Design & Analysis of Algorithm LAB

PAPER CODE	**:**	**CIC-359**
Faculty Name	:	Dr. Moolchand Sharma
Name	:	Amit Singhal
Enrollment No.	:	11614802722
Branch	:	Computer Science & Engg.
Semester | Group	:	5C6

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

PSP Area, Plot No. 1, Sector-22, Rohini, Delhi-110086

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

Computer Science & Engineering Department

VISION

"To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values."

MISSION

To foster an open, multidisciplinary and highly collaborative research environment for producing world-class engineers capable of providing innovative solutions to real-life problems and fulfil societal needs.

LAB Assessment Sheet

S.No.	Experiment Name	M	A	R	K	S	Total Marks	Date of Perf.	Date of Check.	Signature
		R1	R2	R3	R4	R5				

S.No.	Experiment Name	M A R K S					Total Marks	Date of Perf.	Date of Check.	Signature
		R1	R2	R3	R4	R5				

INTRODUCTION TO DESIGN AND ANALYSIS OF ALGORITHMS

Design and Analysis of Algorithms is a fundamental aspect of computer science that involves creating efficient solutions to computational problems and evaluating their performance. It focuses on designing algorithms that effectively address specific challenges and analysing their efficiently in terms of **time** and **space complexity**.

ALGORITHM ANALYSIS: -

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Why the Analysis of Algorithm is important?

- To predict the behavior of an algorithm without implementing it on a specific computer.
- It is much more convenient to have simple measures for the efficiency of an algorithm than to implement the algorithm and test the efficiency every time a certain parameter in the underlying computer system changes.
- It is impossible to predict the exact behavior of an algorithm. There are too many influencing factors.
- The analysis is thus only an approximation; it is not perfect.
- More importantly, by analyzing different algorithms, we can compare them to determine the best one for our purpose.

What are the Advantages of Analysis of Algorithms?

1. Efficiency Assessment: Analyzing algorithms helps determine their time and space complexity, allowing developers to evaluate how efficiently they will perform under different conditions.

2. Optimal Algorithm Selection: It provides insights that guide the selection of the most appropriate algorithm for a given problem, ensuring the best performance based on specific requirements.

3. Scalability Evaluation: Analysis reveals how an algorithm's performance changes with varying input sizes, helping to predict scalability and identify potential issues as data grows.

4. Resource Management: Understanding an algorithm's resource requirements enables better allocation of computational resources, leading to improved overall system performance.

5. Bottleneck Identification: Through analysis, developers can identify inefficient parts of an algorithm, facilitating targeted optimization efforts to enhance performance.

6. Performance Prediction: Analyzing algorithms allows for performance predictions in different scenarios, aiding in planning and system design.

7. Understanding Trade-offs: It helps clarify the trade-offs between time and space complexity, which is crucial for making informed decisions in algorithm design.

8. Facilitating Debugging: A clear understanding of how an algorithm should perform helps identify bugs or inefficiencies more easily during development.

9. Informed Learning and Improvement: Studying algorithm analysis principles fosters a deeper understanding of algorithm design techniques, promoting continuous learning and improvement.

10. Foundation for Advanced Concepts: It provides a basis for exploring more advanced topics in computer science, such as complexity theory and algorithmic optimization.

What are the Disadvantages of Analysis of Algorithms?

1. Complexity: The mathematical and theoretical nature of algorithm analysis can be complex and difficult to understand for beginners.

2. Assumptions: Analysis often relies on assumptions about input data and computational models, which may not reflect real-world scenarios.

3. Overhead: Detailed analysis can introduce additional time and resource overhead during the design and development process.

4. Limited Practicality: Theoretical performance metrics may not always translate to practical performance due to factors like hardware differences and implementation variations.

5. Neglect of Other Factors: Focusing solely on time and space complexity can lead to overlooking other important factors, such as maintainability, readability, and usability.

6. Dynamic Environments: Algorithms may perform differently in dynamic environments, making pre-analysis less reliable for certain applications.

7. Diminishing Returns: For some algorithms, further analysis may yield diminishing returns in terms of insights gained versus effort spent.

Applications of Design and Analysis of Algorithms: -

1. Search Engines: Algorithms are used to index and retrieve information efficiently, optimizing search results based on user queries.

2. Data Compression: Algorithms help reduce file sizes for storage and transmission, making data transfer faster and more efficient.

3. Machine Learning: Used to develop algorithms for training models, optimizing performance, and making predictions based on data patterns.

4. Network Routing: Algorithms determine the most efficient paths for data transmission across networks, improving communication speeds and reliability.

5. Scheduling: Employed in operations research to optimize scheduling tasks in manufacturing, transportation, and project management for better resource utilization.

Lab Exercise - 1

- AIM :: WAP in C++ to implement Bubble, Merge, Quick & Insertion Sort and also evaluate time in each.

1. Bubble Sort

Source_Code ::

```
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

void bubbleSort(vector<int>& arr)
{
    int n = arr.size();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

int main()
{
    cout << "5C6 - Amit Singhal (11614802722)" << endl;
```

```
    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
    cout << "Unsorted Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    clock_t start = clock();
    bubbleSort(arr);
    clock_t end = clock();

    cout << "Bubble Sort:" << endl;
    cout << "Sorted Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    double time_taken_ms = double(end - start) * 1000.0
        / CLOCKS_PER_SEC; // Convert to milliseconds
    cout << "Time taken: " << time_taken_ms << " milliseconds" << endl;

    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads$ vi exp_1.1.cpp
amit@Toshiba-Satellite-C850:~/Downloads$ g++ exp_1.1.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Unsorted Array: 64 34 25 12 22 11 90
Bubble Sort:
Sorted Array: 11 12 22 25 34 64 90
Time taken: 0.003 milliseconds
```

2. Merge Sort

Source_Code ::

```
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

void merge(vector<int>& arr, int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[m + 1 + i];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
```

```
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(vector<int>& arr, int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main()
{
    cout << "5C6 - Amit Singhal (11614802722)" << endl;
    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
    cout << "Unsorted Array: ";
```

```
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    clock_t start = clock();
    mergeSort(arr, 0, arr.size() - 1);
    clock_t end = clock();

    cout << "Merge Sort:" << endl;
    cout << "Sorted Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    double time_taken_ms = double(end - start) * 1000.0
        / CLOCKS_PER_SEC; // Convert to milliseconds
    cout << "Time taken: " << time_taken_ms << " milliseconds" << endl;

    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads$ g++ exp_1.2.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Unsorted Array: 64 34 25 12 22 11 90
Merge Sort:
Sorted Array: 11 12 22 25 34 64 90
Time taken: 0.239 milliseconds
```

3. Quick Sort

Source_Code ::

```
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

int partition(vector<int>& arr, int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

void quickSort(vector<int>& arr, int low, int high)
{
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```
int main()
{
    cout << "5C6 - Amit Singhal (11614802722)" << endl;
    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
    cout << "Unsorted Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;
    clock_t start = clock();
    quickSort(arr, 0, arr.size() - 1);
    clock_t end = clock();

    cout << "Quick Sort:" << endl;
    cout << "Sorted Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    double time_taken_ms = double(end - start) * 1000.0
        / CLOCKS_PER_SEC; // Convert to milliseconds
    cout << "Time taken: " << time_taken_ms << " milliseconds" << endl;

    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads$ g++ exp_1.3.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Unsorted Array: 64 34 25 12 22 11 90
Quick Sort:
Sorted Array: 11 12 22 25 34 64 90
Time taken: 0.026 milliseconds
```

4. Insertion Sort

Source_Code ::

```
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

void insertionSort(vector<int>& arr)
{
    int n = arr.size();
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main()
{
    cout << "5C6 - Amit Singhal (11614802722)" << endl;
    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
```

```
    cout << "Unsorted Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    clock_t start = clock();
    insertionSort(arr);
    clock_t end = clock();

    cout << "Insertion Sort:" << endl;
    cout << "Sorted Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    double time_taken_ms = double(end - start) * 1000.0
        / CLOCKS_PER_SEC; // Convert to milliseconds
    cout << "Time taken: " << time_taken_ms << " milliseconds" << endl;

    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads$ g++ exp_1.4.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Unsorted Array: 64 34 25 12 22 11 90
Insertion Sort:
Sorted Array: 11 12 22 25 34 64 90
Time taken: 0.003 milliseconds
```

20/8/24

VIVA QUESTIONS

Q-1) Which is the best sorting Algorithm?

Ans-1) An Algorithm will be efficient if it has linear time complexity $O(n)$ & atmost logarithmic space complexity $O(\log n)$

	Best Case		Worst Case	
	Time	Space	Time	Space
Bubble Sort	$O(n^2)$	$O(1)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(1)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n)$	$O(n \log n)$	$O(n)$
Quick Sort	$O(n \log n)$	$O(\log n)$	$O(n^2)$	$O(n)$

Q-2) Compare Bubble, Merge, Quick & Insertion sort?

Ans-2) ① Bubble Sort

It is a simple sorting algorithm that repeatedly steps through the input list of elements, comparing the current element with one after the another.

Time complexity ⇒ $O(n^2)$ } Worst case
Space complexity ⇒ $O(1)$

② Merge Sort

It is a sorting algorithm based on divide & conquer technique. It is a comparison based algorithm. We divide the bigger problem into smaller problems & then accumulate the soln of smaller tasks to obtain soln of bigger problem.

Time complexity ⇒ $O(n \log n)$ } Worst case
space complexity ⇒ $O(n)$

Lab Exercise - 2

- AIM :: WAP in C++ to implement Linear & Binary Search and also evaluate time in each.

1. Linear Search

Source_Code ::

```
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

int linearSearch(const vector<int>& arr, int x)
{
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1; // Element not found
}

int main()
{
    cout << "5C6 - Amit Singhal (11614802722)" << endl;

    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
```

```
    cout << "Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;

    int x;
    cout << "Enter the element to search: ";
    cin >> x;
    clock_t start = clock();
    int index = linearSearch(arr, x);
    clock_t end = clock();
    if (index != -1) {
        cout << "Element found at index: " << index << endl;
    } else {
        cout << "Element not found" << endl;
    }
    double time_taken_ms = double(end - start) * 1000.0
        / CLOCKS_PER_SEC; // Convert to milliseconds
    cout << "Time taken: " << time_taken_ms << " milliseconds" << endl;
    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads$ g++ exp_2.1.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Array: 64 34 25 12 22 11 90
Enter the element to search: 25
Element found at index: 2
Time taken: 0.003 milliseconds
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Array: 64 34 25 12 22 11 90
Enter the element to search: 90
Element found at index: 6
Time taken: 0.026 milliseconds
```

2. Binary Search

Source_Code ::

```
#include <algorithm>
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

int binarySearch(const vector<int>& arr, int x)
{
    int left = 0, right = arr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x) {
            return mid;
        }
        if (arr[mid] < x) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Element not found
}

int main()
{
    cout << "5C6 - Amit Singhal (11614802722)" << endl;

    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
```

```
    sort(arr.begin(), arr.end()); // Binary search requires a sorted array
    cout << "Array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;
    int x;
    cout << "Enter the element to search: ";
    cin >> x;
    clock_t start = clock();
    int index = binarySearch(arr, x);
    clock_t end = clock();
    if (index != -1) {
        cout << "Element found at index: " << index << endl;
    } else {
        cout << "Element not found" << endl;
    }
    double time_taken_ms = double(end - start) * 1000.0
        / CLOCKS_PER_SEC; // Convert to milliseconds
    cout << "Time taken: " << time_taken_ms << " milliseconds" << endl;
    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads$ g++ exp_2.2.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Array: 11 12 22 25 34 64 90
Enter the element to search: 25
Element found at index: 3
Time taken: 0.025 milliseconds
amit@Toshiba-Satellite-C850:~/Downloads$ ./a
5C6 - Amit Singhal (11614802722)
Array: 11 12 22 25 34 64 90
Enter the element to search: 90
Element found at index: 6
Time taken: 0.003 milliseconds
```

VIVA QUESTIONS

Q-1) Write down 3 applications of Binary Search.

Ans-1) (i) Binary search is effective when the data set is sorted.

(ii) Binary search are useful to find nearest value to given target.

(iii) Binary search are used for spell checkers & auto-correct.

Q-2) write down 3 applications of Linear Search.

Ans-2) (i) Linear search is efficient for small data sets.

(ii) Linear search is efficient when the data set is unsorted.

(iii) Linear search is efficient for simple data processing

Q-3) Which is better: Linear or Binary search?

Ans-3) In majority of cases, Linear search lags behind Binary search as in binary search, we search for element, if it is more than mid, then we eliminate the left half, else, eliminate right half.

Time complexities

Linear search $\longrightarrow O(n)$ $[T(n) = T(n-1) + c]$

Binary search $\longrightarrow O(\log n)$ $[T(n) = T(n/2) + c]$

Thus, Binary Search is better than Linear Search

③ Quick Sort

It is a sorting technique based on the divide & conquer technique. It is implemented using D & C where we divide the bigger problem into smaller problems, cummulate the soln of smaller problem to find soln of bigger problems.

(i) Unsorted Array

Time complexity ⇒ O(nlogn)
Space complexity ⇒ O(logn) } Best case

(ii) Sorted Array

Time complexity ⇒ O(n²)
Space complexity ⇒ O(n) } Worst case

---

④ Insertion Sort

It is a sorting algorithm where we pick element from unsorted part of array & move to the sorted part.

Time complexity ⇒ O(n²)
Space complexity ⇒ O(1)

MM
10/9/24

Lab Exercise - 3

- AIM :: WAP in C++ to implement Huffman Coding & also evaluate its time complexity.

Source_Code ::

```
#include <ctime>
#include <iomanip>
#include <iostream>
#include <queue>
#include <unordered_map>
#include <vector>

using namespace std;

// Node of Huffman Tree
struct Node {
    char ch;
    int freq;
    Node *left, *right;

    Node(char ch, int freq, Node* left = nullptr, Node* right = nullptr)
    {
        this->ch = ch;
        this->freq = freq;
        this->left = left;
        this->right = right;
```

```
    }
};

// Comparison function for priority queue
struct compare {
    bool operator()(Node* left, Node* right)
    {
        return left->freq > right->freq;
    }
};

// Function to build the Huffman Tree
Node* buildHuffmanTree(const unordered_map<char, int>& freq)
{
    priority_queue<Node*, vector<Node*>, compare> pq;

    for (auto pair : freq) {
        pq.push(new Node(pair.first, pair.second));
    }

    while (pq.size() != 1) {
        Node* left = pq.top();
        pq.pop();
        Node* right = pq.top();
        pq.pop();

        int sum = left->freq + right->freq;
        pq.push(new Node('\0', sum, left, right));
    }
```

```
    return pq.top();
}

// Function to encode the input string
void encode(
    Node* root, const string& str, unordered_map<char, string>& huffmanCode)
{
    if (root == nullptr)
        return;

    if (!root->left && !root->right) {
        huffmanCode[root->ch] = str;
    }

    encode(root->left, str + "0", huffmanCode);
    encode(root->right, str + "1", huffmanCode);
}

// Function to decode the encoded string
string decode(Node* root, const string& str)
{
    string result = "";
    Node* curr = root;
    for (char bit : str) {
        if (bit == '0') {
            curr = curr->left;
        } else {
            curr = curr->right;
        }
```

```
        if (!curr->left && !curr->right) {
            result += curr->ch;
            curr = root;
        }
    }
    return result;
}

int main()
{
    cout << "\n5C6 - Amit Singhal (11614802722)" << endl;

    string text;
    cout << "\nEnter the text to encode: ";
    getline(cin, text);

    unordered_map<char, int> freq;
    for (char ch : text) {
        freq[ch]++;
    }

    clock_t start = clock();
    Node* root = buildHuffmanTree(freq);
    clock_t end = clock();
    double time_taken_build_tree
        = double(end - start) * 1000.0 / CLOCKS_PER_SEC;

    unordered_map<char, string> huffmanCode;
    start = clock();
    encode(root, "", huffmanCode);
```

```
end = clock();
double time_taken_encoding = double(end - start) * 1000.0 / CLOCKS_PER_SEC;

cout << "\nCharacter Encoding Table:" << endl;
cout << "------------------------------" << endl;
cout << setw(10) << "Character" << setw(20) << "Huffman Code" << endl;
cout << "------------------------------" << endl;
for (auto pair : huffmanCode) {
    cout << setw(10) << pair.first << setw(20) << pair.second << endl;
}
cout << "------------------------------" << endl;

cout << "Time taken to build Huffman Tree: " << time_taken_build_tree
     << " milliseconds" << endl;

string encodedString = "";
for (char ch : text) {
    encodedString += huffmanCode[ch];
}

cout << "\nEncoded String: " << encodedString << endl;
cout << "Time taken for encoding: " << time_taken_encoding
     << " milliseconds" << endl;

start = clock();
string decodedString = decode(root, encodedString);
end = clock();
double time_taken_decoding = double(end - start) * 1000.0 / CLOCKS_PER_SEC;

cout << "\nDecoded String: " << decodedString << endl;
```

```
    cout << "Time taken for decoding: " << time_taken_decoding
         << " milliseconds" << endl;

    return 0;
}
```

Output ::

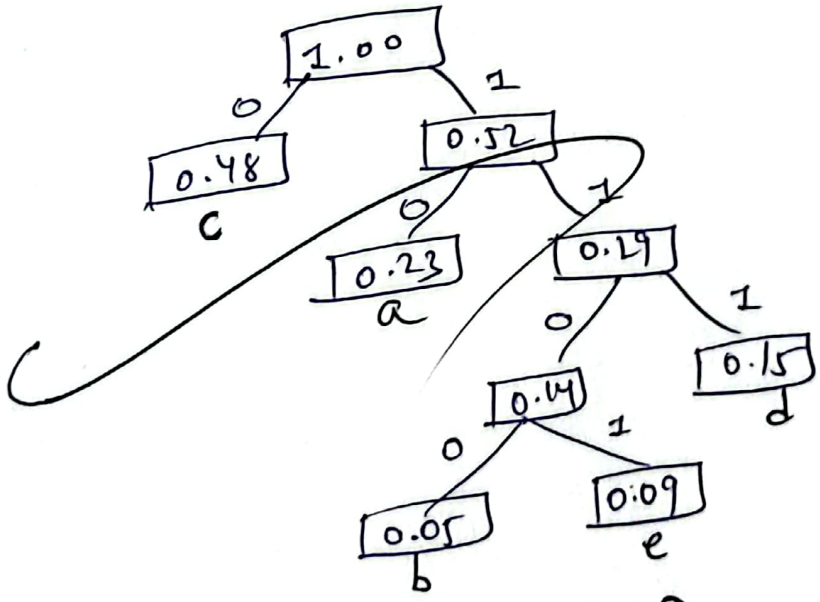
```
amit@Toshiba-Satellite-C850:~/Downloads$ g++ exp_3.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads$ ./a

5C6 - Amit Singhal (11614802722)

Enter the text to encode: Amit Singhal

Character Encoding Table:
------------------------------
 Character        Huffman Code
------------------------------
         l                1110
         i                 110
         n                1111
         a                1010
                          1001
         g                1011
         t                 010
         S                1000
         m                 001
         A                 011
         h                 000
------------------------------
Time taken to build Huffman Tree: 0.034 milliseconds

Encoded String: 01100111001010011000110111111011000010101110
Time taken for encoding: 0.022 milliseconds

Decoded String: Amit Singhal
Time taken for decoding: 0.004 milliseconds
```

VIVA QUESTIONS

Q.1) What is Huffman Coding? Explain with an example?

Ans.1) Huffman coding is been used to assign each character with a variable length code. The length of each character is been decided based on it's occurances & frequency. The character which occurs the most no. of times would have the smallest code & vice-versa

For ex: $L = \Sigma(a,b,c,d,e) : \langle 0.23, 0.05, 0.48, 0.15, 0.09 \rangle$



Therefore,
a : 10
b : 1100
c : 0
d : 111
e : 1101

let text, = "ccaccedbc"

Uniform encoding ⇒ # Bits = 3×10 = 30 bits

Huffman encoding ⇒ # Bits = 17 bits

(# Bits)$_{\text{Uniform encoding}}$ > (# Bits)$_{\text{Huffman coding}}$

Lab Exercise - 4

- AIM :: WAP in C++ to find Minimum Spanning Tree for a Graph & also evaluate its time complexity.

Source_Code ::

```
#include <chrono>
#include <climits>
#include <iomanip>
#include <iostream>
#include <vector>

using namespace std;
using namespace std::chrono;

struct Edge {
    int src, dest, weight;
};

// Function to display the graph in a table format
void displayGraph(int V, const vector<Edge>& edges)
{
    cout << "Original Graph:\n";
    cout << setw(10) << left << "Edges" << setw(10) << left << "Weights"
```

```
        << endl;
    cout << "-----------------" << endl;
    for (const auto& edge : edges) {
        cout << setw(1) << edge.src << " - " << setw(8) << edge.dest << setw(10)
            << edge.weight << endl;
    }
}

// Function to convert edge list to adjacency matrix
vector<vector<int>> toAdjacencyMatrix(int V, const vector<Edge>& edges)
{
    vector<vector<int>> adjMatrix(V, vector<int>(V, 0));
    for (const auto& edge : edges) {
        adjMatrix[edge.src][edge.dest] = edge.weight;
        adjMatrix[edge.dest][edge.src]
            = edge.weight; // Since the graph is undirected
    }
    return adjMatrix;
}

// Function to find the vertex with the minimum key value
int minKey(const vector<int>& key, const vector<bool>& inMST)
{
    int min = INT_MAX, min_index;

    for (int v = 0; v < key.size(); ++v) {
```

```
        if (!inMST[v] && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }

    return min_index;
}

// Function to implement Prim's algorithm to find the MST
void primMST(int V, const vector<vector<int>>& graph)
{
    vector<int> parent(V, -1); // Array to store constructed MST
    vector<int> key(V, INT_MAX); // Key values to pick minimum weight edge
    vector<bool> inMST(
        V, false); // To represent vertices not yet included in MST

    key[0] = 0; // Start from the first vertex

    for (int count = 0; count < V - 1; ++count) {
        int u = minKey(key, inMST);
        inMST[u] = true;

        for (int v = 0; v < V; ++v) {
            if (graph[u][v] && !inMST[v] && graph[u][v] < key[v]) {
                parent[v] = u;
```

```
                key[v] = graph[u][v];
            }
        }
    }
    // Print the constructed MST
    cout << "\nMinimum Spanning Tree (MST):\n";
    cout << setw(10) << left << "Edges" << setw(10) << left << "Weights"
         << endl;
    cout << "------------------" << endl;
    for (int i = 1; i < V; ++i) {
        cout << setw(1) << parent[i] << " - " << setw(8) << i << setw(10)
             << graph[i][parent[i]] << endl;
    }
}

int main() {
    cout << "\n5C6 - Amit Singhal (11614802722)\n" << endl;

    int V = 4; // Number of vertices in the graph
    vector<Edge> edges = { { 0, 1, 7 }, { 0, 2, 9 }, { 0, 3, 14 },
                           { 1, 2, 10 }, { 1, 3, 15 }, { 2, 3, 11 } };

    displayGraph(V, edges);

    // Convert edge list to adjacency matrix
    vector<vector<int>> adjMatrix = toAdjacencyMatrix(V, edges);
```

```
    // Measure the time taken to find the MST
    auto start = high_resolution_clock::now();
    primMST(V, adjMatrix);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "\nTime taken to find MST: " << duration.count()
        << " microseconds\n";

    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads/_LAB_Work/DAA/Code$ g++ exp_4.cpp -o a
amit@Toshiba-Satellite-C850:~/Downloads/_LAB_Work/DAA/Code$ ./a

5C6 - Amit Singhal (11614802722)

Original Graph:
Edges     Weights
-----------------
0 - 1       7
0 - 2       9
0 - 3       14
1 - 2       10
1 - 3       15
2 - 3       11

Minimum Spanning Tree (MST):
Edges     Weights
------------------
0 - 1       7
0 - 2       9
2 - 3       11

Time taken to find MST: 24 microseconds
```

Topic: Experiment - 04. Date: 03/09/24

Viva Questions on Exp - 04.

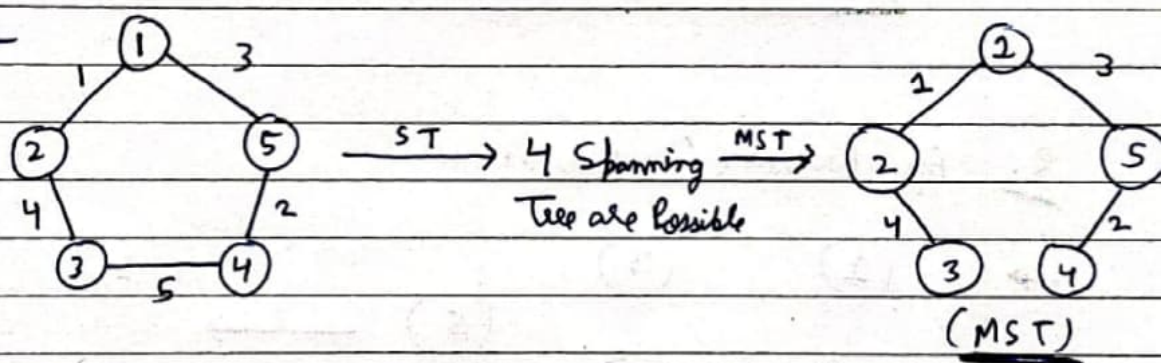
Q→ What are Spanning Tree & MST?

→ A Spanning Tree is an Sub-Graph of Undirected Connected Graph. It Includes all The Vertices of The Graph & Min. Possible Edges.

→ The Spanning Tree should have all Connected Components but No Cycle.
→ The Spanning Tree must have V - Vertices & |V|-1 Edges.
→ For a Complete Graph of V-Vertices, Then, #ST = $V^{(V-2)}$ Trees.

→ An MST is a Subset of The Edges of The Graph That Connects all The Vertices Together with No Cycle & Min. Possible Edge Weight Sum.
→ If The Edge Weights are distinct, Then, There Can be only 1 Unique MST.
→ For a Complete Graph, By Removing (E - V+1) Edges, We Can obtain ST.

Q Eg :-



Q→ In which Way, we Can Represent a Graph?

→ The 2 Ways To Represent Graph are :-

1. Adjacency Matrix : It Is a (V X V) Matrix, Where, V = # Vertices.

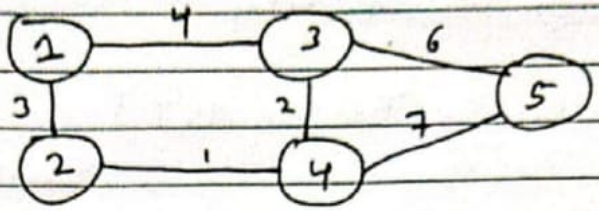
$$a[i][j] = \begin{cases} 1, & \text{If Edge Exist b/w Vertex-i \& Vertex-j} \\ 0, & \text{If Edge Not Exist.} \end{cases}$$

2. Adjacency List : The Adjacency List Is a Way To Represent Graph using Linkedlist. The Space Required Is O(V + 2E).

Teacher Sign .....

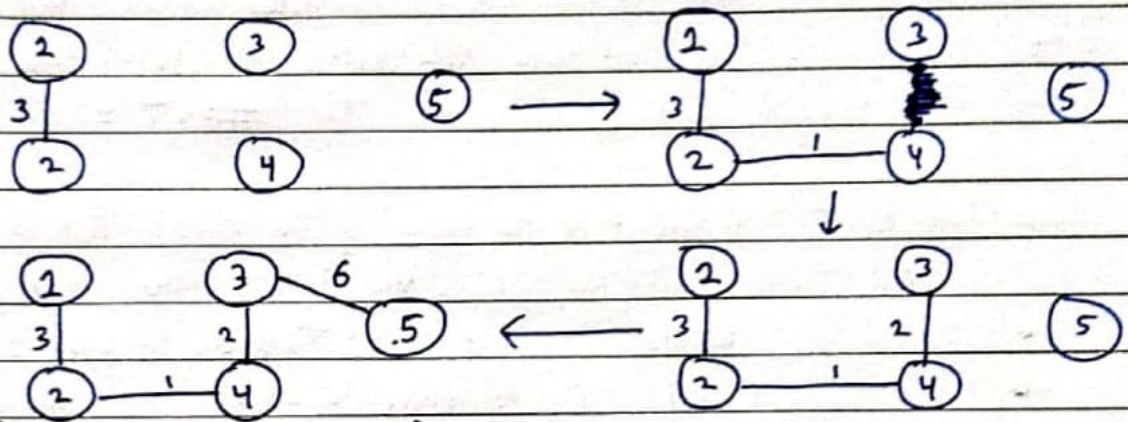
Topic: ..... Date: .....

Q → Obtain The MST Of Below Graph using Prim & Kruskal Algo



1. By Prim's Algorithm

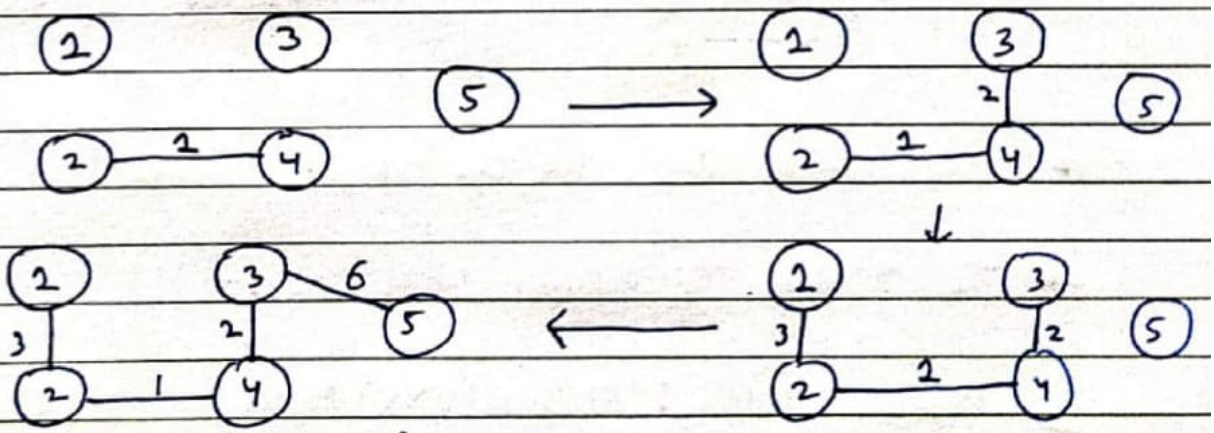
→



(MST with Weight = 12)

2. By Kruskal's Algorithm

→



(MST with Weight = 12)

★ In Prim's Algorithm, The Graph Is Connected In The Intermediate Stages, But, In Kruskal's Algorithm, The Intermediate Graph Can be Disconnected But Final MST's Of Both Algo. will be Connected Graphs.

Classroom

Teacher Sign .....

Lab Exercise - 5

❖ AIM :: WAP in C++ to implement Dijsktra's Algorithm & also calculate time complexity to find the shortest path

Source_Code ::

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

using namespace std;
using namespace std::chrono;

// Structure to represent an edge in the graph
struct Edge {
    int to;
    int weight;
};

// Function to add an edge to the adjacency list
void addEdge(vector<vector<Edge> >& graph, int u, int v, int weight) {
    graph[u].push_back({v, weight});
    graph[v].push_back({u, weight});  // For undirected graph
```

```
}

// Function to display the graph
void displayGraph(const vector<vector<Edge> >& graph) {
    cout << "Graph adjacency list representation:\n";
    for (int i = 0; i < graph.size(); ++i) {
        cout << "Node " << i << ": ";
        for (const auto& edge : graph[i]) {
            cout << "(to: " << edge.to << ", weight: " << edge.weight << ") ";
        }
        cout << endl;
    }
}

// Dijkstra's algorithm implementation
vector<int> dijkstra(const vector<vector<Edge> >& graph,
                     int source,
                     int64_t& timeTaken) {
    int n = graph.size();
    vector<int> dist(n, INT_MAX);  // Distance array, initialized to infinity
    dist[source] = 0;              // Distance to source is 0

    // Priority queue to store {distance, node}
    priority_queue<pair<int, int>, vector<pair<int, int> >,
                   greater<pair<int, int> > >
        pq;
```

```
pq.push({0, source});

// Measure time start
auto start = high_resolution_clock::now();

while (!pq.empty()) {
    int u = pq.top().second;  // Get the node with the smallest distance
    int d = pq.top().first;   // Get the distance of that node
    pq.pop();

    // If the distance in the queue is greater than the already found
    // shortest distance, skip
    if (d > dist[u])
        continue;

    // Explore the neighbors of node u
    for (const auto& edge : graph[u]) {
        int v = edge.to;
        int weight = edge.weight;

        // Relaxation step
        if (dist[u] + weight < dist[v]) {
            dist[v] = dist[u] + weight;
            pq.push({dist[v], v});
        }
    }
```

```
    }

    // Measure time end
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<nanoseconds>(stop - start);
    timeTaken = duration.count();  // Time in nanoseconds

    return dist;
}

int main() {
    cout << "\n5C6 - Amit Singhal (11614802722)\n" << endl;

    int n, e, source;

    // Input: Number of nodes and edges
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> e;

    vector<vector<Edge> > graph(n);

    // Input: Edges
    cout << "\nEnter the edges (u, v, weight):\n";
    for (int i = 0; i < e; ++i) {
        int u, v, weight;
        cin >> u >> v >> weight;
```

```
    addEdge(graph, u, v, weight);
}

cout << endl;

// Display the graph
displayGraph(graph);

// Input: Source node
cout << "\nEnter the source node: ";
cin >> source;

// Time taken for calculating the shortest paths
int64_t totalTime;

// Find shortest paths from the source node to all other nodes
vector<int> dist = dijkstra(graph, source, totalTime);

// Display shortest distances from the source to all other nodes
cout << "\nShortest distances from node " << source << " to all other nodes:\n";
for (int i = 0; i < dist.size(); ++i) {
    if (dist[i] == INT_MAX) {
        cout << "To node " << i << " : Unreachable\n";
    } else {
        cout << "To node " << i << " : " << dist[i] << endl;
    }
```

```
    }


    // Display time complexity
    cout << "\nTime taken to compute shortest paths from node " << source
        << ": " << totalTime << " nanoseconds" << endl;


    return 0;
}
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~/Documents$ g++ prg5.cpp
singhal-amit@singhal-amit-ThinkPad-T430:~/Documents$ ./a.out

5C6 - Amit Singhal (11614802722)

Enter the number of nodes and edges: 4 5

Enter the edges (u, v, weight):
0 1 10
0 2 20
1 2 5
1 3 2
2 3 4

Graph adjacency list representation:
Node 0: (to: 1, weight: 10) (to: 2, weight: 20)
Node 1: (to: 0, weight: 10) (to: 2, weight: 5) (to: 3, weight: 2)
Node 2: (to: 0, weight: 20) (to: 1, weight: 5) (to: 3, weight: 4)
Node 3: (to: 1, weight: 2) (to: 2, weight: 4)

Enter the source node: 0

Shortest distances from node 0 to all other nodes:
To node 0 : 0
To node 1 : 10
To node 2 : 15
To node 3 : 12

Time taken to compute shortest paths from node 0: 15306 nanoseconds
```

Title Experiment - 05. Date 10-9-24

Page No. 1.

Viva Questions of Exp. -05.

Q→ What Is Dijkstra's Algorithm ? What Is It's Time Complexity ?

→ Dijkstra's Algorithm is used To find The shortest path between The Starting Node and To all other Vertices of The Graph. It Works on The Graph The Edges will have all Non-Negative Weights.

→ In Case of Dijkstra's Algorithm :-

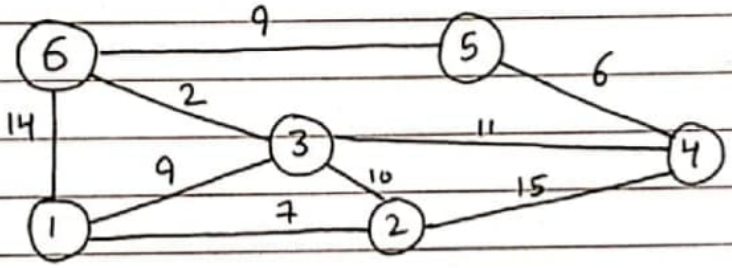
If, dist [A][C] + dist [C][B] < dist [A][B]

Then, dist [A][B] = dist [A][C] + dist [C][B] (Relaxation Step)

→ The Time Complexity of Dijkstra's Algorithm Is O(E log V).

* DRAW BACK :- This Algorithm Fails for (-Ve) Weight Cycles.

Q→ Apply Dijkstra's Algorithm for Graph where Source = 1 ?



→

Source	Destination				
1	2	3	4	5	6
	∞	∞	∞	∞	∞
Itr - I.	(7)	9	∞	∞	14
Itr - II.	(7)	(9)	22	∞	14
Itr - III.	(7)	(9)	20	∞	(11)
Itr - IV.	(7)	(9)	(20)	20	(11)
Itr - V.	(7)	(9)	(20)	(20)	(11)

* The Dijkstra's Algorithm fails for directed Graph having -Ve cycles.

DELTA©