Rashtriya Raksha University

School of Information Technology, Artificial Intelligence & Cyber Security (SITAICS)

At- Lavad, Dahegam, Gandhinagar, Gujarat-382305

Practical File

(Design & Analysis of Algorithms)

Name: Sarthak Sanay
Enrollment No: 230031101611051
Subject Name: Design & Analysis of Algorithms (G4AD18DAA)
Program: B.Tech CSE (with specialization in Cyber Security)
Year: 2nd year (Semester-IV)

This is to certify that **Mr. Sarthak Sanay** has satisfactorily completed **10** out of **10** practical work prescribed by SITAICS (School of Information Technology, Artificial Intelligence, & Cyber Security) at the **Networking** laboratory.

Dr. Ravi Sheth
SUBJECT INCHARGE

INDEX

NAME: SARTHAK SANAY

ENROLLMENT NO: 230031101611051

SUBJECT: DESIGN & ANALYSIS OF ALGORITHMS

SUBJECT CODE: (G4AD18DAA)

PRACTICAL - 1

AIM: To Perform Different String Operations

CODE:

```
// Write a program in C to implement string functions copy(),
compare(), and concatenate() by clearly defining it on your own.

#include <stdio.h>
#include <string.h>

void sanCopy(char a[], char b[]) {
    int i;
    for(i = 0; b[i] != '\0'; i++) {
        a[i] = b[i];
    }
    a[i] = '\0';
    printf("\nCopied string: %s\n", a);
}

void sanCompare(char a[], char b[]) {
    int i = 0;

    // compare the strings character by character
    while (a[i] != '\0' && b[i] != '\0') {
        if (a[i] < b[i]) {
            printf("\nString 1 is smaller than String 2.\n");
            return;
        } else if (a[i] > b[i]) {
            printf("\nString 1 is greater than String 2.\n");
            return;
        }
        i++;
    }

    if (a[i] == '\0' && b[i] != '\0') {
        printf("\nString 1 is smaller than String 2.\n");
    }
    else if (b[i] == '\0' && a[i] != '\0') {
```

```
        printf("\nString 1 is greater than String 2.\n");
    }
    else {
        printf("\nBoth strings are equal.\n");
    }
}

void sanConcatenate(char a[], char b[]) {
    int aLen = strlen(a);
    int bLen = strlen(b);

    int totalLen = aLen + bLen;
    char concat[totalLen + 1];  // +1 for null terminator

    int count = 0;
    for(int i = 0; i < aLen; i++, count++) {
        concat[count] = a[i];
    }
    for(int j = 0; j < bLen; j++, count++) {
        concat[count] = b[j];
    }
    concat[count] = '\0';

    printf("\n%s\n", concat);
}

int main() {
    int ch = 1;
    char s1[20], s2[20];

    printf("Enter 1st string: ");
    fgets(s1, sizeof(s1), stdin);
    s1[strcspn(s1, "\n")] = 0;  // Remove trailing newline

    printf("Enter 2nd string: ");
    fgets(s2, sizeof(s2), stdin);
    s2[strcspn(s2, "\n")] = 0;  // Remove trailing newline

    while (ch != 0) {
        printf("\nEnter 1 to COPY string."
               "\nEnter 2 to COMPARE string."
               "\nEnter 3 to CONCATENATE string."
               "\nEnter 0 to EXIT."
               "\nEnter your choice: ");

        if (scanf("%d", &ch) != 1) {
            printf("\nInvalid input. Please enter a number.\n");
```

```
            continue;
        }

        switch(ch) {
            case 1:
                sanCopy(s1, s2);
                break;

            case 2:
                sanCompare(s1, s2);
                break;

            case 3:
                sanConcatenate(s1, s2);
                break;

            case 0:
                printf("\nProgram exited successfully!\n");
                break;

            default:
                printf("Enter correct choice.\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
@sanaysarthak →/workspaces/daa-lab/Manual String Functions (main) $ ./a.out
Enter 1st string: Sarthak
Enter 2nd string: Sanay

Enter 1 to COPY string.
Enter 2 to COMPARE string.
Enter 3 to CONCATENATE string.
Enter 0 to EXIT.
Enter your choice: 2

String 1 is greater than String 2.
```

```
 Enter 1 to COPY string.
 Enter 2 to COMPARE string.
 Enter 3 to CONCATENATE string.
 Enter 0 to EXIT.
 Enter your choice: 3

 SarthakSanay

 Enter 1 to COPY string.
 Enter 2 to COMPARE string.
 Enter 3 to CONCATENATE string.
 Enter 0 to EXIT.
 Enter your choice: 1

 Copied string: Sanay

 Enter 1 to COPY string.
 Enter 2 to COMPARE string.
 Enter 3 to CONCATENATE string.
 Enter 0 to EXIT.
 Enter your choice: 2
Both strings are equal.

Enter 1 to COPY string.
Enter 2 to COMPARE string.
Enter 3 to CONCATENATE string.
Enter 0 to EXIT.
Enter your choice: 3

SanaySanay

Enter 1 to COPY string.
Enter 2 to COMPARE string.
Enter 3 to CONCATENATE string.
Enter 0 to EXIT.
Enter your choice: 0

Program exited successfully!
```

PRACTICAL - 2

AIM: To implement the Binary Search Algorithm and find its Time Complexity

CODE:

```
#include <stdio.h>
#include <sys/time.h>

int binarySearch(int array[], int startIndex, int endIndex, int
x)
{
  if (endIndex >= startIndex)  // used for checking if the
position is at the extreme ends of the array or not
  {
    int mid = startIndex + (endIndex - startIndex) / 2;

    // If found at mid, then return it
    if(array[mid] == x)
      return mid;

    // Search the left half
    else if(array[mid] > x)
      return binarySearch(array,  startIndex, mid - 1,x);

    // Search the right half
    else
        return binarySearch(array, mid + 1, endIndex, x);
  }
  return -1;
}

int main()
{
    struct timeval start, end;

    int size, ele;
    printf("Enter the size of an array: ");
```

```
    scanf("%d", &size);

    int arr[size];
    printf("\nEnter elements in the array :-\n");
    for(int i=0; i<size; i++)
    {
        printf("Enter element %d : ", i);
        scanf("%d", &arr[i]);
    }

    printf("\nEnter a number to search it in the array: ");
    scanf("%d", &ele);
    printf("Performing Binary Search algorithm to check whether
the number is in the array or not.\n");

    gettimeofday(&start, NULL);
    printf("\nTime before function call = %ld microseconds\n",
start.tv_usec);

    int res = binarySearch(arr, 0, size-1, ele);
    if (res == -1)
        printf("The number %d is not in the Array.", ele);
    else
        printf("The number %d is present at index %d.\n", ele,
res);

    gettimeofday(&end, NULL);
    printf("Time after function call = %ld microseconds\n",
end.tv_usec);
    printf("\nTotal running time = %ld microseconds\n",
end.tv_usec - start.tv_usec);

    return 0;
}
```

P.T.O.

OUTPUT:

```
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ gcc 2-time-binary-search.c
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ ./a.out
Enter the size of an array: 10

Enter elements in the array :-
Enter element 0 : 5
Enter element 1 : 6
Enter element 2 : 12
Enter element 3 : 24
Enter element 4 : 31
Enter element 5 : 35
Enter element 6 : 36
Enter element 7 : 65
Enter element 8 : 78
Enter element 9 : 91

Enter a number to search it in the array: 65
Performing Binary Search algorithm to check whether the number is in the array or not.

Time before function call = 253516 microseconds
The number 65 is present at index 7.
Time after function call = 253560 microseconds

Total running time = 44 microseconds
```

PRACTICAL - 3

AIM: To implement the Merge Sort Algorithm

CODE:

```
#include <stdio.h>

// Function to merge two halves of the array
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int leftArr[n1], rightArr[n2];

    for (int i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        rightArr[i] = arr[mid + 1 + i];

    // Merge the temp arrays back into arr
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = rightArr[j];
```

```
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        // Recursively sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Original Array: \n");
    printArray(arr, n);

    mergeSort(arr, 0, n - 1);

    printf("\nSorted Array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

```
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ gcc 4-merge-sort.c
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ ./a.out
Enter the number of elements: 10
Enter the elements:
23
78
64
256
18
7
45
89
64
100
Original Array:
23 78 64 256 18 7 45 89 64 100

Sorted Array:
7 18 23 45 64 64 78 89 100 256
```

PRACTICAL - 4

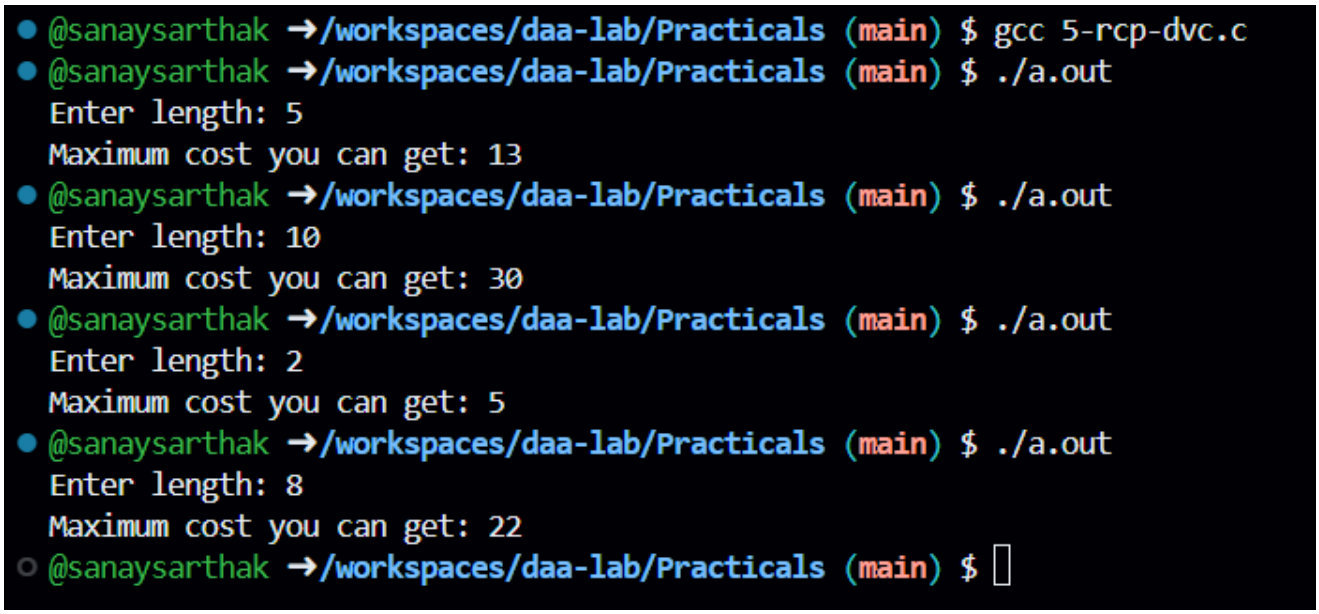
AIM: To solve the Rod Cutting Problem using the Divide and Conquer Approach

CODE:

```
#include <stdio.h>

// Function to compute the maximum value using a
divide-and-conquer approach
int cutRod(int price[], int n) {
    int max_val = 0;

    // Try every possible first cut
    for (int i = 0; i < n; i++) {
        int revenue = price[i] + cutRod(price, n - i - 1);
        if (revenue > max_val) {
            max_val = revenue;
        }
    }

    return max_val;
}

int main() {
    // Example price list (for rod lengths 1 to 10)
    int price[] = {1, 5, 8, 9, 10, 17, 17, 20, 24, 30};
    int n = sizeof(price) / sizeof(price[0]);

    int user_len;
    printf("Enter length: ");
    scanf("%d", &user_len);

    if (user_len >= 1 && user_len <= n)
        printf("Maximum cost you can get: %d\n", cutRod(price,
user_len));
    else
```

```
        printf("Invalid length! Enter a value between 1 and
%d\n", n);

    return 0;
}
```

OUTPUT:



PRACTICAL - 5

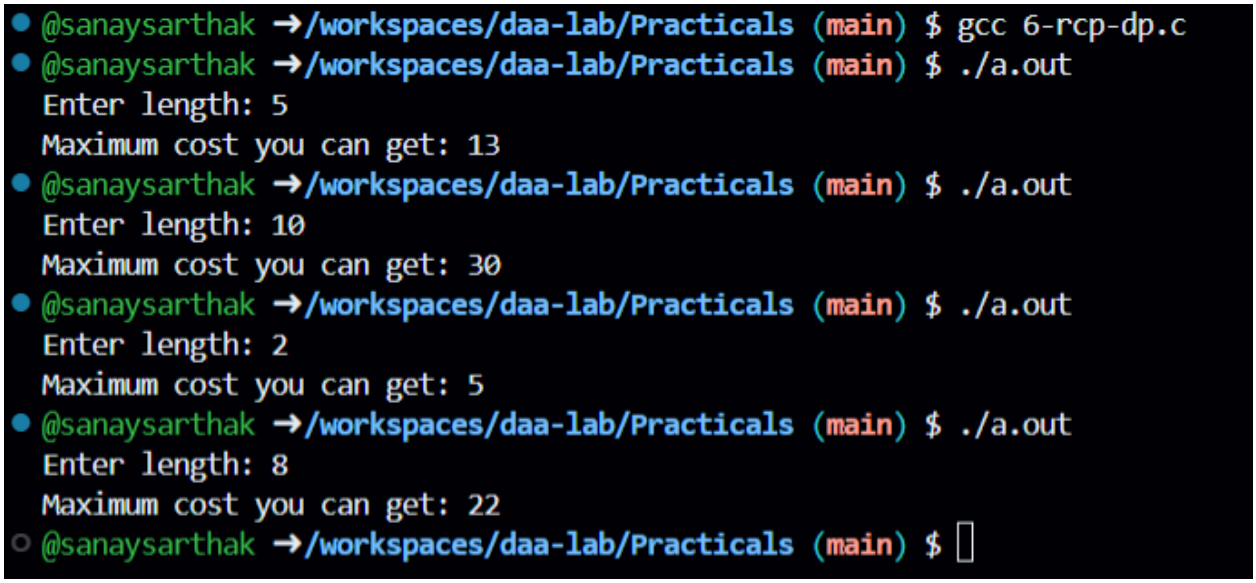
AIM: To solve the Rod Cutting Problem using the Dynamic Programming Approach

CODE:

```
#include <stdio.h>

// Function to compute the maximum value using
dynamic-programming approach
int cutRod(int price[], int n) {
    // dp[j] will hold the maximum revenue for a rod of length j
    int dp[n + 1];
    dp[0] = 0;

    for (int j = 1; j <= n; j++) {
        int max_val = 0;
        // try cutting off a first piece of length (i+1),
leaving j-(i+1)
        for (int i = 0; i < j; i++) {
            int revenue = price[i] + dp[j - i - 1];
            if (revenue > max_val) {
                max_val = revenue;
            }
        }
        dp[j] = max_val;
    }
    return dp[n];
}

int main() {
    // Example price list (for rod lengths 1 to 10)
    int price[] = {1, 5, 8, 9, 10, 17, 17, 20, 24, 30};
    int n = sizeof(price) / sizeof(price[0]);

    int user_len;
    printf("Enter length: ");
    scanf("%d", &user_len);
```

```
    if (user_len >= 1 && user_len <= n)
        printf("Maximum cost you can get: %d\n", cutRod(price,
user_len));
    else
        printf("Invalid length! Enter a value between 1 and
%d\n", n);

    return 0;
}
```

OUTPUT:



PRACTICAL - 6

AIM: To implement the code for LCS (Longest Common Subsequence) in strings, count the total number of subsequences formed, and display the longest one.

CODE:

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

int LCSLength(char X[], char Y[], int m, int n, int dp[][n+1]) {
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            } else if (X[i-1] == Y[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
            } else {
                dp[i][j] = (dp[i-1][j] > dp[i][j-1]) ?
dp[i-1][j] : dp[i][j-1];
            }
        }
    }
    return dp[m][n];
}

void printLCS(char X[], char Y[], int m, int n, int dp[][n+1]) {
    char lcs[dp[m][n] + 1];
    int i = m, j = n, index = dp[m][n];

    lcs[index] = '\0';
    while (i > 0 && j > 0) {
        if (X[i-1] == Y[j-1]) {
            lcs[--index] = X[i-1];
            i--; j--;
        } else if (dp[i-1][j] > dp[i][j-1]) {
            i--;
        } else {
```

```
            j--;
        }
    }
    printf("\nLongest Common Subsequence:  %s\n", lcs);
    printf("\n");
}

int main() {
    char X[] = "ABDECGF";
    char Y[] = "ACBDFGH";
    int m = strlen(X);
    int n = strlen(Y);
    int dp[m+1][n+1];

    // Calculate the length of LCS
    LCSLength(X, Y, m, n, dp);
    printLCS(X, Y, m, n, dp);
    return 0;
}
```

OUTPUT:

```
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ gcc 7-lcs.c
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ ./a.out

  Longest Common Subsequence:  ABDF
```

PRACTICAL - 7

AIM: To implement the Activity Selection Problem using Greedy Algorithm.

CODE:

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

struct Activity {
    int start;
    int finish;
};

// Swap function for sorting
void swap(struct Activity *a, struct Activity *b) {
    struct Activity temp = *a;
    *a = *b;
    *b = temp;
}

// Sort activities by finish time
void sortActivities(struct Activity activities[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (activities[i].finish > activities[j].finish) {
                swap(&activities[i], &activities[j]);
            }
        }
    }
}

// Function to select activities
void activitySelection(struct Activity activities[], int n) {

    // Sort by finish time
    sortActivities(activities, n);
    printf("\nSelected activities (start, finish):\n");
```

```
    int lastFinish = activities[0].finish;
    printf("(%d, %d)\n", activities[0].start,
activities[0].finish);

    // Select remaining compatible activities
    for (int i = 1; i < n; i++) {
        if (activities[i].start >= lastFinish) {
            printf("(%d, %d)\n", activities[i].start,
activities[i].finish);
            lastFinish = activities[i].finish;
        }
    }
}

int main() {
    int n;
    printf("Enter the number of activities: ");
    scanf("%d", &n);
    struct Activity activities[n];

    printf("Enter start and finish times of each activity:\n");
    for (int i = 0; i < n; i++) {
        printf("Activity %d - Start: ", i + 1);
        scanf("%d", &activities[i].start);
        printf("Activity %d - Finish: ", i + 1);
        scanf("%d", &activities[i].finish);
    }

    activitySelection(activities, n);
    return 0;
}
```

P.T.O.

OUTPUT:

```
● @sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ gcc 8-asp-greedy.c
● @sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ ./a.out
  Enter the number of activities: 5
  Enter start and finish times of each activity:
  Activity 1 - Start: 1
  Activity 1 - Finish: 6
  Activity 2 - Start: 0
  Activity 2 - Finish: 3
  Activity 3 - Start: 4
  Activity 3 - Finish: 10
  Activity 4 - Start: 2
  Activity 4 - Finish: 6
  Activity 5 - Start: 4
  Activity 5 - Finish: 8

  Selected activities (start, finish):
  (0, 3)
  (4, 8)
```

PRACTICAL - 8

AIM: To implement the Knapsack Problem using Dynamic Programming.

CODE:

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

int max(int a, int b) {
    return (a > b) ? a : b;
}

// Knapsack DP function
int knapsack(int W, int weight[], int value[], int n) {
    int dp[n + 1][W + 1];

    // Build table dp[][] in bottom-up manner
    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 - 1] <= w)
                dp[i][w] = max(value[i - 1] + dp[i - 1][w -
weight[i - 1]], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    // Track selected items
    int selected[n];
    for (int i = 0; i < n; i++) selected[i] = 0;

    int w = W;
    for (int i = n; i > 0 && w > 0; i--) {
        if (dp[i][w] != dp[i - 1][w]) {
            selected[i - 1] = 1;
```

```
            w -= weight[i - 1];
        }
    }

    printf("\nAll entered item values: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", value[i]);
    }

    printf("\nAll entered item weights: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", weight[i]);
    }

    printf("\n\nSelected item indices (0-based): ");
    for (int i = 0; i < n; i++) {
        if (selected[i])
            printf("%d ", i);
    }

    printf("\nSelected item values: ");
    for (int i = 0; i < n; i++) {
        if (selected[i])
            printf("%d ", value[i]);
    }

    printf("\nSelected item weights: ");
    for (int i = 0; i < n; i++) {
        if (selected[i])
            printf("%d ", weight[i]);
    }

    printf("\n");
    return dp[n][W];
}

int main() {
    int n, W;
    printf("Enter number of items: ");
    scanf("%d", &n);
    int value[n], weight[n];

    printf("Enter the values of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &value[i]);
    }
```

```
    printf("Enter the weights of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weight[i]);
    }

    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &W);
    int result = knapsack(W, weight, value, n);
    printf("\nMaximum value in knapsack of capacity %d = %d\n",
W, result);

    return 0;
}
```

OUTPUT:

```
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ gcc 9-knapsack-dp.c
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ ./a.out
Enter number of items: 5
Enter the values of the items:
50
20
45
10
5
Enter the weights of the items:
19
20
56
47
33
Enter the capacity of the knapsack: 30

All entered item values: 50 20 45 10 5
All entered item weights: 19 20 56 47 33

Selected item indices (0-based): 0
Selected item values: 50
Selected item weights: 19

Maximum value in knapsack of capacity 30 = 50
```

PRACTICAL - 9

AIM: To implement the Knapsack Problem using Greedy Algorithm.

CODE:

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

struct Item {
    int value;
    int weight;
    float ratio;
};

// Function to compare items by value/weight ratio (descending)
int compare(const void *a, const void *b) {
    float r1 = ((struct Item *)a)->ratio;
    float r2 = ((struct Item *)b)->ratio;
    return (r2 > r1) - (r2 < r1);
}

int main() {
    int n;
    float capacity;
    printf("Enter number of items: ");
    scanf("%d", &n);
    struct Item items[n];

    printf("Enter the values of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &items[i].value);
    }

    printf("Enter the weights of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &items[i].weight);
```

```
        items[i].ratio = (float)items[i].value /
items[i].weight;
    }

    printf("Enter the capacity of the knapsack: ");
    scanf("%f", &capacity);

    // Sort items by ratio
    qsort(items, n, sizeof(struct Item), compare);
    float total_value = 0.0;
    float total_weight = 0.0;

    printf("\nSelected items (value, weight, fraction):\n");
    for (int i = 0; i < n && capacity > 0; i++) {
        if (items[i].weight <= capacity) {
            // Take full item
            capacity -= items[i].weight;
            total_value += items[i].value;
            total_weight += items[i].weight;
            printf("(%d, %d, 1.00)\n", items[i].value,
items[i].weight);
        }
        else {
            // Take fractional part
            float fraction = capacity / items[i].weight;
            total_value += items[i].value * fraction;
            total_weight += items[i].weight * fraction;
            printf("(%d, %d, %.2f)\n", items[i].value,
items[i].weight, fraction);
            capacity = 0;  // Knapsack full
        }
    }
    printf("\nTotal weight used: %.2f\n", total_weight);
    printf("Maximum value in knapsack = %.2f\n", total_value);

    return 0;
}
```

P.T.O.

OUTPUT:

```
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ gcc 10-knapsack-greedy.c
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ ./a.out
Enter number of items: 5
Enter the values of the items:
20
30
50
10
15
Enter the weights of the items:
66
23
10
40
53
Enter the capacity of the knapsack: 50

Selected items (value, weight, fraction):
(50, 10, 1.00)
(30, 23, 1.00)
(20, 66, 0.26)

Total weight used: 50.00
Maximum value in knapsack = 85.15
```

PRACTICAL - 10

AIM: To implement the Huffman Coding Algorithm

CODE:

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

#define MAX_CODE_LEN 100

typedef struct Node {
    char ch;
    int freq;
    struct Node *l, *r;
} Node;

Node* new_node(char c, int f) {
    Node* n = malloc(sizeof(Node));
    n->ch   = c;
    n->freq = f;
    n->l = n->r = NULL;
    return n;
}

int cmp_node(const void *a, const void *b) {
    Node *na = *(Node**)a;
    Node *nb = *(Node**)b;
    return na->freq - nb->freq;
}

void print_codes(Node *root, char code[], int depth) {
    if (!root) return;
    if (!root->l && !root->r) {
        code[depth] = '\0';
        printf("'%c' -> %s\n", root->ch, code);
    } else {
        code[depth] = '0';
        print_codes(root->l, code, depth+1);
        code[depth] = '1';
```

```
        print_codes(root->r, code, depth+1);
    }
}

int main(void) {
    int n;
    printf("Enter number of characters: ");
    if (scanf("%d", &n)!=1 || n<=0) return 0;

    // read inputs
    char *chars = malloc(n);
    int  *freqs = malloc(n * sizeof(int));
    printf("\n");
    for (int i = 0; i < n; i++) {
        printf("Character %d: ", i+1);
        scanf(" %c", &chars[i]);
        printf("Frequency of '%c': ", chars[i]);
        scanf("%d", &freqs[i]);
        printf("\n");
    }

    Node **arr = malloc(n * sizeof(Node*));
    for (int i = 0; i < n; i++)
        arr[i] = new_node(chars[i], freqs[i]);

    int sz = n;
    while (sz > 1) {
        qsort(arr, sz, sizeof(Node*), cmp_node);
        Node *a = arr[0], *b = arr[1];
        Node *m = new_node('\0', a->freq + b->freq);
        m->l = a;  m->r = b;
        arr[1] = m;
        arr[0] = arr[sz-1];
        sz--;
    }

    printf("\nGenerated Huffman Codes:\n");
    char code[MAX_CODE_LEN];
    print_codes(arr[0], code, 0);

    return 0;
}
```

OUTPUT:

```
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ gcc huffman-coding.c
@sanaysarthak ➜/workspaces/daa-lab/Practicals (main) $ ./a.out
Enter number of characters: 5

Character 1: T
Frequency of 'T': 4

Character 2: I
Frequency of 'I': 3

Character 3: A
Frequency of 'A': 6

Character 4: G
Frequency of 'G': 2

Character 5: O
Frequency of 'O': 1


Generated Huffman Codes:
'O' -> 000
'G' -> 001
'I' -> 01
'T' -> 10
'A' -> 11
```