");
        vector<int> d(V, numeric_limits<int>::max());
        vector<int> pi(V, -1);
        color[s] = "GRAY";
        d[s] = 0;
        pi[s] = -1;
        queue<int> Q;
        Q.push(s);
```

```
        while (!Q.empty()) {
            int u = Q.front();
            Q.pop();

            for (int v : adj[u]) {
                if (color[v] == "WHITE") {
                    color[v] = "GRAY";
                    d[v] = d[u] + 1;
                    pi[v] = u;
                    Q.push(v);
                }
            }
            color[u] = "BLACK";
        }
        // Displaying the results
        cout << "Vertex\tDistance\tParent" << endl;
        for (int i = 0; i < V; i++) {
            cout << i << "\t" << (d[i] ==
numeric_limits<int>::max() ? -1 : d[i]) << "\t\t" << pi[i] <<
endl;
        }
    }
};
int main() {
    int V = 6; // Number of vertices
    Graph g(V);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
```

```
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);
    cout << "BFS starting from vertex 0:" << endl;
    g.BFS(0);
    return 0;
}
```

OUTPUT: -

```
PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
BFS starting from vertex 0:
Vertex  Distance        Parent
0       0               -1
1       1               0
2       1               0
3       2               1
4       2               1
5       3               3
```

7. Write a program to implement DFS Traversal.

```
#include <iostream>
#include <vector>
#include <limits>

using namespace std;
class Graph {
    int V; // Number of vertices
    vector<vector<int>> adj; // Adjacency list
    vector<string> color;
    vector<int> pi;
    vector<int> d;
    vector<int> f;
    int time;

public:
    Graph(int V) {
        this->V = V;
        adj.resize(V);
        color.resize(V, "WHITE");
        pi.resize(V, -1);
        d.resize(V, 0);
        f.resize(V, 0);
        time = 0;
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u); // Assuming an undirected graph
    }
```

```
    void DFS() {
        for (int u = 0; u < V; u++) {
            color[u] = "WHITE";
            pi[u] = -1;
        }
        time = 0;
        for (int u = 0; u < V; u++) {
            if (color[u] == "WHITE") {
                DFS_VISIT(u);
            }
        }
        // Displaying the results
        cout << "Vertex\tDiscovery Time\tFinish Time\tParent" <<
endl;
        for (int i = 0; i < V; i++) {
            cout << i << "\t" << d[i] << "\t\t" << f[i] << "\t\t"
<< pi[i] << endl;
        }
    }
    void DFS_VISIT(int u) {
        color[u] = "GRAY";
        time++;
        d[u] = time;
        for (int v : adj[u]) {
            if (color[v] == "WHITE") {
                pi[v] = u;
                DFS_VISIT(v);
            }
```

```
        }
        color[u] = "BLACK";
        time++;
        f[u] = time;
    }
};
int main() {
    int V = 6; // Number of vertices
    Graph g(V);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);

    cout << "DFS traversal of the graph:" << endl;
    g.DFS();
    return 0;
}
```

OUTPUT: -

```
PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
DFS traversal of the graph:
Vertex  Discovery Time  Finish Time     Parent
0       1              12              -1
1       2              11              0
2       6              7               4
3       3              10              1
4       5              8               5
5       4              9               3
PS C:\Users\mukes\OneDrive\Desktop\New folder>
```

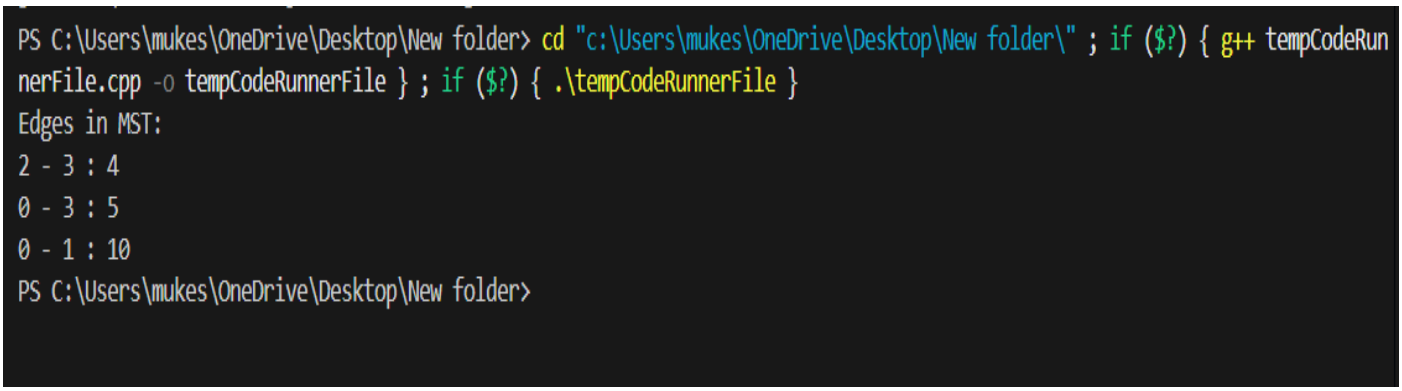
8. Write a program to implement Kruskal Algorithm for minimum spanning tree.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Edge {
    int u, v, weight;
    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }
};
class DisjointSet {
    vector<int> parent, rank;
public:
    DisjointSet(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; ++i)
            parent[i] = i;
    }
    int findSet(int v) {
        if (v != parent[v])
            parent[v] = findSet(parent[v]);
        return parent[v];
    }
    void unionSets(int u, int v) {
```

```
        u = findSet(u);
        v = findSet(v);
        if (u != v) {
            if (rank[u] < rank[v])
                parent[u] = v;
            else if (rank[u] > rank[v])
                parent[v] = u;
            else {
                parent[v] = u;
                rank[u]++;
            }
        }
    }
};
vector<Edge> kruskal(int V, vector<Edge>& edges) {
    sort(edges.begin(), edges.end());
    DisjointSet ds(V);
    vector<Edge> mst;

    for (const Edge& e : edges) {
        if (ds.findSet(e.u) != ds.findSet(e.v)) {
            mst.push_back(e);
            ds.unionSets(e.u, e.v);
        }
    }
    return mst;
}
int main() {
```

```
    int V = 4;
    vector<Edge> edges = {
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };
    vector<Edge> mst = kruskal(V, edges);
    cout << "Edges in MST:\n";
    for (const Edge& e : mst) {
        cout << e.u << " - " << e.v << " : " << e.weight << endl;
    }
return 0;
}
```

OUTPUT: -



9. Write a program to implement Knapsack Problem.

a.) Fractional:

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

struct Item
{
   int index;
   float profit, weight, ratio;
};
bool compare(Item a, Item b)
{
   return a.ratio > b.ratio;
}
int main()
{
   const int n = 5;
   float maxWeight = 20;
   float totalProfit = 0;

   float p[n] = {10, 20, 12, 14, 22};
   float w[n] = {5, 4, 4, 5, 6};
   Item items[n];

   for (int i = 0; i < n; i++)
   {
      items[i].index = i + 1;
```

```
        items[i].profit = p[i];
        items[i].weight = w[i];
        items[i].ratio = p[i] / w[i];
    }
    sort(items, items + n, compare);
    cout << "Items sorted by profit/weight ratio:\n";
    for (int i = 0; i < n; i++)
    {
        cout << "Item " << items[i].index << ": Profit = " <<
items[i].profit
             << ", Weight = " << items[i].weight
             << ", P/W = " << items[i].ratio << endl;
    }
    cout << "\nSelecting items for the knapsack:\n";
    for (int i = 0; i < n && maxWeight > 0; i++)
    {
        if (items[i].weight <= maxWeight)
        {
            cout << "Taking full Item " << items[i].index << endl;
            maxWeight -= items[i].weight;
            totalProfit += items[i].profit;
        }
        else
        {
            float fraction = maxWeight / items[i].weight;
            cout << "Taking " << fraction * 100 << "% of Item " <<
items[i].index << endl;
            totalProfit += items[i].profit * fraction;
```

```
            maxWeight = 0;
        }
    }
    cout << "\nTotal profit earned = " << totalProfit << endl;
    return 0;
}
```

OUTPUT: -

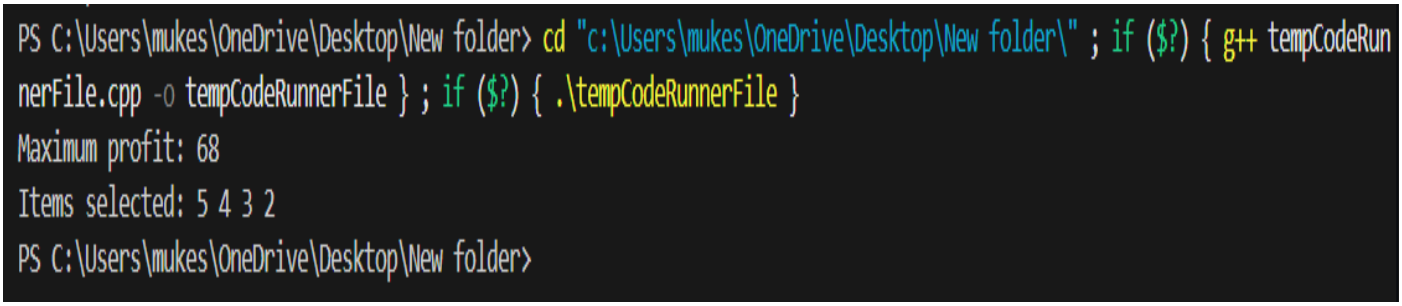
```
PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Items sorted by profit/weight ratio:
Item 2: Profit = 20, Weight = 4, P/W = 5
Item 5: Profit = 22, Weight = 6, P/W = 3.66667
Item 3: Profit = 12, Weight = 4, P/W = 3
Item 4: Profit = 14, Weight = 5, P/W = 2.8
Item 1: Profit = 10, Weight = 5, P/W = 2

Selecting items for the knapsack:
Taking full Item 2
Taking full Item 5
Taking full Item 3
Taking full Item 4
Taking 20% of Item 1

Total profit earned = 70
PS C:\Users\mukes\OneDrive\Desktop\New folder>
```

b.) 0/1:

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

int main() {
    const int n = 5;
    int W = 20;
    int profit[n + 1] = {0, 10, 20, 12, 14, 22};
    int weight[n + 1] = {0, 5, 4, 4, 5, 6};
    int dp[n + 1][W + 1];

    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weight[i] <= w)
                dp[i][w] = max(profit[i] + dp[i - 1][w - weight[i]], dp[i -
1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }
    cout << "Maximum profit: " << dp[n][W] << endl;
    int w = W;
    cout << "Items selected: ";
    for (int i = n; i > 0 && w > 0; i--) {
        if (dp[i][w] != dp[i - 1][w]) {
```

```
            cout << i << " ";
            w -= weight[i];
        }
    }
    cout << endl;

    return 0;
}
```

OUTPUT: -

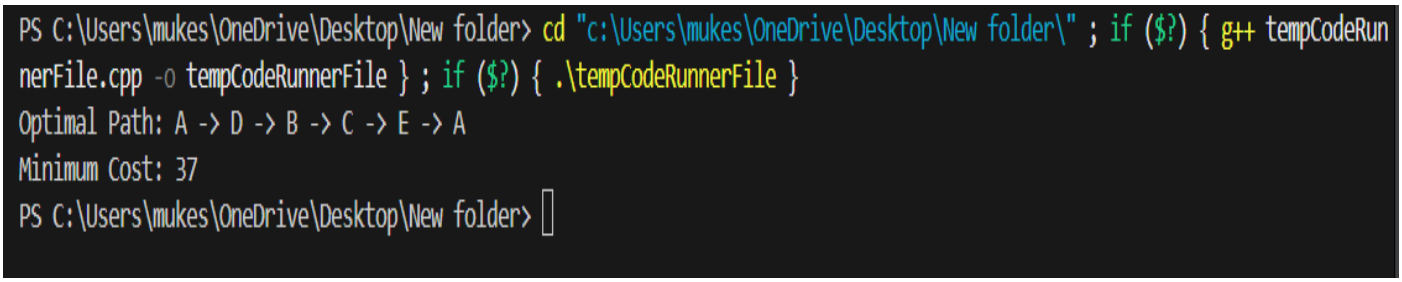


10. Write a program to implement Traveling Salesman Problem.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

int dist[5][5] = {
    {0, 14, 12, 10, 7},
    {14, 0, 9, 6, 13},
    {12, 9, 0, 8, 5},
    {10, 6, 8, 0, 11},
    {7, 13, 5, 11, 0}
};
string nodeName[5] = {"A", "B", "C", "D", "E"};

int main() {
    vector<int> nodes = {1, 2, 3, 4};
    int minCost = INT_MAX;
    vector<int> bestPath;

    do {
        int cost = 0;
        int prev = 0; // Start from A

        for (int i = 0; i < nodes.size(); i++) {
            cost += dist[prev][nodes[i]];
            prev = nodes[i];
        }
```

```
        cost += dist[prev][0]; // Return to A

        if (cost < minCost) {
            minCost = cost;
            bestPath = nodes;
        }
    } while (next_permutation(nodes.begin(), nodes.end()));

cout << "Optimal Path: A -> ";
for (int node : bestPath) {
cout << nodeName[node] << " -> ";
}
cout << "A" << endl;
cout << "Minimum Cost: " << minCost << endl;
return 0;
}
```

OUTPUT: -



11. Write a program to implement Binary Search Tree.

a.) Traversal

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

using namespace std;

struct node{
   int key;
   struct node* left;
   struct node* right;
   node(int val){
      key=val;
      left=right=nullptr;
   }
};
node* insert(node* root,int val){
   if(root==nullptr){
      return new node(val);
   }
   if(val<root->key){
      root->left=insert(root->left,val);
   }
   else{
      root->right=insert(root->right,val);
   }
   return root;
}
```

```
void inorder(node* root){
    if(root!=nullptr){
        inorder(root->left);
        cout<<root->key<<" ";
        inorder(root->right);
    }
}
void preOrder(node* root){
    if(root!=nullptr){
        cout<<root->key<<" ";
        preOrder(root->left);
        preOrder(root->right);
    }
}
void postOrder(node* root){
    if(root!=nullptr){
        postOrder(root->left);
        postOrder(root->right);
        cout<<root->key<<" ";
    }
}
int main()
{
    node* root=nullptr;
    root=insert(root,15);
    insert(root,17);
    insert(root,8);
```

```
    insert(root,7);
    insert(root,9);
    insert(root,10);
    insert(root,12);
    insert(root,13);
    insert(root,16);
    insert(root,19);
    insert(root,11);

    cout<<"Inorder traversal: "<<endl;
    inorder(root);
    cout<<endl<<endl;
    cout<<"\nPre Order traversal: "<<endl;
    preOrder(root);
    cout<<endl<<endl;
    cout<<"\nPost Order traversal: "<<endl;
    postOrder(root);
    return 0;
}
```

OUTPUT: -

```
PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Inorder traversal:
7 8 9 10 11 12 13 15 16 17 19


Pre Order traversal:
15 8 7 9 10 12 11 13 17 16 19


Post Order traversal:
7 11 13 12 10 9 8 16 19 17 15
PS C:\Users\mukes\OneDrive\Desktop\New folder>
```

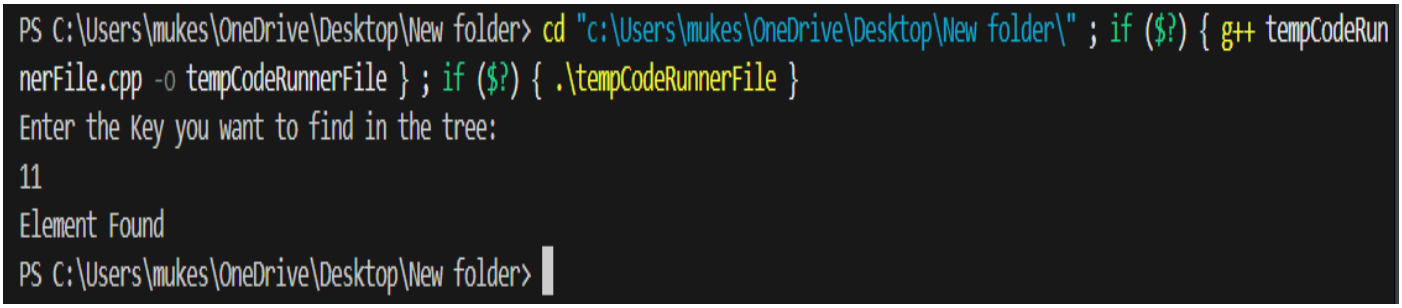
b.)Tree Search

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

using namespace std;

struct node
{
   int key;
   struct node *left;
   struct node *right;
   node(int val)
   {
      key = val;
      left = right = nullptr;
   }
};
node *insert(node *root, int val)
{
   if (root == nullptr)
   {
      return new node(val);
   }
   if (val < root->key)
   {
      root->left = insert(root->left, val);
   }
   else
   {
```

```
        root->right = insert(root->right, val);
    }
    return root;
}
node *treeSearch(node *root, int val)
{
    if (root == nullptr || val == root->key)
    {
        return root;
    }
    if (val < root->key)
    {
        return treeSearch(root->left, val);
    }
    else
    {
        return treeSearch(root->right, val);
    }
}
int main()
{
    node *root = nullptr;
    root = insert(root, 15);
    insert(root, 17);
    insert(root, 8);
    insert(root, 7);
    insert(root, 9);
    insert(root, 10);
```

```
    insert(root, 12);
    insert(root, 13);
    insert(root, 16);
    insert(root, 19);
    insert(root, 11);
    int n;
    cout << "Enter the Key you want to find in the tree: " << endl;
    cin >> n;
    node *get = treeSearch(root, n);
    if (get == nullptr)
    {
        cout << "Element not found";
    }
    else
    {
        cout << "Element Found";
    }
}
```

OUTPUT: -

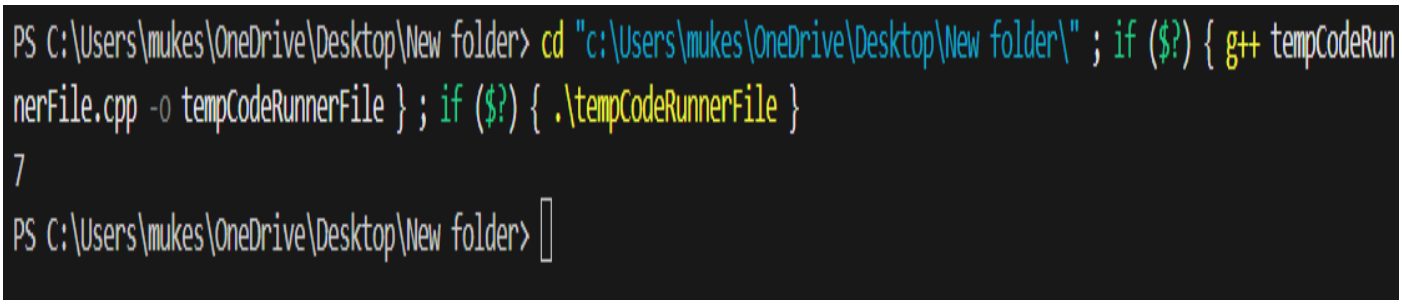


c.) Minimum element of the tree

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

using namespace std;

struct node
{
   int key;
   struct node *left;
   struct node *right;
   node(int val)
   {
      key = val;
      left = right = nullptr;
   }
};
node *insert(node *root, int val)
{
   if (root == nullptr)
   {
      return new node(val);
   }
   if (val < root->key)
   {
      root->left = insert(root->left, val);
   }
   else
   {
```

```
        root->right = insert(root->right, val);
    }
    return root;
}
node *treeMin(node *root){
    while(root->left!=nullptr){
        root=root->left;
    }
    return root;
}
int main()
{
    node *root = nullptr;
    root = insert(root, 15);
    insert(root, 17);
    insert(root, 8);
    insert(root, 7);
    insert(root, 9);
    insert(root, 10);
    insert(root, 12);
    insert(root, 13);
    insert(root, 16);
    insert(root, 19);
    insert(root, 11);
    node *get=treeMin(root);
    cout<<get->key;
}
```

OUTPUT: -


PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
7
PS C:\Users\mukes\OneDrive\Desktop\New folder>

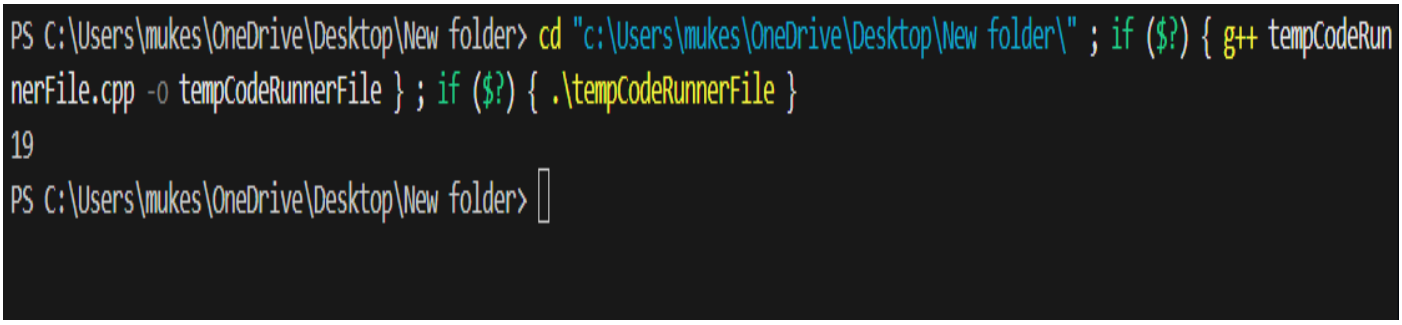
d.) Maximum element of the tree

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

using namespace std;

struct node
{
   int key;
   struct node *left;
   struct node *right;
   node(int val)
   {
      key = val;
      left = right = nullptr;
   }
};
node *insert(node *root, int val)
{
   if (root == nullptr)
   {
      return new node(val);
   }
   if (val < root->key)
   {
      root->left = insert(root->left, val);
   }
```

```
    else
    {
        root->right = insert(root->right, val);
    }
    return root;
}
node *treeMax(node *root){
    while(root->right!=nullptr){
        root=root->right;
    }
    return root;
}
int main()
{
    node *root = nullptr;
    root = insert(root, 15);
    insert(root, 17);
    insert(root, 8);
    insert(root, 7);
    insert(root, 9);
    insert(root, 10);
    insert(root, 12);
    insert(root, 13);
    insert(root, 16);
    insert(root, 19);
    insert(root, 11);
    node *get=treeMax(root);
    cout<<get->key;
```

}

OUTPUT: -


PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
19
PS C:\Users\mukes\OneDrive\Desktop\New folder>

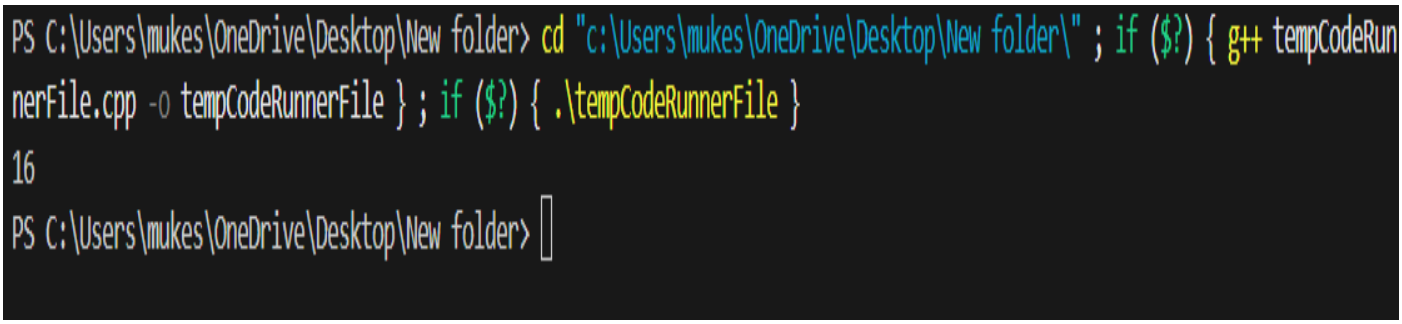
e.) Successor of given element in the tree

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

using namespace std;
struct node
{
   int key;
   struct node *left;
   struct node *right;
   struct node *p;
   node(int val)
   {
      key = val;
      left = right = nullptr;
   }
};
node *insert(node *root, int val)
{
   if (root == nullptr)
   {
      return new node(val);
   }
   if (val < root->key)
   {
      root->left = insert(root->left, val);
      root->left->p=root;
   }
```

```
    else
    {
        root->right = insert(root->right, val);
        root->right->p=root;
    }
    return root;
}
node *treeMin(node *root){
    while(root->left!=nullptr){
        root=root->left;
    }
    return root;
}
node *treeSuccessor(node *root){
    if(root->right!=nullptr){
        return treeMin(root->right);
    }
    node *y=root->p;
    while (y!=nullptr && root==y->left){
        root=y;
        y=y->p;
    }
    return y;
}
int main()
{
    node *root = nullptr;
    root = insert(root, 15);
```

```
    insert(root, 17);
    insert(root, 8);
    insert(root, 7);
    insert(root, 9);
    insert(root, 10);
    node *a=insert(root, 12);
    insert(root, 13);
    insert(root, 16);
    insert(root, 19);
    insert(root, 11);
    node *get=treeSuccessor(a);
    cout<<get->key;
}
```

OUTPUT: -



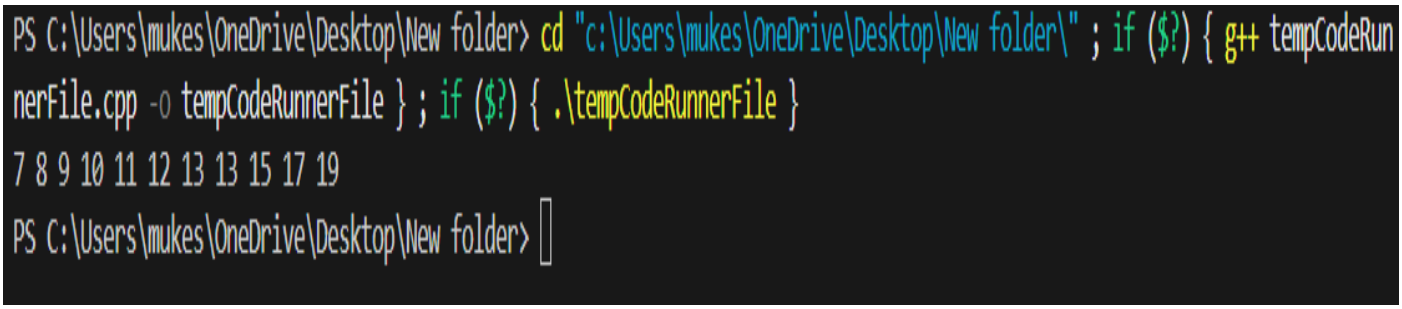
f.) To insert element in the tree

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

using namespace std;

struct node
{
   int key;
   struct node *left;
   struct node *right;
   struct node *parent;
   node(int val)
   {
      key = val;
      left = right = nullptr;
   }
};
node *insert(node *root, node *z)
{
   node *x = root;
   node *y = nullptr;
   while (x != nullptr)
   {
      y = x;
      if (z->key < x->key)
      {
         x = x->left;
      }
```

```
        else
        {
            x = x->right;
        }
    }
    z->parent = y;
    if (y == nullptr)
    {
        root = z;
    }
    else if (z->key < y->key)
    {
        y->left = z;
    }
    else
    {
        y->right = z;
    }
    return root;
}
void inorder(node* root){
    if(root!=nullptr){
        inorder(root->left);
        cout<<root->key<<" ";
        inorder(root->right);
    }
}
int main()
```

```
{
    node *root = nullptr;
    root=insert(root, new node (15));
    insert(root, new node(17));
    insert(root, new node (8));
    insert(root, new node (7));
    insert(root, new node (9));
    insert(root, new node (10));
    insert(root, new node (12));
    insert(root, new node (13));
    insert(root, new node (13));
    insert(root, new node (19));
    insert(root, new node (11));
    inorder(root);
    return 0;
}
```

OUTPUT: -



g.) To delete an element in the tree

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

using namespace std;

struct node
{
   int key;
   struct node *left;
   struct node *right;
   struct node *p;
   node(int val)
   {
      key = val;
      left = right = nullptr;
   }
};
node *insert(node *root, node *z)
{
   node *x = root;
   node *y = nullptr;
   while (x != nullptr)
   {
      y = x;
      if (z->key < x->key)
      {
         x = x->left;
```

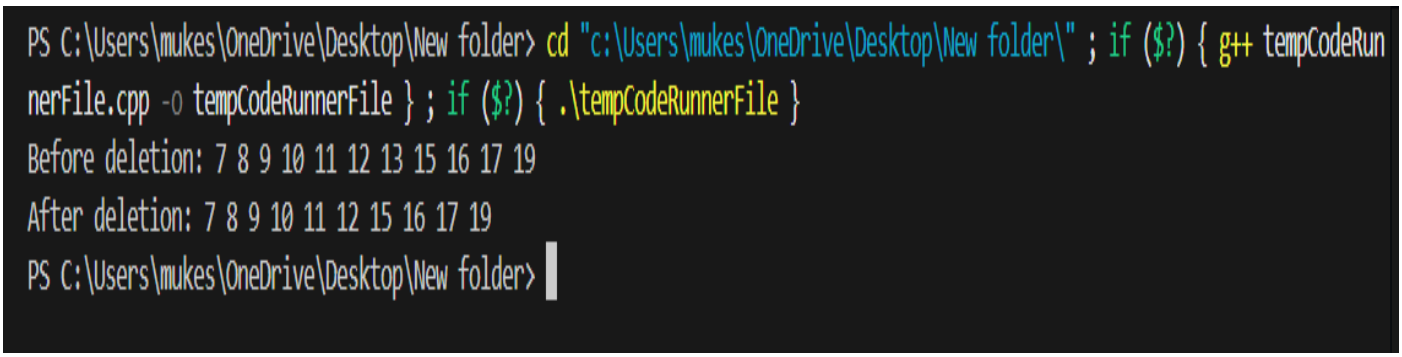
```
        }
        else
        {
            x = x->right;
        }
    }
    z->p = y;
    if (y == nullptr)
    {
        root = z;
    }
    else if (z->key < y->key)
    {
        y->left = z;
    }
    else
    {
        y->right = z;
    }
    return z;
}
node *treeMin(node *root){
    while(root->left!=nullptr){
        root=root->left;
    }
    return root;
}
```

```
node *treeSuccessor(node *root){
    if(root->right!=nullptr){
        return treeMin(root->right);
    }
    node *y=root->p;
    while (y!=nullptr && root==y->right){
        root=y;
        y=y->p;
    }
    return y;
}
node *tree_delete(node *root, node *z){
    node *y = nullptr;
    node *x = nullptr;
    if(z->left==nullptr || z->right==nullptr){
        y=z;
    }
    else{
        y= treeSuccessor(z);
    }
    if(y->left!=nullptr){
        x=y->left;
    }
    else{
        x=y->right;
    }
    if(x!=nullptr){
```

```
        x->p=y->p;
    }
    if(y->p==nullptr){
        root=x;
    }
    else if (y==y->p->left){
        y->p->left=x;
    }
    else{
        y->p->right=x;
    }
    if(y!=z){
        z->key=y->key;
    }
    return y;
}
void inorder(node* root){
    if(root!=nullptr){
        inorder(root->left);
        cout<<root->key<<" ";
        inorder(root->right);
    }
}
int main()
{
    node *root = nullptr;
    root=insert(root, new node (15));
    node *n1=insert(root, new node(17));
```

```
    node *n2=insert(root, new node (8));
    node *n3=insert(root, new node (7));
    node *n4=insert(root, new node (9));
    node *n5=insert(root, new node (10));
    node *n6=insert(root, new node (12));
    node *n7=insert(root, new node (16));
    node *n8=insert(root, new node (13));
    node *n9=insert(root, new node (19));
    node *n10=insert(root, new node (11));
    cout<<"Before deletion: ";
    inorder(root);
    tree_delete(root,n8);
    cout<<endl;
    cout<<"After deletion: ";
    inorder(root);
    return 0;
}
```

OUTPUT: -



12. Write a program to find minimum and maximum element from a given set of numbers.

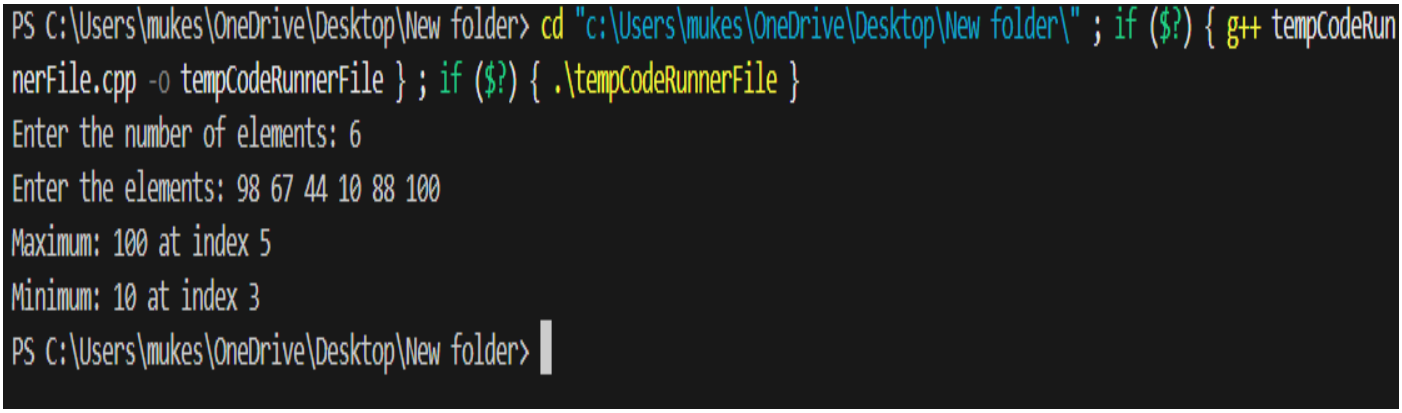
```
#include <iostream>
#include <conio.h>
using namespace std;
void min_max(int arr[], int n)
{
   int i, max, min, p1, p2;

   if (n % 2 == 1)
   {
      min = max = arr[0];
      p1 = p2 = 0;

      for (i = 1; i < n - 1; i += 2)
      {
         if (arr[i] > arr[i + 1])
         {
            if (arr[i] > max)
            {
               max = arr[i];
               p1 = i;
            }
            if (arr[i + 1] < min)
            {
               min = arr[i + 1];
               p2 = i + 1;
            }
         }
```

```
        else
        {
            if (arr[i + 1] > max)
            {
                max = arr[i + 1];
                p1 = i + 1;
            }
            if (arr[i] < min)
            {
                min = arr[i];
                p2 = i;
            }
        }
    }
}
else
{
    if (arr[0] > arr[1])
    {
        max = arr[0]; p1 = 0;
        min = arr[1]; p2 = 1;
    }
    else
    {
        max = arr[1]; p1 = 1;
        min = arr[0]; p2 = 0;
    }
    for (int i = 2; i < n - 1; i += 2)
```

```
{
    if (arr[i] > arr[i + 1])
    {
        if (arr[i] > max)
        {
            max = arr[i];
            p1 = i;
        }
        if (arr[i + 1] < min)
        {
            min = arr[i + 1];
            p2 = i + 1;
        }
    }
    else
    {
        if (arr[i + 1] > max)
        {
            max = arr[i + 1];
            p1 = i + 1;
        }
        if (arr[i] < min)
        {
            min = arr[i];
            p2 = i;
        }
    }
}
```

```
    }
    cout << "Maximum: " << max << " at index " << p1 << endl;
    cout << "Minimum: " << min << " at index " << p2 << endl;
}
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    min_max(arr, n);
    return 0;
}
```

OUTPUT: -



13. Write a program to implement 8-Queen Problem.

```
#include <iostream>
#define N 8
using namespace std;

bool solutionFound = false;
void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            cout << (board[i][j] ? "Q " : ". ");
        cout << endl;
    }
    cout << endl;
}
bool isSafe(int board[N][N], int row, int col) {
    for (int i = 0; i < col; i++) {
        for (int j = 0; j < N; j++) {
            if (board[j][i]) {
                if (j == row || (j - i == row - col) || (j + i ==
row + col))
                    return false;
            }
        }
    }
    return true;
}
bool solveNQueens(int board[N][N], int col) {
    if (col >= N) {
```

```
        printSolution(board);
        solutionFound = true;
        return true;
    }
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQueens(board, col + 1)) return true;
            board[i][col] = 0;
}
}
return false;
}
int main() {
int board[N][N] = {0};
if (!solveNQueens(board, 0))
cout << "No solution exists\n";
return 0;
}
```

OUTPUT: -

```
PS C:\Users\mukes\OneDrive\Desktop\New folder> cd "c:\Users\mukes\OneDrive\Desktop\New folder\" ; if ($?) { g++ tempCodeRun
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Q . . . . . .
. . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .

PS C:\Users\mukes\OneDrive\Desktop\New folder>
```

14. Write a program to implement Sum of Subset.

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

int main()
{
    int a[4] = {5, 7, 8, 15};
    int n;
    cout << "Enter the number: ";
    cin >> n;

    bool found = false;
    for (int mask = 1; mask < (1 << 4); mask++)
    {
        int sum = 0;
        string subset = "{";
        bool first = true;

        for (int i = 0; i < 4; i++)
        {
            if (mask & (1 << i))
            {
                sum += a[i];
                if (!first)
                    subset += ", ";
                subset += to_string(a[i]);
                first = false;
            }
        }
```

```
        subset += "}";
        if (sum == n)
        {
            cout << "Subset with sum " << n << " is " << subset
<< endl;
            found = true;
        }
    }
if (!found)
{
cout << "No subset found with sum " << n << "." << endl;
}
return 0;
}
```

OUTPUT: -