Experiment 10

WAP to perform the empirical analysis of Greedy Huffman algorithm to find the prefix code of the input data.

Naïve algorithm

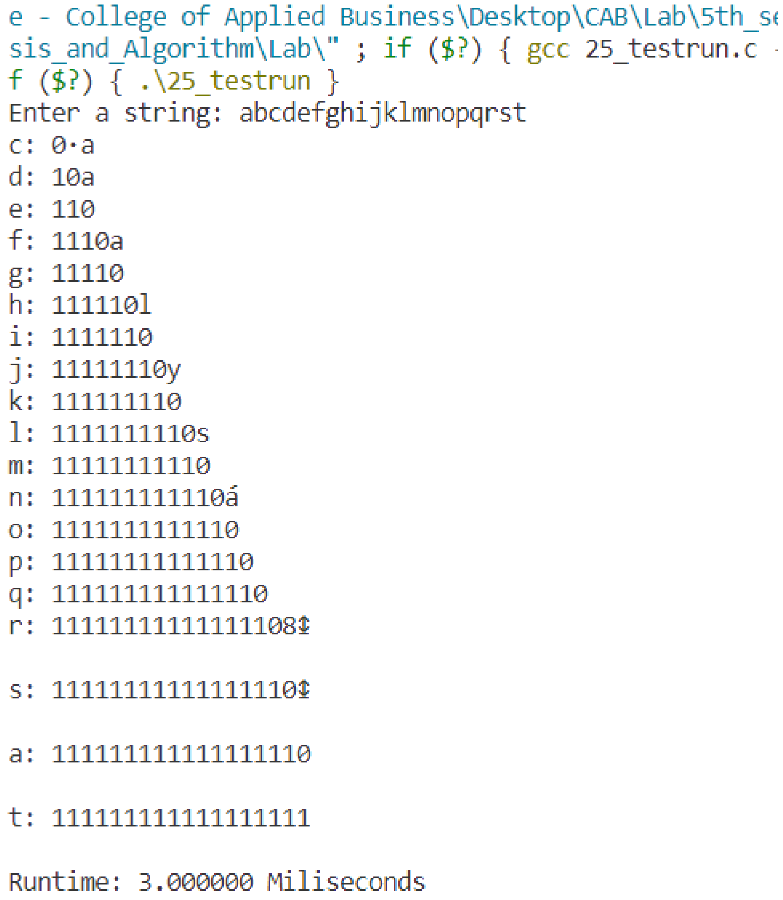
Program:-

```
#include <stdio.h>
#include<string.h>
#include <stdlib.h>
#include <time.h>
#define MAX_CHARS 256
struct Node {
    int freq;
    char data;
    struct Node *left, *right;
};
struct MinHeap {
    int size;
    int capacity;
    struct Node** array;
};
struct MinHeap* createMinHeap(int capacity) {
    struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct Node**)malloc(capacity * sizeof(struct Node*));
    return minHeap;
}
void swapMinHeapNode(struct Node** a, struct Node** b) {
    struct Node* t = *a;
    *a = *b;
    *b = t;
```

```
}
void minHeapify(struct MinHeap* minHeap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq) {
        smallest = left;
    }
    if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq) {
        smallest = right;
    }
    if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }     }
int isSizeOne(struct MinHeap* minHeap) {
    return (minHeap->size == 1);
}
struct Node* extractMin(struct MinHeap* minHeap) {
    if (isSizeOne(minHeap)) {
        return minHeap->array[0];
    }
    struct Node* root = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return root;
}
void insertMinHeap(struct MinHeap* minHeap, struct Node* node) {
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && node->freq < minHeap->array[(i - 1) / 2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    minHeap->array[i] = node;
}
// Function to create a Huffman tree
struct Node* createHuffmanTree(char data[], int freq[], int size) {
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i) {
        struct Node* node = (struct Node*)malloc(sizeof(struct Node));
        node->left = node->right = NULL;
        node->data = data[i];
        node->freq = freq[i];
        insertMinHeap(minHeap, node);
    }
    // Build the Huffman tree
    struct Node *left, *right, *top;
    for (int i = 0; i < size - 1; ++i) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = (struct Node*)malloc(sizeof(struct Node));
        top->freq = left->freq + right->freq;
        top->data = '\0';  // Internal node doesn't hold a character
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);  // Root of the Huffman tree}
void generateCodes(struct Node* root, char arr[], int top){
    if (root->left) {
        arr[top] = '0';
        generateCodes(root->left, arr, top + 1); }
    if (root->right) {
        arr[top] = '1';
        generateCodes(root->right, arr, top + 1); }
    if (root->left == NULL && root->right == NULL) {
        printf("%c: %s\n", root->data, arr);
    }      }
int main() {
    char data[MAX_CHARS]; // Input data
    int freq[MAX_CHARS] = {0};
    int size;
    printf("Enter a string: ");
    fgets(data, MAX_CHARS, stdin);  // Get input from user
    size = strlen(data) - 1;
    for (int i = 0; i < size; ++i) {
        ++freq[data[i]];
    }
    clock_t start = clock();
    struct Node* root = createHuffmanTree(data, freq, size);
    char arr[MAX_CHARS];
    generateCodes(root, arr, 0);
    clock_t end = clock();
    double runtime = ((double)(end - start)*1000) / CLOCKS_PER_SEC;
    printf("Runtime: %f Miliseconds\n", runtime);
    return 0;
}
```

Result Analysis and Discussion:

This experiment has been conducted in a 64-bit system with 16 GB RAM and Processor 12th Gen Intel(R) Core (TM) i5-12500H 3.10 GHz. The algorithm is implemented in C programming language in Visual Studio Code 1.85.1 Code Editor. In this experiment the algorithm to find the prefix code of the input data using Greedy Huffman algorithm has been implemented and executed.



```
e - College of Applied Business\Desktop\CAB\Lab\5th_se
sis_and_Algorithm\Lab\" ; if ($?) { gcc 25_testrun.c -
f ($?) { .\25_testrun }
Enter a string: abcdefghijklmnopqrst
c: 0·a
d: 10a
e: 110
f: 1110a
g: 11110
h: 111110l
i: 1111110
j: 11111110y
k: 111111110
l: 1111111110s
m: 11111111110
n: 111111111110á
o: 1111111111110
p: 11111111111110
q: 111111111111110
r: 11111111111111108‡

s: 1111111111111110‡

a: 111111111111111110

t: 111111111111111111

Runtime: 3.000000 Miliseconds
```

Conclusion:

The prefix code of the data was found by using Greedy Huffman algorithm. The running time is analyzed as O(nlogn)