Q2 Implement Linear Search and Binary Search. Analyze their time complexities.

TIME COMPLEXITY:
1. Binary Search: O(log(n)) (valid for only ordered lists)
1. Linear Search: O(n)

SPACE COMPLEXITY: O(1)

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <chrono>

using namespace std;


int linearSearch(vector<int> & A, int val)
{
    for(int i = 0; i < A.size(); ++i)
        if(A[i] == val)
            return i;
    return -1;
}

int binarySearch(vector<int> & A, int start, int end, int val)
{
    if(start > end)
        return -1;
    int mid = start + (end - start) / 2;
    if(val < A[mid])
        return binarySearch(A, start, mid - 1, val);
    else if(val > A[mid])
        return binarySearch(A, mid + 1, end, val);
    else return mid;
}

int main()
{
    vector<int> A(1000);
    for(int i = 0; i < 1000; ++i)
        A[i] = rand() % 10000;
    sort(A.begin(), A.end());
    cout << "INPUT ARRAY:\n";
    for(int i = 0; i < 1000; ++i)
        cout << A[i] << " ";
    A[998] = 9940;
    cout << endl << endl;
    auto start_linear = chrono::high_resolution_clock::now();
    cout << "Linear Search index for 9940: " << linearSearch(A, 9940) <<
     endl;
    auto end_linear = chrono::high_resolution_clock::now();
    cout << "Binary Search index for 9940: " << binarySearch(A, 0,
     int(A.size()) - 1, 9940) << endl << endl;
    auto stop = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::microseconds>(end_linear
     - start_linear);
    cout << "TIME TAKEN(LINEAR SEARCH): " << duration.count() << "
     microsecond" << endl;
    duration = chrono::duration_cast<chrono::microseconds>(stop -
     end_linear);
    cout << "TIME TAKEN(BINARY SEARCH): " << duration.count() << "
     microsecond" << endl;
}
```

```
OUTPUT:

INPUT ARRAY:
0 33 98 2217 7533 7544 7561 7584 7633 7638 7656 7670 7683 7692 7696 7698
 7699 7704 7709 7709 7719 7722 7730 7732 7746 7747 7758 7773 7781 7787 7801
 7826 7827 7844 7844 7844 7854 7865 7874 7875 7876 7878 7879 7904 7920 7923
 7929 7936 7939 7942 7944 7947 7968 7972 7987 7992 7995 8008 8013 8014 8015
 8024 8028 8034 8048 8060 8080 8111 8124 8138 8142 8144 8150 8155 8163 8165
 8177 8192 8209 8211 ... 9358 9364 9378 9397 9415 9419 9425 9451 9453 9461
 9498 9503 9505 9506 9517 9518 9530 9531 9536 9543 9549 9551 9553 9560 9565
 9579 9589 9590 9593 9594 9603 9605 9627 9649 9651 9662 9669 9684 9715 9719
 9745 9759 9789 9790 9798 9814 9816 9818 9821 9824 9847 9852 9860 9870 9874
 9879 9879 9880 9891 9901 9911 9915 9940 9943 9949 9955 9956 9970 9997

Linear Search index for 9940: 993
Binary Search index for 9940: 993

TIME TAKEN(LINEAR SEARCH): 9 microsecond
TIME TAKEN(BINARY SEARCH): 4 microsecond
Program ended with exit code: 0
```

Q8. Implement Bellman Ford Algorithm. Analyze its time complexity.

TIME COMPLEXITY: O(V * E)

SPACE COMPLEXITY: O(V)

where V is the number of the vertices and E is the number of edges.

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>

using namespace std;

class Edge
{
public:
    int source, destination, weight;
    Edge(int source, int destination, int weight) : source(source),
     destination(destination), weight(weight) {}
};
class Graph
{
public:
    int V, E;
    vector<Edge> edges;
    vector<int> distance;
    Graph(int V, int E) : V(V), E(E), distance(vector<int>(V, INT_MAX))
    {
        edges.reserve(E);
    }
    void addEdge(int source, int destination, int weight)
    {
        edges.push_back(Edge(source, destination, weight));
    }
    void print()
    {
        for(int i = 0; i < this -> E; ++i)
            cout << edges[i].source << "\t" << edges[i].destination << "\t"
             << edges[i].weight << endl;
    }
    void printGraph()
    {
        cout << "VERTEX\tDISTANCE" << endl;
        for(int i = 0; i < this -> V; ++i)
            cout << "  " << i << "\t\t  " << distance[i] << endl;
    }
    void bellmanFord(int source)
    {
        distance[source] = 0;
        for (int i = 1; i <= V - 1; i++)
```

```
            for (int j = 0; j < E; j++)
            {
                int u = edges[j].source;
                int v = edges[j].destination;
                int weight = edges[j].weight;
                if (distance[u] != INT_MAX)
                    distance[v] = min(distance[u] + weight, distance[v]);
            }
        for (int i = 0; i < E; i++)
        {
            int u = edges[i].source;
            int v = edges[i].destination;
            int weight = edges[i].weight;
            if (distance[u] != INT_MAX && distance[u] + weight <
             distance[v])
            {
                cout << "Graph contains negative weight cycle" << endl;
                return;
            }
        }
        printGraph();
    }
};
int main()
{
    Graph graph(5, 8);
    graph.addEdge(0, 1, -1); graph.addEdge(0, 2, 4); graph.addEdge(1, 3,
     2); graph.addEdge(1, 2, 3); graph.addEdge(1, 4, 2); graph.addEdge(3,
     2, 5); graph.addEdge(3, 1, 1); graph.addEdge(4, 3, -3);
    cout << "INPUT GRAPH: " << endl;
    graph.print();
    graph.bellmanFord(0);
    return 0;
}
```

OUTPUT:

```
INPUT GRAPH:
    0    1    -1
    0    2    4
    1    3    2
    1    2    3
    1    4    2
    3    2    5
    3    1    1
    4    3    -3

VERTEX    DISTANCE
  0          0
  1          -1
  2          2
  3          -2
  4          1
Program ended with exit code: 0
```

Q1 c) Implement Bubble Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY:

1. Best Case: O(n)
2. Worst Case: O(n ^ 2)
3. Average Case: O(n ^ 2)

SPACE COMPLEXITY: O(1)

```
#include <iostream>
#include <vector>
#include <random>
using namespace std;
void bubbleSort(vector<int> & A)
{
    int n = int(A.size());
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n - i - 1; ++j)
            if (A[j] > A[j + 1])
                swap(A[j], A[j + 1]);
}
int main()
{
    vector<int> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = rand() % 1000;
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    bubbleSort(A);
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
}
```

```
OUTPUT:
INITIAL STATE OF ARRAY:
807 249 73 658 930 272 544 878 923 709 440 165 492 42 987 503 327 729 840
 612 303 169 709 157 560 933 99 278 816 335 97 826 512 267 810 633 979 149
 579 821 967 672 393 336 485 745 228 91 194 357 1 153 708 944 668 490 124
 196 530 903 722 666 549 24 801 853 977 408 228 933 298 981 635 13 865 814
 63 536 425 669 115 94 629 501 517 195 105 404 451 298 188 123 505 882 752
 566 716 337 438 144

SORTED ARRAY:
1 13 24 42 63 73 91 94 97 99 105 115 123 124 144 149 153 157 165 169 188
 194 195 196 228 228 249 267 272 278 298 298 303 327 335 336 337 357 393
 404 408 425 438 440 451 485 490 492 501 503 505 512 517 530 536 544 549
 560 566 579 612 629 633 635 658 666 668 669 672 708 709 709 716 722 729
 745 752 801 807 810 814 816 821 826 840 853 865 878 882 903 923 930 933
 933 944 967 977 979 981 987
Program ended with exit code: 0
```

Q1 d) Implement Bucket Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY:
1. BEST CASE: O(n) if the input numbers are in uniform distribution
2. WORST CASE: O(n ^ 2)

SPACE COMPLEXITY: O(n)

```
#include <iostream>
#include <vector>
#include <random>


using namespace std;



void bucketSort(vector<double> & A)
{
    int n = int(A.size());

    vector<double> bucket[n];

    for(int i = 0; i < n; i++)
        bucket[int(n * A[i])].push_back(A[i]);

    for(int i = 0; i < n; i++)
         sort(bucket[i].begin(), bucket[i].end());

    int index = 0;

    for(int i = 0; i < n; i++)
        for(int j = 0; j < bucket[i].size(); j++)
            A[index++] = bucket[i][j];
}


int main()
{
    vector<double> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = ((double) rand() / (RAND_MAX));
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    bucketSort(A);
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl;
}
```

```
OUTPUT:

INITIAL STATE OF ARRAY:
7.82637e-06 0.131538 0.755605 0.45865 0.532767 0.218959 0.0470446 0.678865
 0.679296 0.934693 0.383502 0.519416 0.830965 0.0345721 0.0534616 0.5297
 0.671149 0.00769819 0.383416 0.0668422 0.417486 0.686773 0.588977 0.930436
 0.846167 0.526929 0.0919649 0.653919 0.415999 0.701191 0.910321 0.762198
 0.262453 0.0474645 0.736082 0.328234 0.632639 0.75641 0.991037 0.365339
 0.247039 0.98255 0.72266 0.753356 0.651519 0.0726859 0.631635 0.884707
 0.27271 0.436411 0.766495 0.477732 0.237774 0.274907 0.359265 0.166507
 0.486517 0.897656 0.909208 0.0605643 0.904653 0.504523 0.516292 0.319033
 0.986642 0.493977 0.266145 0.0907329 0.947764 0.0737491 0.500707 0.384142
 0.277082 0.913817 0.529747 0.464446 0.94098 0.050084 0.761514 0.770205
 0.827817 0.125365 0.0158677 0.688455 0.868247 0.629543 0.736225 0.725412
 0.999458 0.888572 0.233195 0.306322 0.351015 0.513274 0.591114 0.845982
 0.412081 0.841511 0.269317 0.415395

SORTED ARRAY:
7.82637e-06 0.00769819 0.0158677 0.0345721 0.0470446 0.0474645 0.050084
 0.0534616 0.0605643 0.0668422 0.0726859 0.0737491 0.0907329 0.0919649
 0.125365 0.131538 0.166507 0.218959 0.233195 0.237774 0.247039 0.262453
 0.266145 0.269317 0.27271 0.274907 0.277082 0.306322 0.319033 0.328234
 0.351015 0.359265 0.365339 0.383416 0.383502 0.384142 0.412081 0.415395
 0.415999 0.417486 0.436411 0.45865 0.464446 0.477732 0.486517 0.493977
 0.500707 0.504523 0.513274 0.516292 0.519416 0.526929 0.5297 0.529747
 0.532767 0.588977 0.591114 0.629543 0.631635 0.632639 0.651519 0.653919
 0.671149 0.678865 0.679296 0.686773 0.688455 0.701191 0.72266 0.725412
 0.736082 0.736225 0.753356 0.755605 0.75641 0.761514 0.762198 0.766495
 0.770205 0.827817 0.830965 0.841511 0.845982 0.846167 0.868247 0.884707
 0.888572 0.897656 0.904653 0.909208 0.910321 0.913817 0.930436 0.934693
 0.94098 0.947764 0.98255 0.986642 0.991037 0.999458
Program ended with exit code: 0
```

DESIGN AND ANALYSIS OF ALGORITHMS

ITC17

RAUNIT DALAL
2017UIT2537
IT-1

Q3 Implement Dijkstra's Algorithm. Analyze its time complexity.

TIME COMPLEXITY:
1. O(E + V * logV) using Fibonacci Heaps.
2. O(V^2 + E * logV) using Matrix Representation and Priority Queue.
3. O((V + E) * log V) using Adjacency list and Priority Queue. (this one is used below)

SPACE COMPLEXITY: O(V)

where V is the number of vertices of the graph and E is the number of the edges.

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

using namespace std;

class Vertex
{
public:
    int index;
    int distance;
    bool visited;
};
class Comp
{
public:

    bool operator()(const Vertex& v1, const Vertex& v2)
    {
        return v1.distance > v2.distance;
    }

};
void dijkstras(int ** edges, int v, Vertex * input)
{
    int count = 0;
    int j = 0;
    int sum = 0;
    while (count != v - 1)
    {
        priority_queue<Vertex, vector<Vertex>, Comp> queue;
        for (int i = 0; i < v; i++)
        {
            if (edges[i][j])
            {
                if (input[i].visited)
                {
                    continue;
                }
                if (sum + edges[i][j] < input[i].distance)
                {
                    input[i].distance = sum + edges[i][j];
                }
            }
        }
        for (int i = 0; i < v; i++)
        {
            if (!input[i].visited)
            {
                queue.push(input[i]);
            }
            else continue;
        }
        j = queue.top().index;
        input[j].visited = true;
        sum = queue.top().distance;
        count++;
    }
    cout << "SOURCE VERTEX: " << 0 << endl;
    cout << "DISTANCES: \nVERTEX\tDISTANCE" << endl;
    for (int i = 0; i < v; i++)
    {
        cout << "  " << input[i].index << "\t\t  " << input[i].distance <<
         endl;
    }
}

int main()
{
    cout << "ENTER THE NUMBER OF VERTICES AND EDGES: ";
    int v, e;
    cin >> v >> e;
    int ** edges = new int*[v];
    for (int i = 0; i < v; i++)
    {
        edges[i] = new int[v];
        for (int j = 0; j < v; j++)
            edges[i][j] = 0;
    }
    cout << "INPUT GRAPH: " << endl;
    for (int i = 0; i < e; i++)
    {
        int s, d, w;
        cin >> s >> d >> w;
        edges[s][d] = w;
        edges[d][s] = w;
    }
    Vertex * input = new Vertex[v];
    for (int i = 0; i < v; i++)
    {
        input[i].index = i;
        if (i)
        {
            input[i].distance = INT_MAX;
            input[i].visited = false;
        }
        else
        {
            input[i].visited = true;
            input[i].distance = 0;
        }

    }
    dijkstras(edges, v, input);
    return 0;
}
```

```
OUTPUT:


ENTER THE NUMBER OF VERTICES AND EDGES: 7 10
INPUT GRAPH:
0 1 6
0 2 5
0 3 5
2 3 2
1 2 2
4 6 3
5 6 3
3 5 1
2 4 11
2 4 1
SOURCE VERTEX: 0
DISTANCES:
VERTEX    DISTANCE
  0          0
  1          6
  2          5
  3          5
  4          6
  5          6
  6          9
Program ended with exit code: 0
```

Q1 h) Implement Heap Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY:

1. Best Case: O(n * log(n))
2. Worst Case: O(n * log(n))
3. Average Case: O(n * log(n))

SPACE COMPLEXITY: O(1)

```
#include <iostream>
#include <vector>
#include <random>


using namespace std;


void inplaceHeapSort(vector<int> & input)
{
    int n = int(input.size());
    for (int i = 0; i < n; i++)
    {
        int childIndex = i;
        while(childIndex > 0)
        {
            int parentIndex = (childIndex - 1) / 2;
            if(input[childIndex] < input[parentIndex])
                swap(input[childIndex], input[parentIndex]);
            else
                break;
            childIndex = parentIndex;
        }
    }
    int size = n;
    while (size > 1)
    {
        swap(input[0], input[--size]);
        int Pindex = 0;
        int LCI = 2 * Pindex + 1;
        int RCI = 2 * Pindex + 2;
        int minIndex = Pindex;
        while (LCI < size)
        {
            if(input[minIndex] > input[LCI])
                minIndex = LCI;
            if(RCI < size && input[minIndex] > input[RCI])
                minIndex = RCI;
            if(minIndex == Pindex)
                break;
            swap(input[Pindex], input[minIndex]);
            Pindex = minIndex;
            LCI = 2 * Pindex + 1;
            RCI = 2 * Pindex + 2;
        }
    }
}

int main()
{
    vector<int> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = rand() % 750;
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    inplaceHeapSort(A);
    reverse(A.begin(), A.end());
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl;
}
```

```
OUTPUT:

INITIAL STATE OF ARRAY:
307 499 323 158 430 272 294 128 173 709 690 665 492 542 237 503 577 229 340
 112 303 169 459 407 310 183 99 528 566 85 97 326 512 517 560 633 229 649
 579 321 217 422 643 336 485 245 728 91 444 607 251 653 208 694 418 740 624
 446 30 403 472 166 549 524 551 103 477 408 228 433 298 481 385 13 615 64
 563 36 425 419 615 94 129 1 17 695 105 404 201 48 438 623 5 382 252 566
 466 87 188 144

SORTED ARRAY:
1 5 13 17 30 36 48 64 85 87 91 94 97 99 103 105 112 128 129 144 158 166 169
 173 183 188 201 208 217 228 229 229 237 245 251 252 272 294 298 303 307
 310 321 323 326 336 340 382 385 403 404 407 408 418 419 422 425 430 433
 438 444 446 459 466 472 477 481 485 492 499 503 512 517 524 528 542 549
 551 560 563 566 566 577 579 607 615 615 623 624 633 643 649 653 665 690
 694 695 709 728 740
Program ended with exit code: 0
```

Q6. Implement Huffman Coding. Analyze its time complexity.

TIME COMPLEXITY: O(L + N * log(N))

SPACE COMPLEXITY: O(N)

where L is the length of the input file and N is the number of the unique characters.

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>


using namespace std;



class minHeapNode
{
public:
    char data;
    int freq;
    char representative;
    minHeapNode * left;
    minHeapNode * right;
    minHeapNode(char data, int freq) : data(data), freq(freq),
     representative(data), left(NULL), right(NULL) {}
};

class myComparator
{
public:
    int operator() (const minHeapNode * p1, const minHeapNode * p2)
    {
        if(p1 -> freq == p2 -> freq)
            return p1 -> representative >= p2 -> representative; // ASCII
             cmoparison
        else return p1 -> freq > p2 -> freq;
    }
};


minHeapNode * buildHuffmanTree(map<char, int> & file)
{
    priority_queue<minHeapNode* , vector<minHeapNode*>, myComparator>
     minHeap;
    map<char, int> :: iterator it;
    for (auto it : file)
    {
        minHeap.push(new minHeapNode(it.first, it.second));
    }

    while (minHeap.size() != 1)
    {
        minHeapNode * first = minHeap.top();
        minHeap.pop();
        minHeapNode * second = minHeap.top();
        minHeap.pop();
        minHeapNode * N1 = new minHeapNode('$', first -> freq + second ->
         freq);
        N1 -> left = first;
        N1 -> right = second;
        N1 -> representative = min({N1 -> left -> representative, N1 ->
         right -> representative});
        minHeap.push(N1);
    }

    return minHeap.top();
}

void getCodes(minHeapNode * root, string code, map<char, string> & codes)
{
    if(!root)
    {
        return;
    }
    if (root -> data != '$')
    {
        codes.insert({root -> data, code});
    }
    getCodes(root -> left, code + '0', codes);
    getCodes(root -> right, code + '1', codes);
}

string decode_file(minHeapNode * root, string code)
{
    string ans = "";
    minHeapNode * curr = root;
    for (int i = 0; i < code.size(); i++)
    {
        if (code[i] == '0')
        {
            curr = curr -> left;
        }
        else curr = curr -> right;
        if (!curr -> left && !curr -> right)
        {
            ans += curr -> data;
            curr = root;
        }
    }
    return ans + '\0';
}

void levelOrderTraversal(minHeapNode * root)
```

```
{
    queue<minHeapNode*> q;
    q.push(root);
    q.push(NULL);
    while (1)
    {
        minHeapNode * top = q.front();
        q.pop();
        if(!top)
        {
            cout << endl;
            if(q.empty())
                break;
            q.push(NULL);
        }
        else
        {
            cout << top -> data << " : " << top -> representative << "\t";
            if(top -> left)
                q.push(top -> left);
            if(top -> right)
                q.push(top -> right);
        }
    }
}

int main()
{
    string text = "I love programming!";
    map<char, int> file;
    for (int i = 0; i < text.size(); i++)
    {
        if (file.count(text[i]) == 0)
        {
            file[text[i]] = 1;
        }
        else file[text[i]]++;
    }
    for(auto it : file)
    {
        cout << it.first << ": " << it.second << " " << int(it.first) <<
         endl;
    }
    minHeapNode * root = buildHuffmanTree(file);
    map<char, string> codes;
    getCodes(root, "", codes);

    for(auto it : codes)
    {
        cout << it.first << ": " << it.second << "  " << (int)it.first <<
         endl ;
    }
    string encoded = "";
    for (int i = 0; i < text.size(); i++)
    {
        encoded += codes.at(text[i]);
    }
    float size = float(encoded.size());

    cout << "Average Length: " << round (1.0 * size * 100/ text.size())/100
     << " bits/symbol" << endl;
    cout << "Encoded file : " << encoded << endl;
    cout << "File Decoded : " << decode_file(root, encoded) << endl;
}
```

```
OUTPUT:

Input Text: I love programming!

 : 2 32
!: 1 33
I: 1 73
a: 1 97
e: 1 101
g: 2 103
i: 1 105
l: 1 108
m: 2 109
n: 1 110
o: 2 111
p: 1 112
r: 2 114
v: 1 118
 : 1101  32
!: 11100  33
I: 11101  73
a: 11110  97
e: 11111  101
g: 000  103
i: 0010  105
l: 0011  108
m: 010  109
n: 0110  110
o: 100  111
p: 0111  112
r: 101  114
v: 1100  118
Average Length: 3.79 bits/symbol
Encoded file :
 111011101001110011001111111010111101100000101111100100100010011000011100
File Decoded : I love programming!
Program ended with exit code: 0
```

Q9. a) Implement KMP Algorithm. Analyze its time complexity.

TIME COMPLEXITY: O(m + n)

SPACE COMPLEXITY: O(n ^ 2)

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>

using namespace std;

void computeLPSArray(string & pat, int M, vector<int> & lps);


void KMPSearch(string & pat, string & txt)
{
    int M = int(pat.size());
    int N = int(txt.size());
    vector<int> lps(M);
    computeLPSArray(pat, M, lps);
    int i = 0, j = 0;
    while (i < N)
    {
        if (pat[j] == txt[i])
        {
            j++;
            i++;
        }
        if (j == M)
        {
            cout << "Found pattern at index " <<  i - j << endl;
            j = lps[j - 1];
        }
        else if (i < N && pat[j] != txt[i])
        {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
}

void computeLPSArray(string & pat, int M, vector<int> & lps)
{
    int len = 0;
    lps[0] = 0;
    int i = 1;
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if (len)
                len = lps[len - 1];
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}


int main()
{
    string txt = "ABABDABACDABABCABAB";
    string pat = "ABABCABAB";
    cout << "TEXT: " << txt << endl;
    cout << "PATTERN: " << txt << endl;
    KMPSearch(pat, txt);
    return 0;
}
```

```
OUTPUT:

TEXT: ABABDABACDABABCABAB
PATTERN: ABABDABACDABABCABAB
Found pattern at index 10
Program ended with exit code: 0
```

Q3 Implement Longest Common Subsequence. Analyze its time complexity.

TIME COMPLEXITY: O(m * n)

SPACE COMPLEXITY: O(m * n)

where m and n are the length of the input strings.

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <chrono>

using namespace std;


int lcsDP(string & s1, string & s2)
{
    int m = int(s1.size()), n = int(s2.size());
    int ** dp = new int*[m + 1];
    for(int i = 0; i < m + 1; i++)
        dp[i] = new int[n + 1];
    for(int i = 0; i < m + 1; i++)
        dp[i][0] = 0;
    for(int i = 0; i < n + 1; i++)
        dp[0][i] = 0;
    for(int i = 1; i < m + 1; i++)
    {
        for(int j = 1; j < n + 1; j++)
        {
            if(s1[m - i] == s2[n - j])
                dp[i][j] = 1 + dp[i - 1][j - 1];
            else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    int i = m, j = n;
    string lcs = "";
    while(i > 0 && j > 0)
    {
        if(s1[m - i] == s2[n - j])
        {
            lcs.push_back(s1[m - i]);
            i--;
            j--;
        }
        else if(dp[i - 1][j] > dp[i][j - 1])
            i--;
        else j--;
    }
    cout << "LONGEST COMMON SUBSEQUENCE: " << lcs << endl;
    cout << "LENGTH OF THE LCS: " << lcs.size() << endl;
    return dp[m][n];
}
```

```
int main()
{
    string s1 =
     "nohgdazaxavkhylqvghqpifijohudenozotejoxavkfkzcdqnoxydynavwdyl";
    string s2 = "nohgdazargdcbmbgjcvqpcbadujkxaxujudmbejcrevuvcdobo";
    cout << "S1: " << s1 << endl;
    cout << "S2: " << s2 << endl;
    lcsDP(s1, s2);
}
```

OUTPUT:

```
S1: nohgdazaxavkhylqvghqpifijohudenozotejoxavkfkzcdqnoxydynavwdyl
S2: nohgdazargdcbmbgjcvqpcbadujkxaxujudmbejcrevuvcdobo
LONGEST COMMON SUBSEQUENCE: nohgdazavqpjudeevcdo
LENGTH OF THE LCS: 20
Program ended with exit code: 0
```

Q3 Implement Matrix Multiplication. Analyze their time complexities.

TIME COMPLEXITY: O(a * b * c)

SPACE COMPLEXITY: O(a x b x c)

where (a x b) and (b x c) are the dimensions.

```
#include <iostream>
#include <vector>
#include <random>
#include <chrono>

using namespace std;


vector<vector<int>> multiply(vector<vector<int>> & matrix1,
 vector<vector<int>> & matrix2)
{
    int a = int(matrix1.size()), b = int(matrix1[0].size()), c =
     int(matrix2.size()), d = int(matrix2[0].size());
    if(b != c)
        return {};
    vector<vector<int>> result(a, vector<int>(d, 0));
    for(int i = 0; i < a; ++i)
        for(int k = 0; k < d; ++k)
            for(int j = 0; j < b; ++j)
                result[i][k] += matrix1[i][j] * matrix2[j][k];
    return result;
}

int main()
{
    vector<vector<int>> matrix1 = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    cout << "MATRIX 1: " << endl;
    for (auto & V : matrix1)
    {
        for(int & v : V)
            cout << v << " ";
        cout << endl;
    }
    vector<vector<int>> matrix2 = {
        {1, 0, 0},
        {0, 1, 0},
        {0, 0, 1}
    };
    cout << "MATRIX 2: " << endl;
    for (auto & V : matrix2)
    {
        for(int & v : V)
```

```
            cout << v << " ";
        cout << endl;
    }
    vector<vector<int>> result = multiply(matrix1, matrix2);
    cout << "MATRIX1 * MATRIX2: \n";
    for (auto & V : result)
    {
        for(int & v : V)
            cout << v << " ";
        cout << endl;
    }
}
```

```
OUTPUT:

MATRIX 1:
1 2 3
4 5 6
7 8 9
MATRIX 2:
1 0 0
0 1 0
0 0 1
MATRIX1 * MATRIX2:
1 2 3
4 5 6
7 8 9
Program ended with exit code: 0
```

Q1 a) Implement Merge Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY:

1. Best Case: O(n * log(n))
2. Worst Case: O(n * log(n))
3. Average Case: O(n * log(n))

SPACE COMPLEXITY: O(n)

```
#include <iostream>
#include <vector>
#include <random>


using namespace std;


void merge(vector<int> & A, int start, int end)
{
    vector<int> placeholder(end - start + 1);
    int mid = start + (end - start) / 2, i = start, j = mid + 1, index = 0;
    while(i <= mid and j <= end)
    {
        if(A[i] < A[j])
            placeholder[index++] = A[i++];
        else placeholder[index++] = A[j++];
    }
    while(i <= mid)
        placeholder[index++] = A[i++];

    while(j <= end)
        placeholder[index++] = A[j++];

    for(int i = start; i <= end; ++i)
        A[i] = placeholder[i - start];

}

void mergeSort(vector<int> & A, int start, int end)
{
    if (start >= end)
        return;
    int mid = start + (end - start) / 2;
    mergeSort(A, start, mid);
    mergeSort(A, mid + 1, end);
    merge(A, start, end);

}
int main()
{
    vector<int> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = rand() % 500;
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    mergeSort(A, 0, int(A.size()) - 1);
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl;
}
```

```
OUTPUT:

INITIAL STATE OF ARRAY:
307 249 73 158 430 272 44 378 423 209 440 165 492 42 487 3 327 229 340 112
 303 169 209 157 60 433 99 278 316 335 97 326 12 267 310 133 479 149 79 321
 467 172 393 336 485 245 228 91 194 357 1 153 208 444 168 490 124 196 30
 403 222 166 49 24 301 353 477 408 228 433 298 481 135 13 365 314 63 36 425
 169 115 94 129 1 17 195 105 404 451 298 188 123 5 382 252 66 216 337 438
 144

SORTED ARRAY:
1 1 3 5 12 13 17 24 30 36 42 44 49 60 63 66 73 79 91 94 97 99 105 112 115
 123 124 129 133 135 144 149 153 157 158 165 166 168 169 169 172 188 194
 195 196 208 209 209 216 222 228 228 229 245 249 252 267 272 278 298 298
 301 303 307 310 314 316 321 326 327 335 336 337 340 353 357 365 378 382
 393 403 404 408 423 425 430 433 433 438 440 444 451 467 477 479 481 485
 487 490 492
Program ended with exit code: 0
```

Q9. b) Implement Naive Searching Algorithm. Analyze its time complexity.

TIME COMPLEXITY: O(m * n)

SPACE COMPLEXITY: O(1)

```cpp
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>

using namespace std;

void search(string & pat, string & txt)
{
    int M = int(pat.size());
    int N = int(txt.size());
    for (int i = 0; i <= N - M; i++)
    {
        int j;
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;
        if (j == M)
            cout << "Pattern found at index: " << i << endl;
    }
}


int main()
{
    string txt = "ABABDABACDABABCABAB";
    string pat = "ABABCABAB";
    cout << "TEXT: " << txt << endl;
    cout << "PATTERN: " << txt << endl;
    search(pat, txt);
    return 0;
}


//OUTPUT:
//
//TEXT: ABABDABACDABABCABAB
//PATTERN: ABABDABACDABABCABAB
//Pattern found at index: 10
//Program ended with exit code: 0
```

Q5. Implement Optimal Binary Search Tree. Analyze its time complexity.

TIME COMPLEXITY: O(n ^ 3)

SPACE COMPLEXITY: O(n ^ 2)

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>

using namespace std;

vector<int> getPreSum(vector<int> & frequencies)
{
    vector<int> sum(frequencies.size());
    sum[0] = frequencies[0];
    for(int i = 1; i < frequencies.size(); ++i)
        sum[i] = sum[i - 1] + frequencies[i];
    return sum;
}

int sumInterval(vector<int> & sum, int i, int j)
{
    return sum[j] - ((i == 0) ? 0 : sum[i - 1]);
}

int optimalSearchTree(vector<int> & keys, vector<int> & freq)
{
    vector<int> sum = getPreSum(freq);
    int n = int(keys.size());
    int cost[n][n];
    for (int i = 0; i < n; i++)
        cost[i][i] = freq[i];
    for (int L = 2; L <= n; L++)
    {
        for (int i = 0; i <= n - L + 1; i++)
        {
            int j = i + L - 1;
            cost[i][j] = INT_MAX;
            for (int r = i; r <= j; r++)
            {
                int c = ((r > i) ? cost[i][r - 1] : 0) + ((r < j) ? cost[r
                 + 1][j] : 0) + sumInterval(sum, i, j);
                cost[i][j] = min(c, cost[i][j]);
            }
        }
    }
    return cost[0][n-1];
}
```

```
int main()
{
    vector<int> keys = {10, 12, 20, 24, 30};
    vector<int> frequencies = {34, 8, 50, 12, 9};
    cout << "KEYS: ";
    for(int & v : keys)
        cout << v << " ";
    cout << endl;
    cout << "FREQUENCIES: ";
    for(int & v : keys)
        cout << v << " ";
    cout << endl;
    cout << "Cost of Optimal BST: " << optimalSearchTree(keys, frequencies)
     << endl;
    return 0;
}


//OUTPUT:
//
//KEYS: 10 12 20 24 30
//FREQUENCIES: 10 12 20 24 30
//Cost of Optimal BST: 193
//
//Program ended with exit code: 0
```

Q1 b) Implement Quick Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY:

1. Best Case: O(n * log(n))
2. Worst Case: O(n ^ 2)
3. Average Case: O(n * log(n))

SPACE COMPLEXITY: O(1)

```
#include <iostream>
#include <vector>
#include <random>


using namespace std;


int partition(vector<int> & arr, int start, int end)
{
    int pivot = arr[end];
    int smallIndex = start;
    for(int i = start; i < end; i++)
        if(arr[i] <= pivot)
            swap(arr[smallIndex++], arr[i]);
    swap(arr[smallIndex], arr[end]);
    return smallIndex;
}

void quickSort(vector<int> & input, int start, int end)
{
    if(start >= end)
        return;
    int p = partition(input, start, end);
    quickSort(input, start, p - 1);
    quickSort(input, p + 1, end);
}

int main()
{
    vector<int> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = rand() % 500;
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    quickSort(A, 0, int(A.size()) - 1);
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl;
}
```

```
OUTPUT:

INITIAL STATE OF ARRAY:
307 249 73 158 430 272 44 378 423 209 440 165 492 42 487 3 327 229 340 112
 303 169 209 157 60 433 99 278 316 335 97 326 12 267 310 133 479 149 79 321
 467 172 393 336 485 245 228 91 194 357 1 153 208 444 168 490 124 196 30
 403 222 166 49 24 301 353 477 408 228 433 298 481 135 13 365 314 63 36 425
 169 115 94 129 1 17 195 105 404 451 298 188 123 5 382 252 66 216 337 438
 144

SORTED ARRAY:
1 1 3 5 12 13 17 24 30 36 42 44 49 60 63 66 73 79 91 94 97 99 105 112 115
 123 124 129 133 135 144 149 153 157 158 165 166 168 169 169 172 188 194
 195 196 208 209 209 216 222 228 228 229 245 249 252 267 272 278 298 298
 301 303 307 310 314 316 321 326 327 335 336 337 340 353 357 365 378 382
 393 403 404 408 423 425 430 433 433 438 440 444 451 467 477 479 481 485
 487 490 492
Program ended with exit code: 0
```

Q1 e) Implement Radix Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY: O(d * (n + b)) where b is base, d = log(max) / log(b)

SPACE COMPLEXITY: O(1)

```
#include <iostream>
#include <vector>
#include <random>


using namespace std;


#include<iostream>
using namespace std;

int getMax(vector<int> & arr, int n)
{
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

void countSort(vector<int> & arr, int n, int exp)
{
    int output[n];
    int i, count[10] = {0};

    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    for (i = n - 1; i >= 0; i--)
    {
        output[count[(arr[i] / exp) % 10 ] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// Radix Sort
void radixSort(vector<int> & A)
{
    int n = int(A.size());
    int m = *max_element(A.begin(), A.end());
    for (int exp = 1; m / exp > 0; exp *= 10)
        countSort(A, n, exp);
}


int main()
{
    vector<int> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = rand() % 750;
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    radixSort(A);
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl;
}
```

```
OUTPUT:

INITIAL STATE OF ARRAY:
307 499 323 158 430 272 294 128 173 709 690 665 492 542 237 503 577 229 340
 112 303 169 459 407 310 183 99 528 566 85 97 326 512 517 560 633 229 649
 579 321 217 422 643 336 485 245 728 91 444 607 251 653 208 694 418 740 624
 446 30 403 472 166 549 524 551 103 477 408 228 433 298 481 385 13 615 64
 563 36 425 419 615 94 129 1 17 695 105 404 201 48 438 623 5 382 252 566
 466 87 188 144

SORTED ARRAY:
1 5 13 17 30 36 48 64 85 87 91 94 97 99 103 105 112 128 129 144 158 166 169
 173 183 188 201 208 217 228 229 229 237 245 251 252 272 294 298 303 307
 310 321 323 326 336 340 382 385 403 404 407 408 418 419 422 425 430 433
 438 444 446 459 466 472 477 481 485 492 499 503 512 517 524 528 542 549
 551 560 563 566 566 577 579 607 615 615 623 624 633 643 649 653 665 690
 694 695 709 728 740
Program ended with exit code: 0
```

Q9. c) Implement Rabin Karp Algorithm. Analyze its time complexity.

TIME COMPLEXITY:
1. AVERAGE CASE: O(m + n)
2. BEST CASE: O(m + n)
3. WORST CASE: O(m * n)

SPACE COMPLEXITY: O(1)

where m is the length of pattern and n is the length of text.

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>
#define d 256


using namespace std;


void search(string & pat, string & txt, int q)
{
    int M = int(pat.size());
    int N = int(txt.size());
    int i, j;
    int p = 0;
    int t = 0;
    int h = 1;
    for (i = 0; i < M - 1; i++)
        h = (h * d) % q;
    for (i = 0; i < M; i++)
    {
        p = (d * p + pat[i]) % q;
        t = (d * t + txt[i]) % q;
    }
    for (i = 0; i <= N - M; i++)
    {
        if (p == t)
        {
            for (j = 0; j < M; j++)
                if(txt[i + j] != pat[j])
                    break;
            if (j == M)
                cout << "Pattern found at index: " << i << endl;
        }
        if(i < N - M)
        {
            t = (d * (t - txt[i] * h) + txt[i + M]) % q;
            if (t < 0)
                t = (t + q);
        }
    }
}


int main()
{
    string txt = "GEEKS FOR GEEKS";
    string pat = "GEEK";
    cout << "TEXT: " << txt << endl;
    cout << "PATTERN: " << txt << endl;
    int q = 37;
    search(pat, txt, q);
    return 0;
}
```

OUTPUT:

```
TEXT: GEEKS FOR GEEKS
PATTERN: GEEKS FOR GEEKS
Pattern found at index: 0
Pattern found at index: 10
Program ended with exit code: 0
```

Q1 g) Implement Selection Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY:

1. Best Case: O(n^2)
2. Worst Case: O(n^2)
3. Average Case: O(n^2)

SPACE COMPLEXITY: O(1)

```
#include <iostream>
#include <vector>
#include <random>


using namespace std;


int minIndex(vector<int> & a, int i, int j)
{
    if (i == j)
        return i;
    int k = minIndex(a, i + 1, j);
    return (a[i] < a[k]) ? i : k;
}

void recurSelectionSort(vector<int> & a, int n, int index = 0)
{
    if (index == n)
        return;
    int k = minIndex(a, index, n-1);
    if (k != index)
        swap(a[k], a[index]);
    recurSelectionSort(a, n, index + 1);
}

int main()
{
    vector<int> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = rand() % 750;
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    recurSelectionSort(A, int(A.size()));
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl;
}
```

```
OUTPUT:

INITIAL STATE OF ARRAY:
307 499 323 158 430 272 294 128 173 709 690 665 492 542 237 503 577 229 340
 112 303 169 459 407 310 183 99 528 566 85 97 326 512 517 560 633 229 649
 579 321 217 422 643 336 485 245 728 91 444 607 251 653 208 694 418 740 624
 446 30 403 472 166 549 524 551 103 477 408 228 433 298 481 385 13 615 64
 563 36 425 419 615 94 129 1 17 695 105 404 201 48 438 623 5 382 252 566
 466 87 188 144

SORTED ARRAY:
1 5 13 17 30 36 48 64 85 87 91 94 97 99 103 105 112 128 129 144 158 166 169
 173 183 188 201 208 217 228 229 229 237 245 251 252 272 294 298 303 307
 310 321 323 326 336 340 382 385 403 404 407 408 418 419 422 425 430 433
 438 444 446 459 466 472 477 481 485 492 499 503 512 517 524 528 542 549
 551 560 563 566 566 577 579 607 615 615 623 624 633 643 649 653 665 690
 694 695 709 728 740
Program ended with exit code: 0
```

Q1 f) Implement Shell Sort Algorithm. Analyze its time complexity.

TIME COMPLEXITY:
1. BEST CASE: O(n ^ 2)

SPACE COMPLEXITY: O(1)

```cpp
#include <iostream>
#include <vector>
#include <random>
using namespace std;
void shellSort(vector<int> & arr)
{
    int n = int(arr.size());
    for (int gap = n / 2; gap > 0; gap /= 2)
        for (int i = gap; i < n; i += 1)
        {
            int temp = arr[i], j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];
            arr[j] = temp;
        }
}
int main()
{
    vector<int> A(100);
    for(int i = 0; i < 100; ++i)
        A[i] = rand() % 100;
    cout << "INITIAL STATE OF ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl << endl;
    shellSort(A);
    cout << "SORTED ARRAY:\n";
    for(int i = 0; i < 100; ++i)
        cout << A[i] << " ";
    cout << endl;
}
```

OUTPUT:

```
INITIAL STATE OF ARRAY:
7 49 73 58 30 72 44 78 23 9 40 65 92 42 87 3 27 29 40 12 3 69 9 57 60 33 99
 78 16 35 97 26 12 67 10 33 79 49 79 21 67 72 93 36 85 45 28 91 94 57 1 53
 8 44 68 90 24 96 30 3 22 66 49 24 1 53 77 8 28 33 98 81 35 13 65 14 63 36
 25 69 15 94 29 1 17 95 5 4 51 98 88 23 5 82 52 66 16 37 38 44

SORTED ARRAY:
1 1 1 3 3 3 4 5 5 7 8 8 9 9 10 12 12 13 14 15 16 16 17 21 22 23 23 24 24 25
 26 27 28 28 29 29 30 30 33 33 33 35 35 36 36 37 38 40 40 42 44 44 44 45 49
 49 49 51 52 53 53 57 57 58 60 63 65 65 66 66 67 67 68 69 69 72 72 73 77 78
 78 79 79 81 82 85 87 88 90 91 92 93 94 94 95 96 97 98 98 99
Program ended with exit code: 0
```