Design and Analysis of Algorithm

(KCS553)

LABORATORY MANUAL

B.TECH III rd YEAR – V th SEMESTER

`

United College of Engineering & Research, Praygraj
Department of Computer Science & Engineering

Vision of the Department

- To enhance effective teaching and learning by strengthening high academic goals of the students and strong academic leadership among the faculty members. Besides, the department envisages generating an active professional and research environment through industry and R & D orientation at the national and international levels.

- To become a self-sustained unit with its expanded depth and breadth in the areas of Computer Science & Engineering focusing on innovative educational research, applied and interdisciplinary nature of Applied Computing, Consultancy and Training.

Mission of the Department

- To provide Quality education for latest technologies and involving them in live projects in order to achieve the highest standards in theoretical and practical aspects across the computer science discipline.

- To develop necessary skills in collaboration with industry and academia inculcating sincere learning and nobility in profession.

- To develop technical abilities and skills along with its practical implementation in youth to meet the need of profession and society.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: To provide students the necessary fundamentals of mathematics, science and engineering to create, select and apply appropriate techniques, resources and modern IT tools including simulation and modeling to complex engineering application. Further to prepare them for R & D and consultancy enabling them formulate, solve and analyze engineering problems for the higher learning and professional outcomes.

PEO2: To provide students adequate exposure to skill enhancement, trainings and opportunities to work as teams on multidisciplinary projects with effective communication skills and leadership qualities enabling them equipped with the abilities to work logically, accurately, ethically and efficiently, to generate new knowledge, ideas or products, to implement these solutions in practice, and to develop an ability to analyze the requirements of the software, its design and its technical specifications yielding novel engineering solutions.

PEO3: To prepare students for a successful career and work with social and human values meeting the requirements of Indian and multinational companies. Further, to design, construct, implement and evaluate a computer based system, process, component or program to meet desired needs within realistic constraints such as economics, environmental, social, political, health and safety, manufacturability and sustainability and to promote student awareness on the life-long value-based professional learning.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity.

PSO 2: The ability to understand the evolutionary changes in computing, apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success, real world problems and meet the challenges of the future.

PSO 3: The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, lifelong learning and a zest for higher studies and also to act as a good citizen by inculcating in them moral values & ethics.

PROGRAM OUTCOMES (POs)

PO -1	**Engineering Knowledge**: Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to Computer Science.
PO -2	**Problem Analysis**: Identify, Formulate, review research literature and analyze complex engineering problems related to CS and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
PO -3	**Design/Development of solutions**: Design solutions for complex engineering problems related to computer science and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural societal and environmental considerations
PO -4	**Conduct Investigations of Complex problems**: Use research–based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO -5	**Modern Tool Usage**: Create, Select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to computer science related complex engineering activities with an understanding of the limitations
PO -6	**The Engineer and Society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the computer science professional engineering practice
PO -7	**Environment and Sustainability:** Understand the impact of the computer science professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development
PO -8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO -9	**Individual and Team Work**: Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary Settings
PO -10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large such as able to comprehend and with write effective reports and design documentation, make effective presentations and give and receive clear instructions.
PO -11	**Project Management and Finance**: Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments
PO -12	**Life-Long Learning**: Recognize the need for and have the preparation and ability to engage in independent and life-long learning the broadest context of technological change

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3. Student should enter into the laboratory with:

   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

   b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.

   c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

OBJECTIVES AND OUTCOMES

OBJECTIVES:

1. To understand the importance of time complexity in context of writing efficient programs for real world problems.
2. To develop skills to apply appropriate data structures in problem solving and compare the complexity of various algorithms.

COURSE OUTCOMES:

Course Outcome (CO)	Bloom's Knowledge Level (KL)	
At the end of course , the student will be able to:		
CO 1	To implement Recursive programs and Selection sort.	K1, K2
CO 2	To study and implement Merge sort, heap sort quick sort and counting sort.	K2, K3
CO 3	To develop program greedy design strategy such as Knapsack problem	K2, K3
CO 4	To Implement program based on Dynamic Programming and Backtracking such as Longest Common Subsequence, Knapsack Problem and N queen problem.	K2, K3
CO 5	To learn and implement String matching algorithms and approximation algorithms.	K3, K4

RECOMMENDED SYSTEM / SOFTWARE REQUIREMENTS:

1. Intel based desktop PC of 166MHz or faster processor with at least 64 MB RAM and 100 MB free disk space.

2. Turbo C++ compiler or GCC compilers .

USEFUL TEXT BOOKS / REFERECES:

1. Thomas H. Coreman, Charles E. Leiserson and Ronald L. Rivest, "Introduction to Algorithms", Printice Hall of India.
2. E. Horowitz & S Sahni, "Fundamentals of Computer Algorithms"
3. Aho, Hopcraft, Ullman, "The Design and Analysis of Computer Algorithms" Pearson Education, 2008

List of Programs

Topic	Program list	Course Outcome	Bloom's Level
Searching	1. Write a program in C to implement Recursive Linear Search.	CO1	K1,K2
	2. Write a program in C to implement Recursive Binary Search.	CO1	K2,K3
Sorting	3. Write a program in C to implement Bubble Sort and Selection Sort.	CO1	K2,K3
	4. Write a program in C to implement Insertion Sort.	CO1	K2,K3
	5. Write a program in C to implement Quick Sort.	CO1	K2,K3
	6. Write a program in C to implement Merge Sort.	CO2	K2,K3
	7. Write a program in C to implement Counting Sort.	CO2	K2,K3
	8. Write a program in C to implement Heap Sort.	CO2	K2,K3
	9. Write a program in C to implement Radix Sort.	CO2	K2,K3

Dynammic Programming	10. Write a program in C to implement Matrix Chain Multiplication problem.	CO4	K2,K3
	11. Write a program in C to implement Longest Chain Subsequence. 12. Write a program in C to implement 0/1 Kanpsack Problem	CO4	K2,K3
Greedy Knapsack	13. Write a program in C to implement fractional Kanpsack Problem	CO4	K2,K3
String Matching	14. Write a program in C on the Naïve String matching problem. 15. Write a program in C on the KMP String matching problem.	CO5	K2,K3

Program on the Recursive Linear Search

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

int RecusiveLS(int ar[],int value,int index,int n)
{
int pos=0;
if(index>=n)
return 0;
else if ( ar[index]== value )
            {   pos= index +1;
              return pos;
           }
    return  RecursiveLS(ar,value,index+1,n)
}
void main()
{    int *a,i,n,m;
   clrscr();
   printf("Enter the size of an array: ");
   scanf("%d",&n);
            a=(int*)malloc(n*sizeof(int));

   printf("Enter the elements  ");
   for(i=0;i<n;i++)
            {
             scanf("%d",(a+i));
   }

   printf("Enter the number to be search: ");
   scanf("%d",&m);
c=RecursiveLS(a,m,0,n);
if(c==0)
             printf("The number is not found.\n");
   else
             printf("The number %d is found at position %d\n",m,pos+1);
      getch();
}
```

Program on the Recursive Binary Search

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int c=0;
int RecusiveBS(int ar[],int element,int start_index,int end_index)
{
if (end_index >= start_index){
      int middle = start_index + (end_index - start_index )/2;
      if (array[middle] == element)
         return middle;
      if (array[middle] > element)
         return RecusiveBS (ar, start_index, middle-1, element);
      return RecusiveBS (ar, middle+1, end_index, element);
   }
Return -1;
}
void main()
{

   int *a,i,n,m,pos;
   clrscr();
   printf("Enter the size of an array: ");
   scanf("%d",&n);
          a=(int*)malloc(n*sizeof(int));

   printf("Enter the elements in ascending order: ");
   for(i=0;i<n;i++)
   {
           scanf("%d",(a+i));
   }

   printf("Enter the number to be search: ");
   scanf("%d",&m);
          c= RecusiveBS (a,0,n-1,m);


   if(c==-1)
           printf("The number is not found.\n");
   else
           printf("The number %d is found at position %d\n",m,c+1);
  getch();}
```

Program on Bubble Sort

```
/*SORTING OF INTEGERS BY BUBBLE SORT METHOD*/
#define MAXSIZE 25
void main()
{
 int a[MAXSIZE],i,j,k,temp,n;
 clrscr();
 printf("HOW MANY ELEMENTS IN ARRAY(<25):");
 scanf("%d",&n);
 printf("ENTER %d INTEGER:\n",n);
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 for(i=0;i<n;i++)
 {
  for(j=0;j<n-1-i;j++)
  {
   if(a[j]>a[j+1])
   {
    temp=a[j];
    a[j]=a[j+1];
    a[j+1]=temp;
   }
  }
 }
 printf("SORTED LIST IS AS FOLLOW:\n");
 for(i=0;i<n;i++)
  printf("%d ",a[i]);
 getch();
}
```

Program on Selection Sort

```
#define MAXSIZE 25
void main()
{
 int a[MAXSIZE],i,j,n,temp,loc,min;
 clrscr();
 printf("HOW MANY ELEMENT IN ARRAY:");
 scanf("%d",&n);
 printf("ENTER %d ELEMENT:\n",n);
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 for(i=0;i<n;i++)
 {
  min=a[i];
  loc=i;
  for(j=i+1;j<n;j++)
  {
   if(a[j]<min)
   {
    min=a[j];
    loc=j;
   }
  }
  if(loc!=i)
  {
   temp=a[i];
   a[i]=a[loc];
   a[loc]=temp;
  }
 }
 printf("SORTED LIST IS AS FOLLOWS:\n");
 for(i=0;i<n;i++)
  printf("%d ",a[i]);
 getch();
}
```

Program on Insertion Sort

```
/insettion sort
#define MAXSIZE 25
void main()
{
 int a[MAXSIZE],i,j,n,temp;
 clrscr();
 printf("HOW MANY ELEMENT IN ARRAY:");
 scanf("%d",&n);
 printf("ENTER %d ELEMENT:\n",n);
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 for(i=1;i<n;i++)
 {
  temp=a[i];
  j=i-1;
  while(temp<a[j] && j>=0)
  {
   a[j+1]=a[j];
   j--;
  }
  a[j+1]=temp;
 }
 printf("SORTED LIST IS AS FOLLOWS:\n");
 for(i=0;i<n;i++)
  printf("%d ",a[i]);
 getch();
}
```

Program on Quick Sort

Implementation of Quick Sort Algorithm in C:

```
# include <stdio.h>

// to swap two numbers
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];  // selecting last element as pivot
    int i = (low - 1);  // index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If the current element is smaller than or equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
/*
    a[] is the array, p is starting index, that is 0,
    and r is the last index of array.
*/
void quicksort(int a[], int p, int r)
{
    if(p < r)
    {
        int q;
        q = partition(a, p, r);
        quicksort(a, p, q-1);
        quicksort(a, q+1, r);
    }
}


// function to print the array
void printArray(int a[], int size)
{
```

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

int main()
{
    int arr[] = {9, 7, 5, 11, 12, 2, 14, 3, 10, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    // call quickSort function
    quicksort(arr, 0, n-1);

    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
```

Program on the Merge Sort

```
int arr[20];       // array to be sorted

int main()
{
 int n,i;

 printf("Enter the size of array\n");  // input the elements
 scanf("%d",&n);
 printf("Enter the elements:");
 for(i=0; i<n; i++)
  scanf("%d",&arr[i]);

 merge_sort(arr,0,n-1);  // sort the array

 printf("Sorted array:");  // print sorted array
 for(i=0; i<n; i++)
  printf("%d",arr[i]);

 return 0;
}

int merge_sort(int arr[],int low,int high)
{
 int mid;
 if(low<high) {
  mid=(low+high)/2;
  // Divide and Conquer
  merge_sort(arr,low,mid);
  merge_sort(arr,mid+1,high);
  // Combine
  merge(arr,low,mid,high);
 }

 return 0;
}

int merge(int arr[],int l,int m,int h)
{
 int arr1[10],arr2[10];  // Two temporary arrays to
 hold the two arrays to be merged
 int n1,n2,i,j,k;
 n1=m-l+1;
 n2=h-m;
```

```
  for(i=0; i<n1; i++)
    arr1[i]=arr[l+i];
  for(j=0; j<n2; j++)
    arr2[j]=arr[m+j+1];

  arr1[i]=9999;  // To mark the end of each temporary array
  arr2[j]=9999;

  i=0;
  j=0;
  for(k=l; k<=h; k++) { //process of combining two sorted arrays
    if(arr1[i]<=arr2[j])
      arr[k]=arr1[i++];
    else
      arr[k]=arr2[j++];
  }

  return 0;
}
```

Program on the Counting Sort

```
#include <stdio..h>

#include <conio.h>

void Counting_sort(int [], int, int);

void main()

{ int *a,i,n,m;

   clrscr();

   printf("Enter the size of an array: ");

   scanf("%d",&n);

   a=(int*)malloc(n*sizeof(int));

   printf("Enter the elements  ");

   for(i=0;i<n;i++)

{ scanf("%d",(a+i));

  }

  Counting_sort(A, k, n);

getch();

}

void Counting_sort(int A[], int k, int n)

{           int i, j;

            int B[15], C[100];

            for(i = 0; i <= k; i++)

                        C[i] = 0;
```

```
        for(j =1; j <= n; j++)
                        C[A[j]] = C[A[j]] + 1;
        for(i = 1; i <= k; i++)
                        C[i] = C[i] + C[i-1];
        for(j = n; j >= 1; j--)
        {
                        B[C[A[j]]] = A[j];
                        C[A[j]] = C[A[j]] - 1;
        }
printf(" The sorted array ");
for(i=1;i<=n;i++)
printf("%d",B[i]);
}
```

Program on Heap Sort

```
void heapify(int arr[], int n, int i)
{
   int largest = i;  // Initialize largest as root
   int l = 2*i + 1;  // left = 2*i + 1
   int r = 2*i + 2;  // right = 2*i + 2
   int temp;

   if (l < n && arr[l] > arr[largest])
            largest = l;


   if (r < n && arr[r] > arr[largest])
            largest = r;


   if (largest != i)
   {
                                    temp=arr[i];
                           arr[i]=arr[largest];
                           arr[largest]=temp;

            heapify(arr, n, largest);
   }
}


void heapSort(int arr[], int n)
{        int i,temp;

   for (i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);


   for (i=n-1; i>=0; i--)    {

                                    temp=arr[0];
                           arr[0]=arr[i];
                           arr[i]=temp;

            heapify(arr, i, 0);
   }
```

```
}
int main()
{
   int n,i;

  printf("Enter the size of array\n");  // input the elements
  scanf("%d",&n);
  printf("Enter the elements:");
  for(i=0; i<n; i++)
   scanf("%d",&arr[i]);


   heapSort(arr, n);

  printf("Sorted array:");  // print sorted array
  for(i=0; i<n; i++)
   printf("%d",arr[i]);


    return 0;
}
```

Program on Radix Sort

```
#include<stdio.h>
int getMax(int arr[], int n) {
 int mx = arr[0];
    int i;
    for (i = 1; i < n; i++)
     if (arr[i] > mx)
       mx = arr[i];
 return mx;
}

      void countSort(int arr[], int n, int exp) {
 int output[n]; // output array
  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];

 // Build the output array
 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];
}

// The main function to that sorts arr[] of size n using Radix Sort
void radixsort(int arr[], int n) {
 int m = getMax(arr, n);

 int exp;
 for (exp = 1; m / exp > 0; exp *= 10)
    countSort(arr, n, exp);
}

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

```
int main()
{
  int n,i;

 printf("Enter the size of array\n");  // input the elements
 scanf("%d",&n);
 printf("Enter the elements:");
 for(i=0; i<n; i++)
```

```
    scanf("%d",&arr[i]);


    radixsort(arr, n);

  printf("Sorted array:");  // print sorted array
  for(i=0; i<n; i++)
    printf("%d",arr[i]);


     return 0;
}
```

Program on Matrix Chain Multiplication

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

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n

int MatrixChainMultiplication(int p[], int n)
{
    int m[n][n];
    int i, j, k, L, q;

    for (i=1; i<n; i++)
        m[i][i] = 0;    //number of multiplications are 0(zero) when there is only one matrix

    //Here L is chain length. It varies from length 2 to length n.
    for (L=2; L<n; L++)
    {
        for (i=1; i<n-L+1; i++)
        {
            j = i+L-1;
            m[i][j] = INT_MAX;  //assigning to maximum value

            for (k=i; k<=j-1; k++)
            {
                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (q < m[i][j])
                {
                    m[i][j] = q;    //if number of multiplications found less that number will be updated.
```

```
            }
        }
    }
}

    return m[1][n-1];   //returning the final answer which is M[1][n]

}

int main()
{
    int n,i;
    printf("Enter number of matrices\n");
    scanf("%d",&n);

    n++;

    int arr[n];

    printf("Enter dimensions \n");

    for(i=0;i<n;i++)
    {
        printf("Enter d%d :: ",i);
        scanf("%d",&arr[i]);
    }

    int size = sizeof(arr)/sizeof(arr[0]);

    printf("Minimum number of multiplications is %d ", MatrixChainMultiplication(arr, size));

    return 0;
}
```

Program on Longest Common Sequence

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

int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];

void print(int i,int j)
{
	if(i==0 || j==0)
		return;
	if(b[i][j]=='c')
	{
		print(i-1,j-1);
		printf("%c",x[i-1]);
	}
	else if(b[i][j]=='u')
		print(i-1,j);
	else
		print(i,j-1);
}

void lcs()
{
	m=strlen(x);
	n=strlen(y);
	for(i=0;i<=m;i++)
		c[i][0]=0;
	for(i=0;i<=n;i++)
		c[0][i]=0;

	//c, u and l denotes cross, upward and downward directions respectively
	for(i=1;i<=m;i++)
		for(j=1;j<=n;j++)
		{
			if(x[i-1]==y[j-1])
			{
				c[i][j]=c[i-1][j-1]+1;
				b[i][j]='c';
			}
			else if(c[i-1][j]>=c[i][j-1])
			{
				c[i][j]=c[i-1][j];
```

```
                                        b[i][j]='u';
                                }
                                else
                                {
                                        c[i][j]=c[i][j-1];
                                        b[i][j]='l';
                                }
                        }
}
int main()
{
        printf("Enter 1st sequence:");
        scanf("%s",x);
        printf("Enter 2nd sequence:");
        scanf("%s",y);
        printf("\nThe Longest Common Subsequence is ");
        lcs();
        print(m,n);
return 0;
}
```

Program on 0/1 Knapsack Algorithm

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];
   for (i = 0; i <= n; i++)
   {
      for (w = 0; w <= W; w++)
      {
         if (i==0 || w==0)
            K[i][w] = 0;
         else if (wt[i-1] <= w)
               K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
         else
               K[i][w] = K[i-1][w];
      }
   }
   return K[n][W];
}
int main()
{
    int i, n, val[20], wt[20], W;

    printf("Enter number of items:");
    scanf("%d", &n);

    printf("Enter value and weight of items:\n");
    for(i = 0;i < n; ++i){
     scanf("%d%d", &val[i], &wt[i]);
    }

    printf("Enter size of knapsack:");
    scanf("%d", &W);

    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

Program on Fractional Knapsack Algorithm

```
# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity) {
   float x[20], tp = 0;
   int i, j, u;
   u = capacity;

   for (i = 0; i < n; i++)
      x[i] = 0.0;

   for (i = 0; i < n; i++) {
      if (weight[i] > u)
         break;
      else {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }
   }

   if (i < n)
      x[i] = u / weight[i];

   tp = tp + (x[i] * profit[i]);

   printf("\nThe result vector is:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);

   printf("\nMaximum profit is:- %f", tp);

}

int main() {
   float weight[20], profit[20], capacity;
   int num, i, j;
   float ratio[20], temp;

   printf("\nEnter the no. of objects:- ");
   scanf("%d", &num);
```

```
  printf("\nEnter the wts and profits of each object:- ");
  for (i = 0; i < num; i++) {
    scanf("%f %f", &weight[i], &profit[i]);
  }

  printf("\nEnter the capacityacity of knapsack:- ");
  scanf("%f", &capacity);

  for (i = 0; i < num; i++) {
    ratio[i] = profit[i] / weight[i];
  }

  for (i = 0; i < num; i++) {
    for (j = i + 1; j < num; j++) {
      if (ratio[i] < ratio[j]) {
        temp = ratio[j];
        ratio[j] = ratio[i];
        ratio[i] = temp;

        temp = weight[j];
        weight[j] = weight[i];
        weight[i] = temp;

        temp = profit[j];
        profit[j] = profit[i];
        profit[i] = temp;
      }
    }
  }

  knapsack(num, weight, profit, capacity);
  return(0);
}
```

Program on Naïve String matching Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void naive_string_matcher(int[],int[],int,int);
void main()
{
        int *txt,*pat,n,m,i,j,h;
        clrscr();
        printf("Enter the number of elements for Text");
        scanf("%d",&n);
        txt=(int*)calloc(n,sizeof(int));
        for(i=0;i<n;i++)
        {
                printf("Enter the value");
                scanf("%d",&txt[i]);
        }
        printf("Enter the number of elements for Pattern");
        scanf("%d",&m);
        pat=(int*)calloc(m,sizeof(int));
        for(i=0;i<m;i++)
        {
                printf("Enter the value");
                scanf("%d",&pat[i]);
        }
        naive_string_matcher(txt,pat,n,m);
        getch();

}
void naive_string_matcher(int txt[],int pat[],int n,int m)
{
        int j,h,i;
        for(j=0;j<=n-m;j++)
        {
                h=0;
                for(i=0;i<m;i++)
                {
                        if(pat[i]==txt[j+i])
                        {
                                h++;
                        }
                }

                if(h==m)
                {
```

```
                printf("\n \nCONGRATULATIONS..!!..PATTERN IS FOUND IN THE
STRING AT : %d",j);
                }

        }
                if(h==0)
                {
                printf("\n \n SORRY..!!..PATTERN IS NOT A SUBSTRING OF THE
STRING. \n \n");
                }

}
```

Program on KMP String matching Algorithm

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

void prefixSuffixArray(char* pat, int M, int* pps) {
  int length = 0;
  pps[0] = 0;
  int i = 1;
  while (i < M) {
    if (pat[i] == pat[length]) {
      length++;
      pps[i] = length;
      i++;
    } else {
      if (length != 0)
      length = pps[length - 1];
      else {
        pps[i] = 0;
        i++;
      }
    }
  }
}
void KMPAlgorithm(char* text, char* pattern) {
  int M = strlen(pattern);
  int N = strlen(text);
  int pps[M];
  prefixSuffixArray(pattern, M, pps);
  int i = 0;
  int j = 0;
  while (i < N) {
```

```
    if (pattern[j] == text[i]) {
      j++;
      i++;
    }
    if (j == M) {
      printf("Found pattern at index %d\n", i - j);
      j = pps[j - 1];
    }
    else if (i < N && pattern[j] != text[i]) {
      if (j != 0)
      j = pps[j - 1];
      else
      i = i + 1;
    }
  }
}
int main() {
  char text[] = "xyztrwqxyzfg";
  char pattern[] = "xyz";
  printf("The pattern is found in the text at the following index : \n");
  KMPAlgorithm(text, pattern);
  return 0;
}
```