

DESIGN AND ANALYSIS OF ALGORITHM

NAME: UPPU MANJU SRI NATH

ROLL NO:CH.SC.U4CSE24249

1)HUFFMAN PROGRAM

CODE:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

struct Node {
    char data;
    unsigned freq;
    struct Node *left, *right;
};

struct Node* newNode(char data, unsigned freq) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}

struct MinHeap {
    unsigned size;
    struct Node* arr[MAX];
};

void swap(struct Node** a, struct Node** b) {
```

```

struct Node* t = *a;
*a = *b;
*b = t;
}

void heapify(struct MinHeap* heap, int i) {
    int smallest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;

    if (left < heap->size && heap->arr[left]->freq < heap->arr[smallest]->freq)
        smallest = left;

    if (right < heap->size && heap->arr[right]->freq < heap->arr[smallest]->freq)
        smallest = right;

    if (smallest != i) {
        swap(&heap->arr[i], &heap->arr[smallest]);
        heapify(heap, smallest);
    }
}

struct Node* extractMin(struct MinHeap* heap) {
    struct Node* temp = heap->arr[0];
    heap->arr[0] = heap->arr[--heap->size];
    heapify(heap, 0);
    return temp;
}

void insertHeap(struct MinHeap* heap, struct Node* node) {
    int i = heap->size++;
    while (i && node->freq < heap->arr[(i-1)/2]->freq) {
        heap->arr[i] = heap->arr[(i-1)/2];
        i = (i-1)/2;
    }
    heap->arr[i] = node;
}

struct Node* buildTree(char data[], int freq[], int size) {

```

```

struct MinHeap heap;
heap.size = 0;

for (int i = 0; i < size; i++)
    heap.arr[heap.size++] = newNode(data[i], freq[i]);

for (int i = (heap.size-1)/2; i >= 0; i--)
    heapify(&heap, i);

while (heap.size > 1) {
    struct Node* left = extractMin(&heap);
    struct Node* right = extractMin(&heap);

    struct Node* top = newNode('$', left->freq + right->freq);
    top->left = left;
    top->right = right;

    insertHeap(&heap, top);
}

return extractMin(&heap);
}

void printCodes(struct Node* root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    if (!root->left && !root->right) {
        printf("%c: ", root->data);
        for (int i = 0; i < top; i++)
            printf("%d", arr[i]);
        printf("\n");
    }
}

```

```

}
}

int main() {
char data[] = {'A','B','C','D','E','F'};
int freq[] = {5,9,12,13,16,45};
int size = sizeof(data)/sizeof(data[0]);

struct Node* root = buildTree(data, freq, size);

int arr[MAX], top = 0;
printf("Huffman Codes:\n");
printCodes(root, arr, top);

return 0;
}

```

OUTPUT:

```

manjusrinath2604@manjusrinath2604:~$ cd Documents
manjusrinath2604@manjusrinath2604:~/Documents$ gcc -o huffman huffman.c
manjusrinath2604@manjusrinath2604:~/Documents$ ./huffman
Huffman Codes:
F: 0
C: 100
D: 101
A: 1100
B: 1101
E: 111
manjusrinath2604@manjusrinath2604:~/Documents$

```

Time Complexity: $O(N + M \log M)$:

$O(n + m \log m)$ where N is the input string length and M is the number of unique characters – the frequency counting takes $O(N)$ while building the Huffman tree with m nodes takes $O(M \log M)$ due to repeated extractMin and insert operations in the min-heap.

Space Complexity: $O(N + M)$:

$O(N + M)$ where N stores the input string and M stores the Huffman tree nodes and related data structures – since M is bounded by 256 (ASCII characters), this effectively becomes $O(N)$ for practical purposes.

2)JOB SEQUENCING PROGRAM

CODE:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int id;
    int deadline;
    int profit;
} Job;

int compare(const void *a, const void *b) {
    Job *j1 = (Job *)a;
    Job *j2 = (Job *)b;
    return j2->profit - j1->profit;
}

int main() {
    int n;
    printf("Enter number of jobs: ");
    scanf("%d", &n);

    Job jobs[n];

    for(int i = 0; i < n; i++) {
```

```
printf("Enter deadline and profit for job %d: ", i+1);
scanf("%d %d", &jobs[i].deadline, &jobs[i].profit);
jobs[i].id = i + 1;
}
```

```
qsort(jobs, n, sizeof(Job), compare);
```

```
int maxDeadline = 0;
for(int i = 0; i < n; i++)
if(jobs[i].deadline > maxDeadline)
maxDeadline = jobs[i].deadline;
```

```
int slot[maxDeadline];
for(int i = 0; i < maxDeadline; i++)
slot[i] = -1;
```

```
int totalProfit = 0;
```

```
for(int i = 0; i < n; i++) {
for(int j = jobs[i].deadline - 1; j >= 0; j--) {
if(slot[j] == -1) {
slot[j] = jobs[i].id;
totalProfit += jobs[i].profit;
break;
}
}
}
```

```
printf("\nScheduled Jobs: ");
for(int i = 0; i < maxDeadline; i++)
if(slot[i] != -1)
printf("J%d ", slot[i]);
```

```
printf("\nTotal Profit = %d\n", totalProfit);
```

```
return 0;
}
```

OUTPUT:

```
manjusrinath2604@manjusrinath2604:~$ cd Documents
manjusrinath2604@manjusrinath2604:~/Documents$ gcc -o jobsequencing jobsequencing.c
manjusrinath2604@manjusrinath2604:~/Documents$ ./ jobsequencing
bash: ./: Is a directory
manjusrinath2604@manjusrinath2604:~/Documents$ ./jobsequencing
Enter number of jobs: 5
Enter deadline and profit for job 1: 32
2
Enter deadline and profit for job 2: 23
1
Enter deadline and profit for job 3: 45
5
Enter deadline and profit for job 4: 64
3
Enter deadline and profit for job 5: 91
4

Scheduled Jobs: J2 J1 J3 J4 J5
Total Profit = 15
manjusrinath2604@manjusrinath2604:~/Documents$
```

TIME COMPLEXITY: $O(N^2)$:

THE PROGRAM TAKES $O(N^2)$ TIME BECAUSE IT USES BUBBLE SORT TO ORDER JOBS BY PROFIT AND A NESTED LOOP TO ASSIGN JOBS TO AVAILABLE TIME SLOTS IN THE WORST CASE.

SPACE COMPLEXITY : (O):

THE PROGRAM REQUIRES $O(N)$ SPACE TO STORE THE JOBS AND THE TIME SLOT ARRAY, WHICH ARE PROPORTIONAL TO THE NUMBER OF JOBS.