Devang Patel Institute of Advance Technology & Research

**Department of Computer Science & Engineering**

## Subject Name: Design & Analysis of Algorithms Semester: 5th

Subject Code: CE342 Academic year: July-December 2020

Id: 18DCS007

**Index**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. No.** | **Name of Experiment** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | **Page No.** | |
| **1.** | **Implement and analyze algorithms given below.** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **1.1** | Factorial (Iterative and Recursive) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 6 | |
|  | **1.2** | Euclidean algorithm | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 12 | |
|  | **1.3** | Matrix Addition and Matrix Multiplication(Iterative) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 14 | |
|  | **1.4** | Recursive Linear Search and Binary Search | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 21 | |
|  | **1.5** | Find a subset of a given set S = {s1,s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1,2,6} and {1,8}.A suitable message is to be displayed if the given problem instance doesn't have  a solution. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 27 | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **2.** | **Implement and analyze algorithms given below.(Compare**  **them)** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **2.1** | Bubble Sort | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 30 | |
|  | **2.2** | Selection Sort | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 32 | |
|  | **2.3** | Insertion Sort | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 34 | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **3.** | **Divide and Conquer Strategy** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **3.1** | Implement and perform analysis of worst case of Merge  Sort and Quick sort. Compare both algorithms. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 37 | |
|  | **3.2** | Implement the program to find X^Y using divide and conquer strategy and print the total number of multiplications required to find X^Y. Test the program for following test cases: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 43 | |
|  | | | | | | | | | | **Test Case** | | | | | | | **X** | | **Y** |  | | | | | | | | | |
| 1 | | | | | | | 2 | | 6 |
| 2 | | | | | | | 7 | | 25 |
| 3 | | | | | | | 5 | | 34 |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **4.** | **Greedy Approach** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **4.1** | A cashier at any mall needs to give change of an amount | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 45 | |
|  | to customers many times in a day. Cashier has multiple | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | number of coins available with different denominations | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | which is described by a set C. Implement the program for | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | a cashier to find the minimum number of coins required to | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | find a change of a particular amount A. Output should be | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | the total number of coins required of given denominations. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | Check the program for following test cases: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | **Test Coin denominations C Amount A** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | **Case** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | 1 ₹1, ₹2, ₹3 ₹ 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | 2 ₹18, ₹17, ₹5, ₹1 ₹ 22 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | 3 ₹100, ₹25, ₹10, ₹5, ₹1 ₹ 289 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | Is the output of Test case 2 is optimal? Write your | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | observation. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | **4.2** | Let S be a collection of objects with profit-weight values. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 49 | |
|  | Implement the fractional knapsack problem for S | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | assuming we have a sack that can hold objects with total | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | weight W. Check the program for following test cases: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  | **Test** | | **S** | | | | | | | | | | | | | **profit-weight values** | | | | | | | | | **W** | | |  | |
|  | **Case** | |  | | | | | | | | | | | | |  | | | | | | | | |  | | |
|  | 1 | | {A,B,C} | | | | | | | | | | | | | Profit:(1,2,5) | | | | | | | | | 5 | | |
|  |  | |  | | | | | | | | | | | | | Weight: (2,3,4) | | | | | | | | |  | | |
|  | 2 | | {A,B,C,D,E,F,G} | | | | | | | | | | | | | Profit:(10,5,15,7,6,18,3) | | | | | | | | | 15 | | |
|  |  | |  | | | | | | | | | | | | | Weight: (2,3,5,7,1,4,1) | | | | | | | | |  | | |
|  | 3 | | {A,B,C,D,E,F,G} | | | | | | | | | | | | | A:(12,4),B:(10,6), | | | | | | | | | 18 | | |
|  |  | |  | | | | | | | | | | | | | C:(8,5),D:(11,7), | | | | | | | | |  | | |
|  |  | |  | | | | | | | | | | | | | E:(14,3),F:(7,1), G:(9,6) | | | | | | | | |  | | |
|  | **4.3** | Suppose you want to schedule N activities in a Seminar | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 54 | |
|  | Hall. Start time and Finish time of activities are given by | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | pair of (si,fi) for ith activity. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | Implement the program to maximize the utilization of | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | Seminar Hall. (Maximum activities should be selected.) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  | **Test** | | | | | **Number of** | | | | | | | | **(si,fi)** | | | | | | | | | | | | | |  | |
|  | **Case** | | | | | **activities** | | | | | | | |  | | | | | | | | | | | | | |
|  |  | | | | | **(N)** | | | | | | | |  | | | | | | | | | | | | | |
|  | 1 | | | | | 9 | | | | | | | | (1,2), 1,3),(1,4),(2,5),(3,7), | | | | | | | | | | | | | |
|  |  | | | | |  | | | | | | | | (4,9), (5,6), (6,8), (7,9) | | | | | | | | | | | | | |
|  | 2 | | | | | 11 | | | | | | | | (1,4),(3,5),(0,6),(3,8),(5,7), | | | | | | | | | | | | | |
|  |  | | | | |  | | | | | | | | (5,9), (6,10), (8,12),(8,11) | | | | | | | | | | | | | |
|  |  | | | | |  | | | | | | | | (12,14), (2,13) | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **5.** | **Dynamic Programming** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **5.1** | Implement a program which has BNMCOEF() function  that takes two parameters n and k and returns the value of | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 58 | |
|  |  | Binomial Coefficient C(n, k). Compare the dynamic programming implementation with recursive implementation of BNMCOEF(). (In output, entire table should be displayed.) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | | | | | | | | **Test**  **Case** | | | | | **n** | | | | | **k** | | | |  | | | | | | | |
| 1 | | | | | 5 | | | | | 2 | | | |
| 2 | | | | | 11 | | | | | 6 | | | |
| 3 | | | | | 12 | | | | | 5 | | | |
|  | **5.2** | Implement the program 4.2 using Dynamic Programing.  Compare Greedy and Dynamic approach. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 60 | |
|  | **5.3** | Given a chain < A1, A2,…,An> of n matrices, where for i=1,2,…,n matrix Ai with dimensions. Implement the program to fully parenthesize the product A1,A2,…,An in a way that minimizes the number of scalar multiplications. Also calculate the number of scalar multiplications for all possible combinations of matrices. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 62 | |
|  | **Test**  **Case** | | | | | | **n** | | | | | **Matrices with dimensions** | | | | | | | | | | | | | | | |  | |
| 1 | | | | | | 3 | | | | | A1: 3\*5, A2: 5\*6, A3: 6\*4 | | | | | | | | | | | | | | | |
| 2 | | | | | | 6 | | | | | A1: 30\*35, A2: 35\*15, A3: 15\*5, A4:  5\*10, A5: 10\*20, A6: 20\*25 | | | | | | | | | | | | | | | |
|  | **5.4** | Implement a program to print the longest common subsequence for the following strings: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 65 | |
|  | | **Test**  **Case** | | | | | | | | | **String1** | | | | | | | **String2** | | | | | | | |  | | | |
| 1 | | | | | | | | | ABCDAB | | | | | | | BDCABA | | | | | | | |
| 2 | | | | | | | | | EXPONENTIAL | | | | | | | POLYNOMIAL | | | | | | | |
| 3 | | | | | | | | | LOGARITHM | | | | | | | ALGORITHM | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **6.** | **Graph** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **6.1** | Write a program to detect cycles in an directed graph. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 67 | |
|  | **6.2** | From a given vertex in a weighted graph, implement a program to find shortest paths to other vertices using Dijkstra’s algorithm. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 70 | |
|  | **Test**  **Case** | | | | **Adjacency Matrix of graph** | | | | | | | | | | | | | | | | | | | **Start**  **Vertex** | | | |  | |
|  |  |  | 1 | | | |  | | | | | | | | | | | | | | | | | | | 1 | | | |  | |  | |
| 2 | | | |  | | | | | | | | | | | | | | | | | | | 3 | | | |
|  | **6.3** | Find Minimum Cost spanning tree of a given undirected  graph using Prim’s algorithm. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 73 | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **7.** | **Backtracking** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **7.1** | Implement a program to print all permutations of a given string. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 76 | |
|  | | | | | | | | | **Test**  **Case** | | | | | | **String** | | | | | |  | | | | | | | | |
| 1 | | | | | | ACT | | | | | |
| 2 | | | | | | NOTE | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **8.** | **String Matching Algorithm** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **8.1** | Suppose you are given a source string S[0 ..n − 1] of length n, consisting of symbols a and b. Suppose that you are given a pattern string P[0 ..m − 1] of length m < n, consisting of symbols a, b, and \*, representing a pattern to be found in string S. The symbol \* is a “wild card” symbol, which matches a single symbol, either a or b. The other symbols must match exactly. The problem is to output a sorted list M of valid “match positions”, which are positions j in S such that pattern P matches the substring S [j..j + |P|− 1]. For example, if S = ababbab and P = ab\*, then the output M should be [0, 2]. Implement a  straightforward, naive algorithm to solve the problem. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 78 | |
|  | **8.2** | Implement Rabin karp algorithm and test it on the | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 81 | |
|  |  | following test cases: | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |
|  | **Test Case** | | | **String** | | | | | | | | | | | | | | | | | | | **Pattern** | | | |  | |
| 1 | | | 2359023141526739921 | | | | | | | | | | | | | | | | | | | 31415  q=13 | | | |
| 2 | | | ABAAABCDBBABCDDEBCABC | | | | | | | | | | | | | | | | | | | ABC  q=101 | | | |

**PRACTICAL – 1**

**AIM: Implement and analyse algorithms given below**

1. **FACTORIAL (Iterative and Recursive)**

* **Iterative**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int main(){**

**cout<<"\nITERATIVE\n\n";**

**long int n,fact=1,cnt=0;**

**cout<<"Enter number:";**

**cin>>n;**

**for(int i=1;i<=n;i++)**

**{**

**fact=fact\*i;**

**cnt++;**

**}**

**cout<<"\nCounter : "<<cnt;**

**cout<<"\nFactorial : "<<fact;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**}**

**OUTPUT**



**ANALYSIS TABLE**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT (Counter)** |
| **37** | **37** |
| **6** | **6** |
| **14** | **14** |
| **49** | **49** |
| **28** | **28** |
| **10** | **10** |

**GRAPH**

* **Recursive**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int cnt=0;**

**long int factfunc(long int n)**

**{**

**if(n==0)**

**return 1;**

**else**

**{**

**cnt++;**

**return factfunc(n-1)\*n;**

**}**

**}**

**int main(){**

**cout<<"\nRECURSIVE\n\n";**

**long int n,fact=1;**

**cout<<"Enter number:";**

**cin>>n;**

**fact=factfunc(n);**

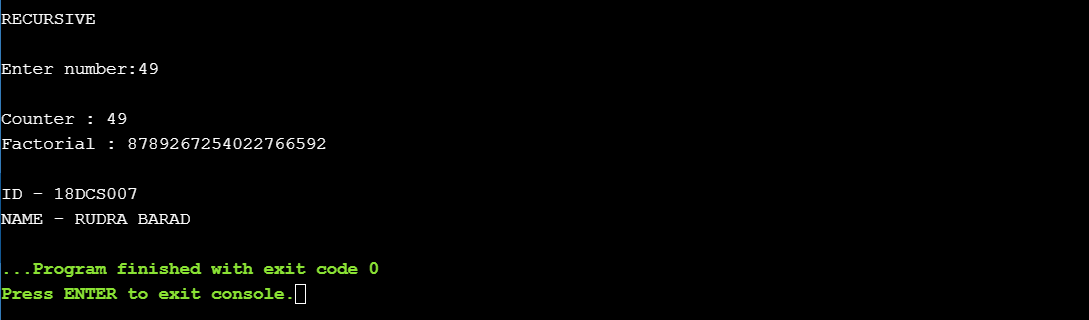
**cout<<"\nCounter : "<<cnt;**

**cout<<"\nFactorial : "<<fact;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**}**

**OUTPUT**



**ANALYSIS TABLE**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT (Counter)** |
| **7** | **7** |
| **46** | **46** |
| **31** | **31** |
| **16** | **16** |
| **49** | **49** |
| **24** | **24** |

**GRAPH**

**CONCLUSION**

In this practical I implemented the factorial of given number by both iterative and recursive method. I learned to analyse the algorithm by plotting the graph.

1. **EUCLIDEAN ALGORITHM**

**PROGRAM CODE**

**#include <iostream>**

**using namespace std;**

**int counter=0;**

**int gcd(int a, int b) {**

**if (b == 0)**

**return a;**

**counter++;**

**return gcd(b, a % b);**

**}**

**int main() {**

**int a , b,gcdno;**

**cout<<"Enter the value of a : ";**

**cin>>a;**

**cout<<"Enter the value of b : ";**

**cin>>b;**

**gcdno=gcd(a, b);**

**cout<<"GCD : "<<gcdno<<endl;**

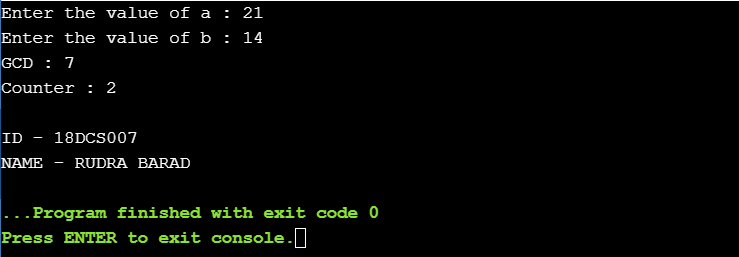
**cout<<"Counter : "<<counter;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**}**

**OUTPUT**



**ANALYSIS TABLE**

|  |  |  |  |
| --- | --- | --- | --- |
| **NUMBER 1** | **NUMBER 2** | **DIFFERENCE** | **OUTPUT** |
| **25** | **26** | **1** | **3** |
| **35** | **37** | **2** | **4** |
| **65** | **68** | **3** | **5** |
| **44** | **40** | **4** | **2** |
| **50** | **56** | **6** | **4** |
| **17** | **10** | **7** | **4** |
| **31** | **19** | **12** | **6** |
| **37** | **23** | **14** | **6** |

**GRAPH**

**CONCLUSION**

In this practical I implemented and analysed algorithms by performing practical to find GCD. I learned to analyse the algorithm by plotting the graph.

1. **Matrix Addition & Matrix Multiplication (Iterative)**

* **MATRIX ADDITION**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**int m, n, c, d, first[10][10], second[10][10], sum[10][10],counter=0;**

**cout<<"\nEnter the number of rows and columns of matrix : ";**

**cin>>m>>n;**

**cout<<"\nEnter the elements of first matrix : \n";**

**for (c = 0; c < m; c++)**

**for (d = 0; d < n; d++)**

**cin>>first[c][d];**

**cout<<"\nEnter the elements of second matrix : \n";**

**for (c = 0; c < m; c++)**

**for (d = 0 ; d < n; d++)**

**cin>>second[c][d];**

**cout<<"\nSum of entered matrices :-\n";**

**for (c = 0; c < m; c++) {**

**for (d = 0 ; d < n; d++) {**

**counter++;**

**sum[c][d] = first[c][d] + second[c][d];**

**cout<<sum[c][d]<<" ";**

**}**

**cout<<endl;**

**}**

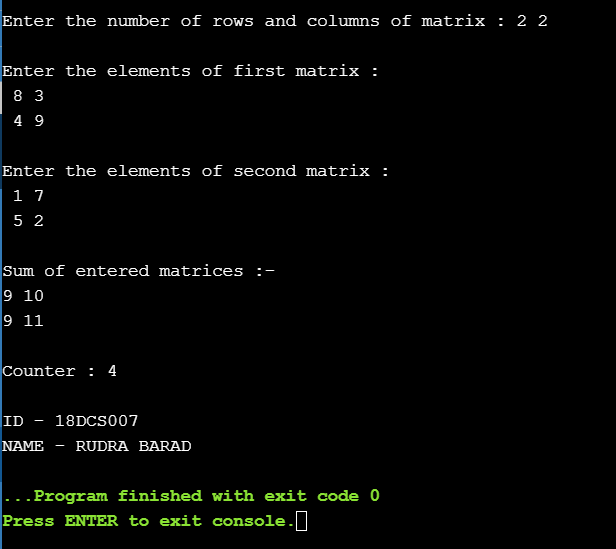
**cout<<"\nCounter : "<<counter;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**}**

**OUTPUT**



**ANALYSIS TABLE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MATRIX A** | | **MATRIX B** | | **COUNTER** |
| **ROW A** | **COLUMN A** | **ROW B** | **COLUMN B** |
| **2** | **2** | **2** | **2** | **4** |
| **3** | **2** | **3** | **2** | **6** |
| **3** | **3** | **3** | **3** | **9** |
| **4** | **3** | **4** | **3** | **12** |
| **5** | **5** | **5** | **5** | **25** |

**GRAPH**

* **MATRIX MULTIPLICATION**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int main(){**

**int a[10][10], b[10][10], result[10][10], r1, c1, r2, c2, i, j, k,counter=0;**

**cout<<"\nEnter the number of rows and columns of first matrix : ";**

**cin>>r1>>c1;**

**cout<<"\nEnter the number of rows and columns of second matrix : ";**

**cin>>r2>>c2;**

**while (c1 != r2)**

**{**

**cout<<"Error! column of first matrix not equal to row of second.\n\n";**

**cout<<"\nEnter the number of rows and columns of first matrix : ";**

**cin>>r1>>c1;**

**cout<<"\nEnter the number of rows and columns of second matrix : ";**

**cin>>r2>>c2;**

**}**

**for(i=0; i<r1; ++i)**

**for(j=0; j<c1; ++j)**

**{**

**cout<<"Enter elements for position "<<i+1<<" "<< j+1<<" :";**

**cin>>a[i][j];**

**a[i][j]=i+j+1;**

**}**

**cout<<"\nEnter elements of matrix 2:\n";**

**for(i=0; i<r2; ++i)**

**for(j=0; j<c2; ++j)**

**{**

**cout<<"Enter elements for position "<<i+1<<" "<< j+1<<" :";**

**cin>>b[i][j];**

**b[i][j]=i+j+1;**

**}**

**for(i=0; i<r1; ++i)**

**for(j=0; j<c2; ++j)**

**{**

**result[i][j] = 0;**

**}**

**for(i=0; i<r1; ++i)**

**for(j=0; j<c2; ++j)**

**for(k=0; k<c1; ++k)**

**{**

**result[i][j]+=a[i][k]\*b[k][j];**

**counter++;**

**}**

**cout<<"\nOutput Matrix:\n";**

**for(i=0; i<r1; ++i)**

**for(j=0; j<c2; ++j)**

**{**

**cout<<result[i][j]<<" ";**

**if(j == c2-1)**

**cout<<"\n";**

**}**

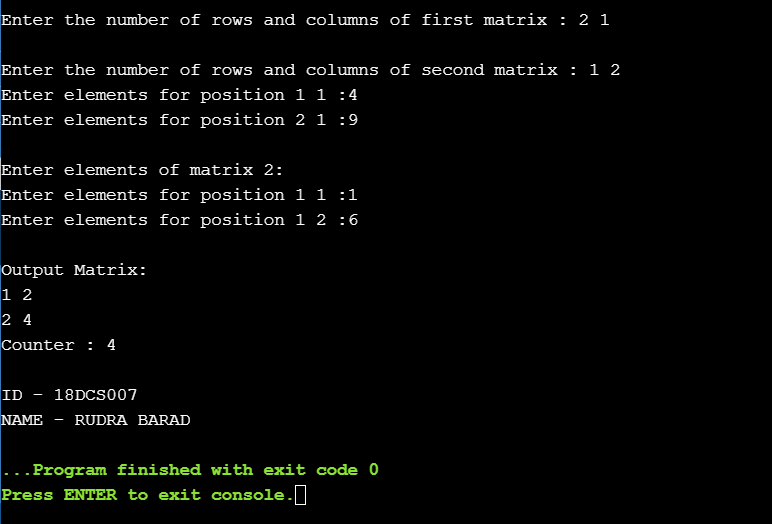
**cout<<"Counter : "<<counter;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**}**

**OUTPUT**



**ANALYSIS TABLE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MATRIX A** | | **MATRIX B** | | **COUNTER** |
| **ROW A** | **COLUMN A** | **ROW B** | **COLUMN B** |
| **2** | **1** | **1** | **2** | **4** |
| **4** | **2** | **2** | **2** | **8** |
| **4** | **3** | **4** | **3** | **12** |
| **6** | **2** | **2** | **3** | **18** |
| **6** | **1** | **1** | **4** | **24** |

**GRAPH**

**CONCLUSION**

Here we concluded by implementing and analysing the PROGRAM CODE of matrix addition and multiplication, the time complexity of matrix addition is O(n^2) and of matrix multiplication is O(n^) by analysing the graph of matrix addition is linear whereas of matrix multiplication is slightly curved in beginning and then linear.

1. RECURSIVE LINEAR SEARCH AND BINARY SEARCH

* **Linear Search**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int counter=0;**

**int RecursiveLS(int arr[], int value, int index, int n)**

**{**

**counter++;**

**int pos = 0;**

**if(index >= n)**

**return 0;**

**else if (arr[index] == value)**

**{**

**pos = index + 1;**

**return pos;**

**}**

**else**

**return RecursiveLS(arr, value, index+1, n);**

**return pos;**

**}**

**int main()**

**{**

**int n, value, pos, m = 0, arr[100];**

**cout<<"\nEnter the total elements in the array : ";**

**cin>>n;**

**cout<<"\nEnter the array elements : \n";**

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

**cin>>arr[i];**

**cout<<"\nEnter the element to search :";**

**cin>>value;**

**pos = RecursiveLS(arr, value, 0, n);**

**if (pos != 0)**

**cout<<"\nElement found at pos "<< pos;**

**else**

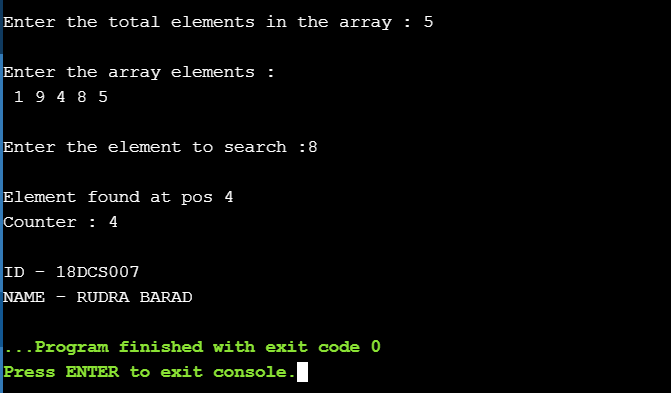
**cout<<"\nElement not found";**

**cout<<"\nCounter : "<<counter;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**OUTPUT**



**ANALYSIS TABLE (Best Case)**

|  |  |  |
| --- | --- | --- |
| **TOTAL ELENEMTS** | **POSITION OF ELEMENT TO BE SEARCHED** | **COUNTER** |
| **5** | **1** | **1** |
| **7** | **1** | **1** |
| **10** | **1** | **1** |
| **15** | **1** | **1** |
| **20** | **1** | **1** |

**ANALYSIS TABLE (Worst Case)**

|  |  |  |
| --- | --- | --- |
| **TOTAL ELENEMTS** | **POSITION OF ELEMENT TO BE SEARCHED** | **COUNTER** |
| **5** | **5** | **5** |
| **7** | **7** | **7** |
| **10** | **10** | **10** |
| **15** | **15** | **15** |
| **20** | **20** | **20** |

**GRAPH**

* **Binary Search**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int counter=0;**

**int binarySearch(int A[], int low, int high, int x)**

**{**

**counter++;**

**if (low > high)**

**return -1;**

**int mid = (low + high)/2;**

**// int mid = low + (high - low)/2;**

**if (x == A[mid])**

**return mid;**

**else if (x < A[mid])**

**return binarySearch(A, low, mid - 1, x);**

**else**

**return binarySearch(A, mid + 1, high, x); }**

**int main()**

**{**

**int nums,target;**

**cout<<"\nEnter the total elements in the array : ";**

**cin>>nums;**

**int arr[nums];**

**cout<<"\nEnter the array elements : \n";**

**for (int i = 0; i < nums; i++)**

**cin>>arr[i];**

**cout<<"\nEnter the element to search ";**

**cin>>target;**

**int n = sizeof(arr)/sizeof(arr[0]);**

**int low = 0, high = n - 1;**

**int index = binarySearch(arr, low, high, target);**

**if (index != -1)**

**cout<<"\nElement found at index "<<index;**

**else**

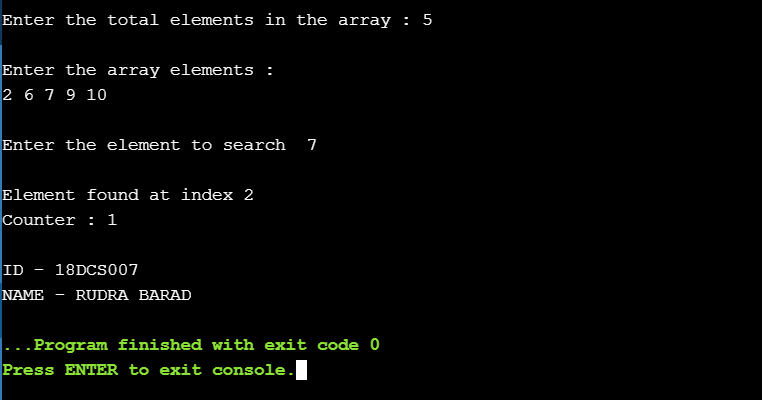
**cout<<"\nElement not found in the array";**

**cout<<"\nCounter : "<<counter;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**OUTPUT**



**ANALYSIS TABLE (Best Case)**

|  |  |  |
| --- | --- | --- |
| **TOTAL ELENEMTS** | **POSITION OF ELEMENT TO BE SEARCHED** | **COUNTER** |
| **4** | **2** | **1** |
| **9** | **5** | **1** |
| **13** | **7** | **1** |
| **15** | **8** | **1** |
| **18** | **9** | **1** |

**ANALYSIS TABLE (Worst Case)**

|  |  |  |
| --- | --- | --- |
| **TOTAL ELENEMTS** | **POSITION OF ELEMENT TO BE SEARCHED** | **COUNTER** |
| **5** | **5** | **3** |
| **10** | **10** | **4** |
| **33** | **33** | **6** |
| **90** | **90** | **7** |
| **155** | **155** | **8** |

**GRAPH**

**CONCLUSION**

Here we concluded by implementing and analysing the PROGRAM CODE of linear and binary search the time complexity of linear search best case is O(1) and worst case is O(n) whereas in binary search the best case is O(1) and the worst case is also O(logn) by analysing the graph of linear search in best case is constant and of worst case is linear whereas that of binary search the graph of best case is constant and of worst case is non-linear.

1. **Find a subset of a given set S = {s1, s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int counter=0;**

**bool isSubsetSum(int set[], int n, int sum)**

**{**

**counter++;**

**if (sum == 0)**

**return true;**

**if (n == 0 && sum != 0)**

**return false;**

**if (set[n-1] > sum)**

**return isSubsetSum(set, n-1, sum);**

**return isSubsetSum(set, n-1, sum) ||**

**isSubsetSum(set, n-1, sum-set[n-1]);**

**}**

**int main()**

**{**

**int set[] = {1,2,7,8,3,5,9,99};**

**int sum = 14;**

**cout<<"\nSum is : "<<sum;**

**int n = sizeof(set)/sizeof(set[0]);**

**if (isSubsetSum(set, n, sum) == true)**

**cout<<"\nFound a subset with given sum\n";**

**else**

**cout<<"No subset with given sum";**

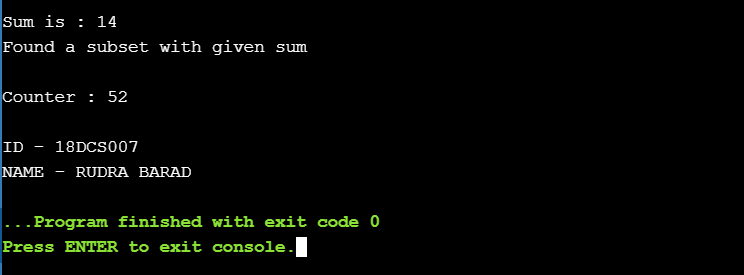
**cout<<"\nCounter : "<<counter;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**}**

**OUTPUT**



**ANALYSIS TABLE**

|  |  |
| --- | --- |
| **TOTAL NO OF ELEMENTS** | **COUNTER** |
| **2** | **7** |
| **3** | **12** |
| **8** | **52** |
| **10** | **68** |
| **14** | **128** |

**GRAPH**

**CONCLUSION**

Here we concluded by implementing and analysing the PROGRAM CODE of subset sum the time complexity of subset sum is O(2^n) and by analysing the graph is non-linear.

**PRACTICAL – 2**

**AIM: Implement and analyse algorithms given below. (Compare them)**

**2.1 BUBBLE SORT**

**PROGRAM CODE**

**#include <iostream>**

**using namespace std;**

**int counter = 0;**

**void bubbleSort(int arr[], int size){**

**int temp;**

**for(int i=0; i<size; i++){**

**for(int j=0; j<size-i-1; j++){**

**if(arr[j] > arr[j+1]){**

**temp = arr[j];**

**arr[j] = arr[j+1];**

**arr[j+1] = temp;**

**counter++;**

**}**

**}**

**}**

**}**

**int main(){**

**int n;**

**cout<<"Please enter the size of array : ";**

**cin>>n;**

**int arr[n];**

**cout<<endl<<"Please enter the array elements : ";**

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

**{**

**cin>>arr[i];**

**}**

**bubbleSort(arr, n);**

**cout<<endl<<"The sorted array is : ";**

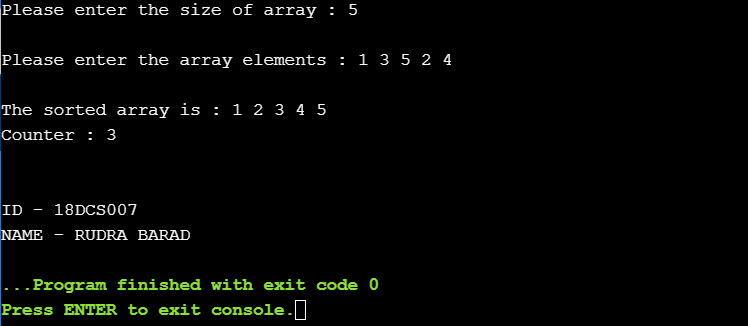
**for(int i=0;i<n;i++)**

**cout<<arr[i]<<" ";**

**cout<<endl<<"Counter : "<<counter<<endl;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**OUTPUT**



**2.2 SELECTION SORT**

**PROGRAM CODE**

**#include <iostream>**

**using namespace std;**

**int counter=0;**

**void selectionSort(int arr[], int size){**

**int temp;**

**int min\_index = 0;**

**for(int i=0;i<size;i++){**

**min\_index=i;**

**for(int j=i; j<size; j++){**

**if(arr[min\_index] > arr[j]){**

**min\_index = j;**

**}**

**}**

**temp = arr[i];**

**arr[i] = arr[min\_index];**

**arr[min\_index] = temp;**

**counter++;**

**}**

**}**

**int main(){**

**int n;**

**cout<<"\nEnter size of array : ";**

**cin>>n;**

**int arr[n];**

**cout<<"Enter array elements : ";**

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

**cin>>arr[i];**

**selectionSort(arr, n);**

**cout<<endl<<"Sorted array is : ";**

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

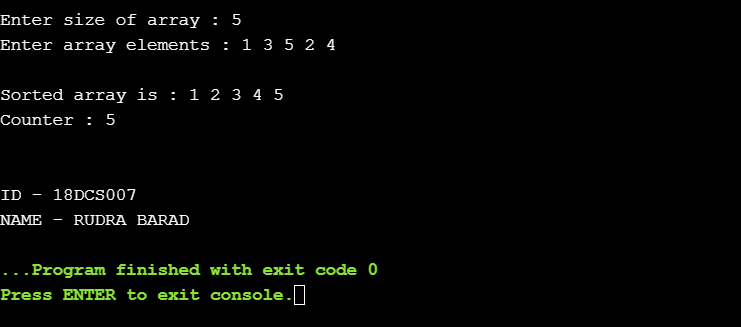
**cout<<arr[i]<<" ";**

**cout<<endl<<"Counter : "<<counter<<endl;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**}**

**OUTPUT**



**2.3 INSERTION SORT**

**PROGRAM CODE**

**#include <iostream>**

**using namespace std;**

**int counter=0;**

**void insertionSort(int arr[], int size){**

**int temp;**

**for(int i=0;i<size-1;i++){**

**for(int j=i+1; j>0; j--){**

**if(arr[j] < arr[j-1]){**

**temp = arr[j];**

**arr[j] = arr[j-1];**

**arr[j-1] = temp;**

**counter ++ ;**

**}**

**}**

**}**

**}**

**int main(){**

**int n;**

**cout<<"\nEnter the size of the array : ";**

**cin>>n;**

**int arr[n];**

**cout<<endl<<"Please enter the array elements : ";**

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

**cin>>arr[i];**

**insertionSort(arr, n);**

**cout<<endl<<"The sorted array is : "<<endl;**

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

**cout<<arr[i]<<" ";**

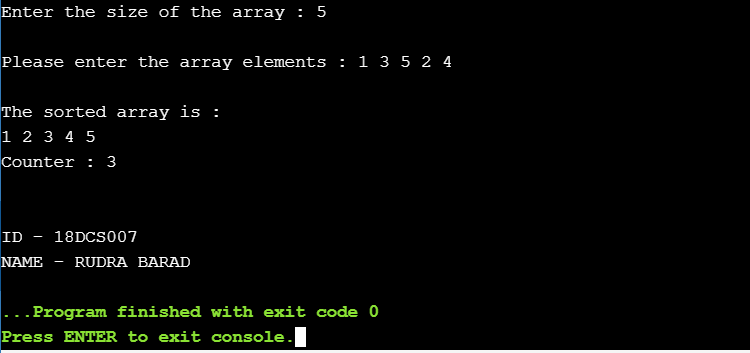
**cout<<endl<<"Counter : "<<counter<<endl;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**}**

**OUTPUT**



**ANALYSIS TABLE**

|  |  |  |  |
| --- | --- | --- | --- |
| INPUT SIZE | BUBBLE SORT | INSERTION SORT | SELECTION SORT |
| 4 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 |
| 6 | 5 | 5 | 5 |
| 7 | 13 | 13 | 9 |
| 8 | 15 | 15 | 9 |
| 9 | 25 | 25 | 10 |
| 10 | 27 | 27 | 12 |
| 11 | 55 | 55 | 30 |
| 12 | 66 | 66 | 36 |
| 13 | 78 | 78 | 42 |

**GRAPH**

**CONCLUSION**

In this practical we studied the comparison between selection sort, bubble sort and insertion sort. We also studied their worst, best and average case scenarios and accordingly we plotted the graph.

**PRACTICAL – 3**

**DIVIDE AND CONQUER STRATEGY**

**3.1 AIM: Implement and perform analysis of worst case of Merge. Sort and Quick sort. Compare both algorithms.**

* **MERGE SORT**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int counter=0;**

**void merge(int arr[], int left, int middle, int right){**

**int sizeLeft = middle - left + 1;**

**int sizeRight = right - middle;**

**int leftArray[sizeLeft], rightArray[sizeRight];**

**for(int i=0;i<sizeLeft;i++){**

**leftArray[i] = arr[left + i];**

**}**

**for(int i=0;i<sizeRight;i++){**

**rightArray[i] = arr[middle+1+i];**

**}**

**int i=0;**

**int j=0;**

**int k=left;**

**while(i<sizeLeft && j< sizeRight){**

**if(leftArray[i] <= rightArray[j]){**

**arr[k] = leftArray[i];**

**i++;**

**} else {**

**arr[k] = rightArray[j];**

**j++;**

**}**

**k++;**

**}**

**while(i<sizeLeft){**

**arr[k] = leftArray[i];**

**i++;**

**k++;**

**}**

**while(i<sizeRight){**

**arr[k] = rightArray[j];**

**j++;**

**k++;**

**}**

**}**

**void mergeSort(int arr[], int left,int right){**

**counter++;**

**if (right > left){**

**int middle = (left + right) / 2;**

**mergeSort(arr, left, middle);**

**mergeSort(arr, middle+1, right);**

**merge(arr, left, middle, right);**

**}**

**}**

**int main(){**

**int size;**

**cout << "Enter the size of the array: ";**

**cin >> size;**

**int arr[size];**

**cout << "Enter the values in the array: "<< endl;**

**for(int i=0;i<size;i++){**

**cin >> arr[i];**

**}**

**mergeSort(arr,0,size);**

**cout << "The sorted array will be: "<< endl;**

**for(int i=0;i<size;i++){**

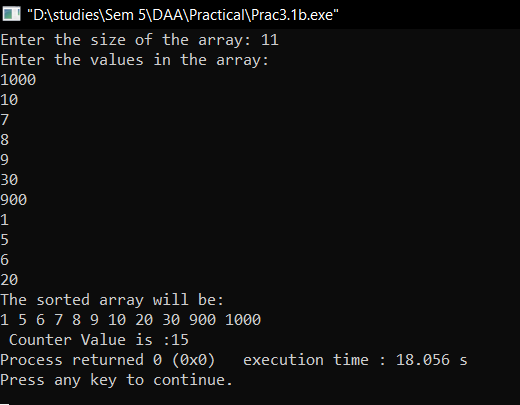
**cout << arr[i] << " ";**

**}**

**cout<<"\n Value of counter is : "<<counter;**

**}**

**OUTPUT**

****

* **QUICK SORT**

**PROGRAM CODE**

**#include <iostream>**

**using namespace std;**

**int counter=0;**

**int partition(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(int arr[], int low, int high){**

**counter++;**

**if(low < high){**

**int pivotIndex = partition(arr,low,high);**

**quickSort(arr,low,pivotIndex-1);**

**quickSort(arr,pivotIndex+1,high);**

**}**

**}**

**int main(){**

**int size;**

**cout << "Enter the size of the array: ";**

**cin >> size;**

**int arr[size];**

**cout << "Enter the values in the array: "<< endl;**

**for(int i=0;i<size;i++){**

**cin >> arr[i];**

**}**

**quickSort(arr,0,size-1);**

**cout << "The sorted array will be: "<< endl;**

**for(int i=0;i<size;i++){**

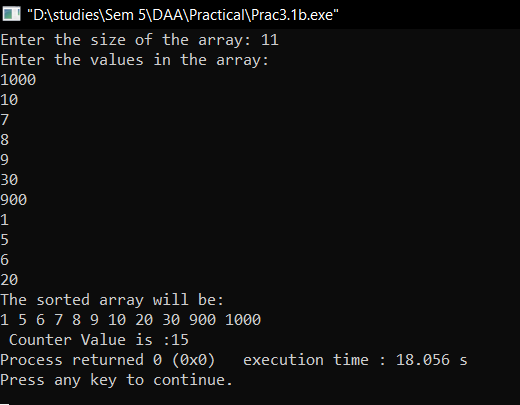
**cout << arr[i] << " ";**

**}**

**cout<<"\n Counter Value is :"<<counter;**

**}**

**OUTPUT**



**ANALYSIS TABLE**

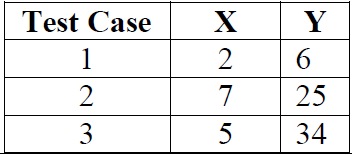
|  |  |  |
| --- | --- | --- |
| **No. Of Elements** | **Merge Sort** | **Quick Sort** |
| 6 | 13 | 11 |
| 7 | 15 | 13 |
| 8 | 17 | 15 |
| 10 | 21 | 19 |
| 13 | 27 | 25 |
| 14 | 29 | 27 |
| 16 | 33 | 31 |

**GRAPH**

**CONCLUSION**

**In this practical we have implemented the merge sort and quick sort algorithms using c++ PROGRAM CODEming language. We have analysed and compared both the algorithms based on their respective worst cases.**

**3.2 AIM: Implement the PROGRAM CODE to find X^Y using divide and conquer strategy and print the total number of multiplications required to find X^Y. Test the PROGRAM CODE for following test cases:**



**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int counter = 0;**

**double expo(double base, int power){**

**counter++;**

**if(power == 0){**

**return 1;**

**}**

**int temp = expo(base, power/2);**

**if(power % 2 == 0){**

**return temp \* temp;**

**} else if (power%2 ==1){**

**return base \* temp \* temp;**

**}**

**}**

**int main(){**

**double base;**

**int power;**

**cout << "Enter the base: ";**

**cin >> base;**

**cout << "Enter the power: ";**

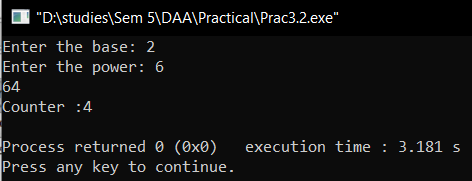
**cin >> power;**

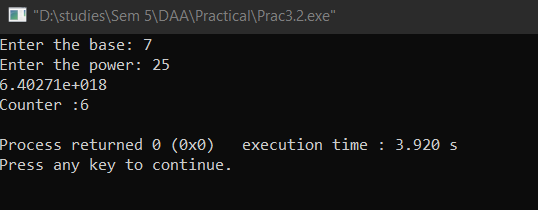
**cout << expo(base,power) << endl;**

**cout << counter << endl;**

**}**

**OUTPUT**

****

****

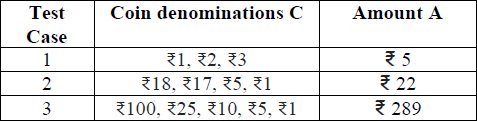
**CONCLUSION**

**In this practical we have learnt the basics of divide and conquer strategy. We have also tried to optimize the power of element problem using divide and conquer strategy.**

**PRACTICAL – 4**

**4.1 AIM: A cashier at any mall needs to give change of an amount to customers many times in a day. Cashier has multiple number of coins available with different denominations which is described by a set C. Implement the PROGRAM CODE for a cashier to find the minimum number of coins required to find a change of a particular amount A. Output should be the total number of coins required of given denominations.**

**Check the PROGRAM CODE for following test cases:**



**PROGRAM CODE**

**#include <bits/stdc++.h>**

**using namespace std;**

**void minFinder(int sum,int length,int coins[])**

**{**

**int ans[sum],j=0,i=0;**

**for(i=length-1;i>=0;i--)**

**{**

**while(sum>=coins[i])**

**{**

**sum=sum-coins[i];**

**ans[j]=coins[i];**

**j++;**

**}**

**}**

**cout<<"\nTotal coins required are : "<<j<<"\nThey are : ";**

**for(i=0;i<j;i++)**

**cout<<ans[i]<<" ";**

**cout<<"\n";**

**}**

**int main()**

**{**

**int length,i,sum;**

**cout<<"\nEnter the total types of coins : ";**

**cin>>length;**

**int coins[length];**

**cout<<"Enter the denomination of coins : ";**

**for(i=0;i<length;i++)**

**cin>>coins[i];**

**int n = sizeof(coins)/sizeof(coins[0]);**

**sort(coins,coins+n);**

**cout<<"Enter the final sum of coins : ";**

**cin>>sum;**

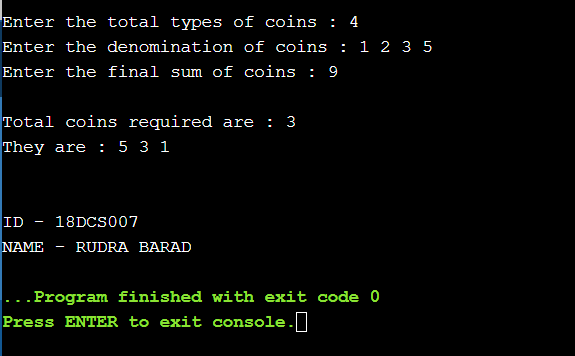
**minFinder(sum,length,coins);**

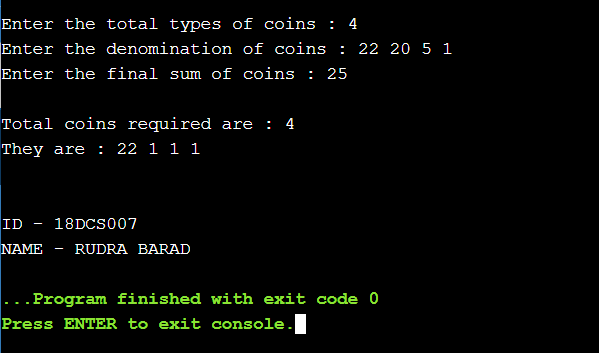
**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

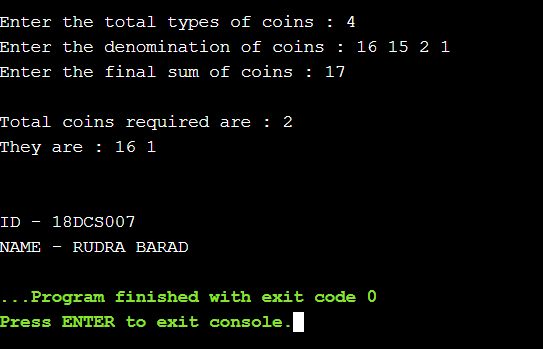
**return 0;**

**}**

**OUTPUT**







**OBSERVATION:**

By performing the above practical it observed that it didn’t gave optimum solution for every test cases.

Analysis of different test cases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Denomination | Amount | No. of coins | Optimum Sol. |
| 1 | Rs 1,2,3,5 | Rs 9 | 3 | YES |
| 2 | Rs 22,20,5,1 | Rs 25 | 4 | NO |
| 3 | Rs 16,15,2,1 | Rs 17 | 2 | YES |

From the observation we can say that the solution for test case 2 is not optimal.

Obtained Solution: 22+1+1+1 = 25 (4 coins)

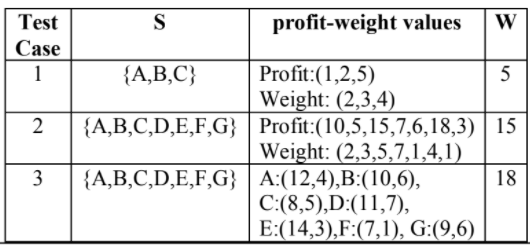
Optimal Solution: 20 + 5 = 25 (2 coins)

**CONCLUSION**

This problem is a variation of 'Coin Change Problem'. Worst case: When the only coin present is Rs. 1 Coin. So in that case, time complexity becomes O(A) where A is the amount to be paid.

**4.2 AIM: Let S be a collection of objects with profit-weight values. Implement the fractional Knapsack problem for S assuming we have a sack that can hold objects with total weight W.**

**Check the PROGRAM CODE for following test cases:**



**PROGRAM CODE**

**#include <iostream>**

**#include <bits/stdc++.h>**

**using namespace std;**

**typedef struct {**

**double v;**

**double w;**

**} Item;**

**void input(Item items[],int sizeOfItems) {**

**cout << "Enter total "<< sizeOfItems <<" Item's values and weight" <<**

**endl;**

**for(int i = 0; i < sizeOfItems; i++) {**

**cout << "Enter "<< i+1 << " V ";**

**cin >> items[i].v;**

**cout << "Enter "<< i+1 << " W ";**

**cin >> items[i].w;**

**}**

**}**

**void display(Item items[], int sizeOfItems) {**

**int i;**

**cout << "values: ";**

**for(i = 0; i < sizeOfItems; i++) {**

**cout << items[i].v << "\t";**

**}**

**cout << endl << "weight: ";**

**for (i = 0; i < sizeOfItems; i++) {**

**cout << items[i].w << "\t";**

**}**

**cout << endl;**

**}**

**bool compare(Item a, Item b) {**

**double r1 = (double)(a.v / a.w);**

**double r2 = (double)(b.v / b.w);**

**return r1 > r2;**

**}**

**double knapsack(Item items[], int sizeOfItems, int W) {**

**int i, j;**

**double totalValue = 0, totalWeight = 0;**

**cout<<"Profit per unit of weight :\n";**

**cout<<"Value Weight Profit\n";**

**for (int i = 0; i < sizeOfItems; i++)**

**{**

**cout << items[i].v << " " << items[i].w << " "**

**<< ((double)items[i].v / items[i].w) << endl;**

**}**

**sort(items, items+sizeOfItems, compare);**

**for(i=0; i<sizeOfItems; i++) {**

**if(totalWeight + items[i].w<= W) {**

**totalValue += items[i].v ;**

**totalWeight += items[i].w;**

**} else {**

**int wt = W-totalWeight;**

**totalValue += items[i].v\*((double)wt / items[i].w);**

**totalWeight += wt;**

**break;**

**}**

**}**

**cout << "Total weight in bag " << totalWeight<<endl;**

**return totalValue;**

**}**

**int main() {**

**int W,n;**

**cout<<"Enter total number of items :";**

**cin>>n;**

**Item items[n];**

**input(items, n);**

**cout << "Entered data \n";**

**display(items,n);**

**cout<< "Enter Knapsack weight \n";**

**cin >> W;**

**double mxVal = knapsack(items, n, W);**

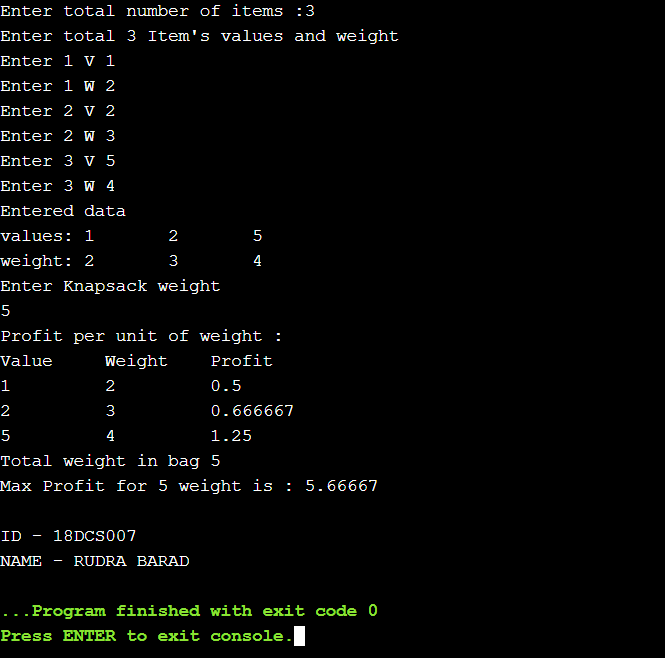
**cout << "Max Profit for "<< W <<" weight is : "<< mxVal;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

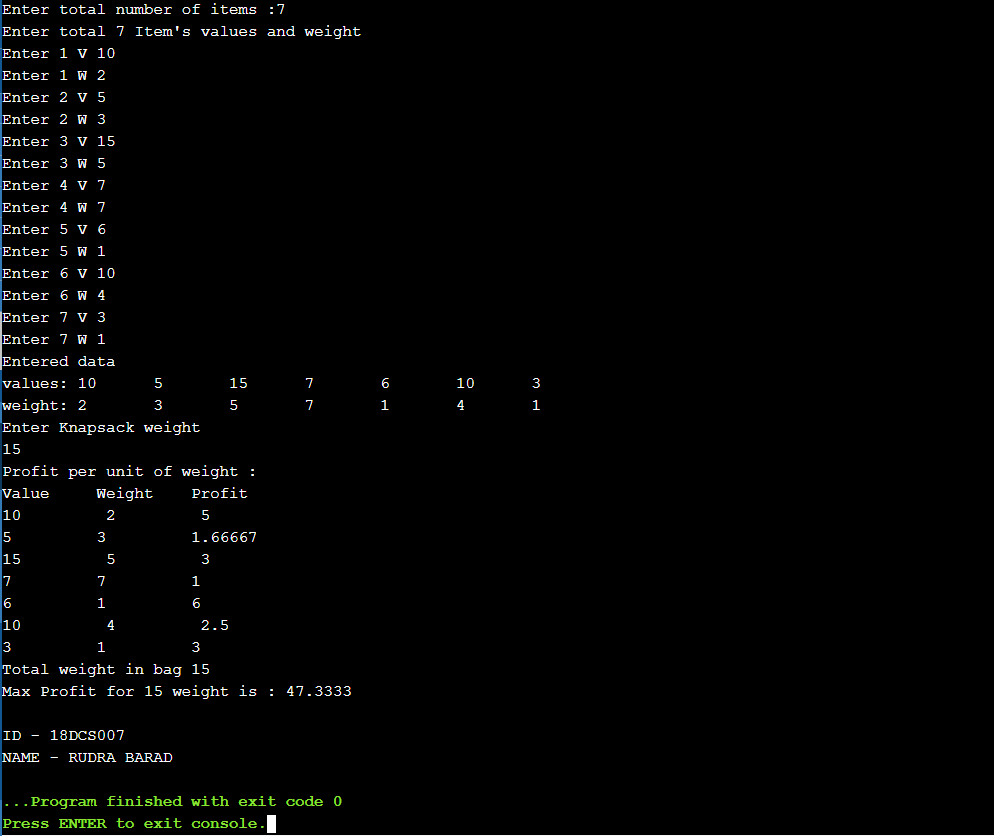
**return 0;**

**}**

**OUTPUT 1**



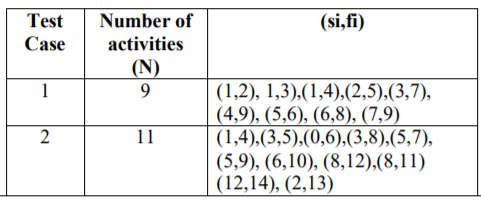
**OUTPUT 2**



**CONCLUSION**

In this practical we have tried to code a cpp PROGRAM CODE to solve the fractional knapsack problem. The problem is based on getting maximum profit items in the knapsack having fixed carry capacity and also to utilize maximum (whole, in fractional knapsack) carrying capacity of the knapsack. Here greedy approach is used to solve the problem.

**4.3 AIM: Suppose you want to schedule N activities in a SeminarHall. Start time and Finish time of activities are given bypair of (si,fi) for ith activity.Implement the PROGRAM CODE to maximize the utilization ofSeminar Hall. (Maximum activities should be selected.)**



**PROGRAM CODE**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

typedef struct {

int s;

int f;

} activ;

void input(activ arr[],int lng) {

cout << "Enter total "<< lng <<" Item's Start and Finish time :-\n\n";

for(int i = 0; i < lng; i++) {

cout << "Enter Start Time For Activity "<< i+1<<":" ;

cin >> arr[i].s;

cout << "Enter Finish Time For Activity "<< i+1<<":";

cin >> arr[i].f;

cout<<"\n";

}

}

void display(activ arr[], int lng) {

int i;

cout << "Start Time: ";

for(i = 0; i < lng; i++) {

cout << "\t"<< arr[i].s ;

}

cout << endl << "Finish Time: ";

for (i = 0; i < lng; i++) {

cout << "\t"<< arr[i].f ;

}

cout << endl;

}

bool compare(activ a, activ b) {

return a.f < b.f;

}

void MaxAct(activ arr[], int n)

{

sort(arr, arr+n, compare);

cout << "Following activities are selected :\n";

int i = 0;

cout << "(" << arr[i].s << ", " << arr[i].f << "), ";

for (int j = 1; j < n; j++)

{

if (arr[j].s >= arr[i].f)

{

cout << "(" << arr[j].s << ", "

<< arr[j].f << "), ";

i = j;

}

}

}

int main() {

int n;

cout<<"Enter total number of Activities :";

cin>>n;

activ arr[n];

input(arr, n);

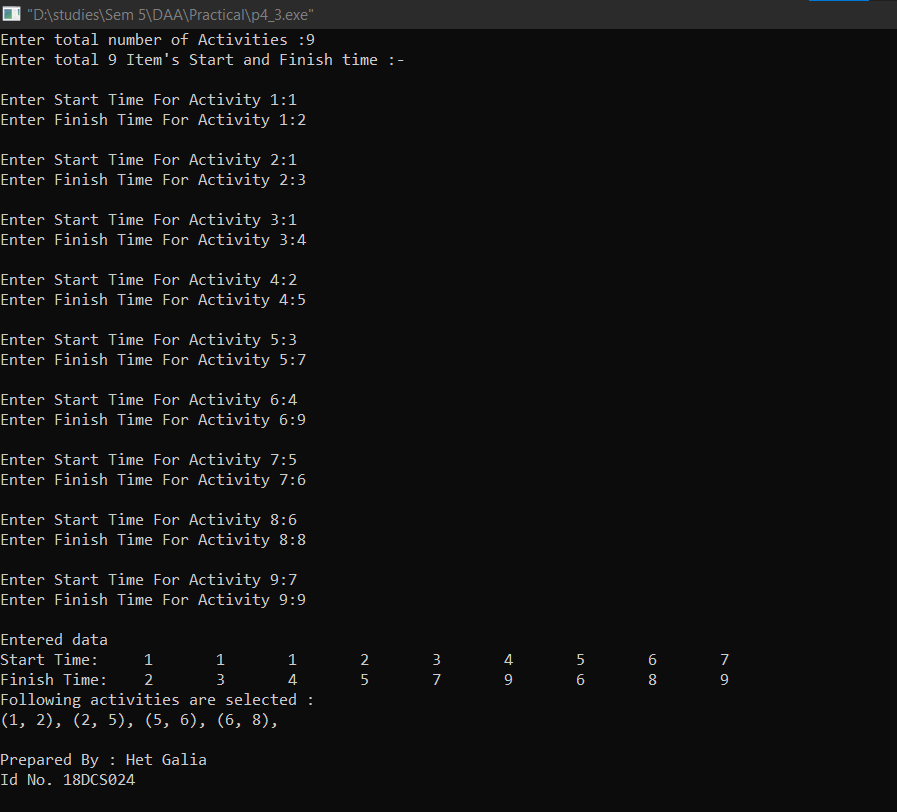
cout << "Entered data \n";

display(arr,n);

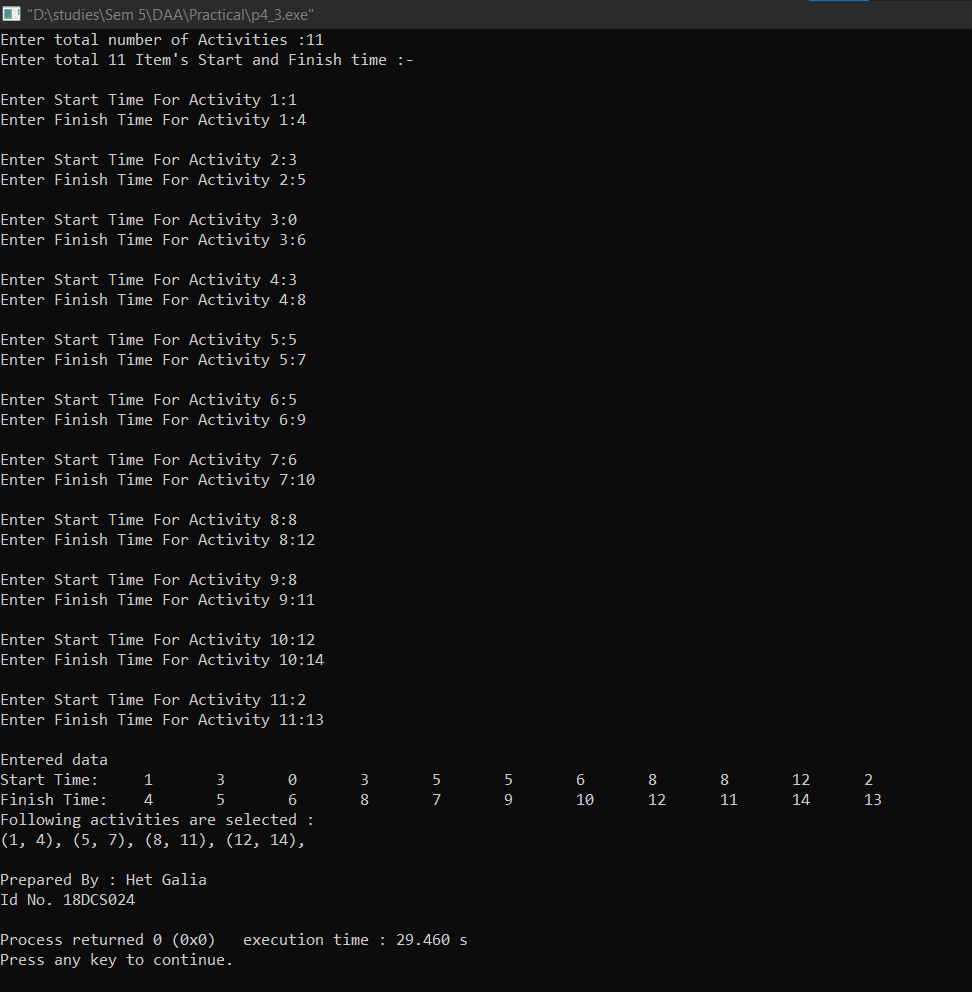
MaxAct(arr, n);

}

**OUTPUT 1**

****

**OUTPUT 2**

****

**CONCLUSION**

In this practical we have attempted to code a cpp PROGRAM CODE to solve the activity selection problem. The problem is based on getting maximum activities completed such that maximum amount of the schedule can be utilized .In the problem definition we are given to solve this problem for seminar hall. Here greedy approach is used to solve the problem which allows to complete maximum activities so that maximum schedule time of seminar hall can be used.

**PRATICAL - 5**

**DYNAMIC PROGRAM CODEMING**

**5.1 AIM: Implement a PROGRAM CODE which has BNMCOEF() function that takes two parameters n and k and returns the value of Binomial Coefficient C(n, k). Compare the dynamic PROGRAM CODEming implementation with recursive implementation of BNMCOEF(). (In output, entire table should be displayed.)**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **n** | **k** |
| 1 | 5 | 2 |
| 2 | 11 | 6 |
| 3 | 12 | 5 |

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int binomialCoeff(int n, int k)**

**{**

**// Base Cases**

**if (k == 0 || k == n)**

**return 1;**

**// Recur**

**return binomialCoeff(n - 1, k - 1) +**

**binomialCoeff(n - 1, k);**

**}**

**int main()**

**{**

**int n,k;**

**cout<<"...BINOMIAL CO-EFFICIENT...\n";**

**cout<<"Enter the value of n : ";**

**cin>>n;**

**cout<<"Enter the value of k : ";**

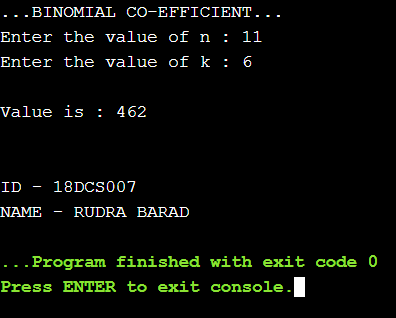
**cin>>k;**

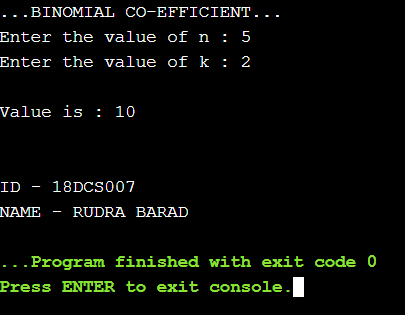
**cout<<"\nValue is : "<<binomialCoeff(n,k)<<endl;**

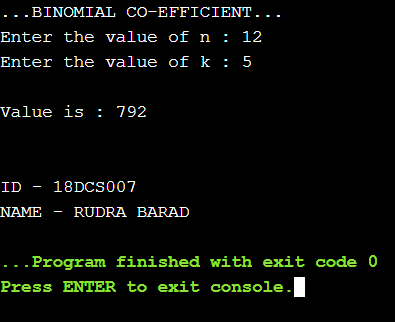
**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**OUTPUT**







**5.2 AIM: Implement the PROGRAM CODE 4.2 using Dynamic PROGRAM CODEming. Compare Greedy and Dynamic approach.**

**PROGRAM CODE**

**#include<iostream>**

**using namespace std;**

**int maximum(int x,int y)**

**{**

**if(x>y)**

**return x;**

**else**

**return y;**

**}**

**int knapsack(int bag\_capacity,int weight[],int profit[],int number)**

**{**

**int matrix[number+1][bag\_capacity+1];**

**for(int i=0;i<number+1;i++)**

**for(int j=0;j<bag\_capacity+1;j++)**

**{**

**if(i==0 || j==0)**

**matrix[i][j]=0;**

**else if (j>=weight[i-1])**

**matrix[i][j]=maximum(matrix[i-1][j],profit[i-1]+matrix[i-1][j-weight[i-1]]);**

**else**

**matrix[i][j]=matrix[i-1][j];**

**}**

**return matrix[number][bag\_capacity];**

**}**

**int main()**

**{**

**int number,bag\_capacity;**

**cout<<".....BINARY KNAPSACK PROBLEM.....";**

**cout<<"\nEnter the size of arrays : ";**

**cin>>number;**

**int weight[number],profit[number];**

**cout<<"\nEnter the weights :";**

**for(int i=0;i<number;i++)**

**cin>>weight[i];**

**cout<<"Enter the profits :";**

**for(int i=0;i<number;i++)**

**cin>>profit[i];**

**cout<<"Enter bag capacity : ";**

**cin>>bag\_capacity;**

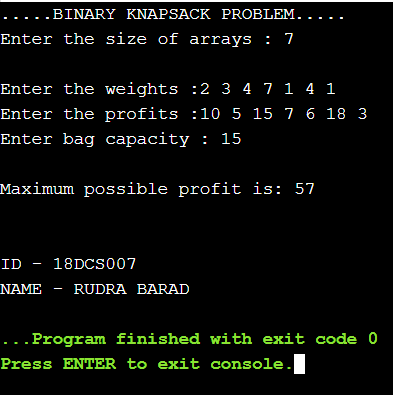
**cout<<"\nMaximum possible profit is: " << knapsack(bag\_capacity,weight,profit,number) <<endl;**

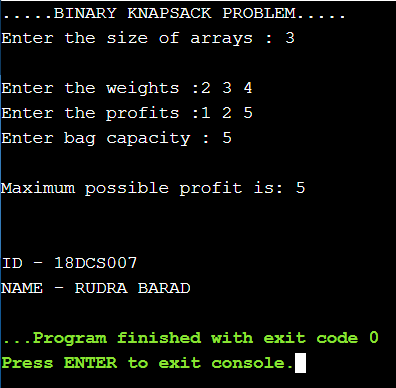
**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**}**

**OUTPUT**





**5.3 AIM: Given a chain < A1, A2,...,An> of n matrices, where for i=1,2,...,n matrix Ai with dimensions. Implement the PROGRAM CODE to fully parenthesize the product A1,A2,...,An in a way that minimizes the number of scalar multiplications. Also calculate the number of scalar multiplications for all possible combinations of matrices.**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **n** | **Matrices with dimensions** |
| 1 | 3 | A1:3\*5,A2:5\*6,A3:6\*4 |
| 2 | 6 | A1: 30\*35, A2: 35\*15, A3: 15\*5, A4: 5\*10, A5: 10\*20, A6: 20\*25 |

**PROGRAM CODE**

**#include<bits/stdc++.h>**

**using namespace std;**

**int MatrixMultiplication(int p[], int n)**

**{**

**int m[n][n];**

**int i, j, k, L, q;**

**for (i = 1; i < n; i++)**

**m[i][i] = 0;**

**// L is chain length.**

**for (L = 2; L < n; L++)**

**{**

**for (i = 1; i < n - L + 1; i++)**

**{**

**j = i + L - 1;**

**m[i][j] = INT\_MAX;**

**for (k = i; k <= j - 1; k++)**

**{**

**// q = cost/scalar multiplications**

**q = m[i][k] + m[k + 1][j] +**

**p[i - 1] \* p[k] \* p[j];**

**if (q < m[i][j])**

**m[i][j] = q;**

**}**

**}**

**}**

**return m[1][n - 1];**

**}**

**int main()**

**{**

**int n;**

**cout<<"...MATRIX CHAIN MULTIPLICATION...\n\n";**

**cout<<"Enter total number of dimension values : ";**

**cin>>n;**

**int arr[n];**

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

**{**

**cout<<"Enter P"<<i<<" : ";**

**cin>>arr[i];**

**}**

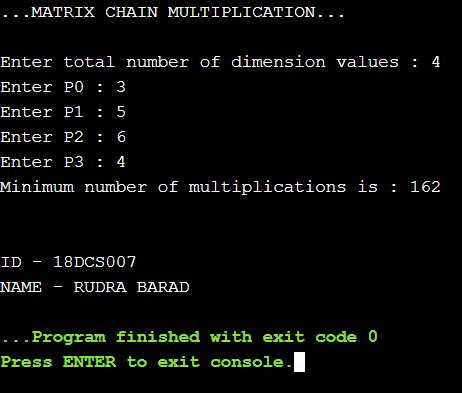
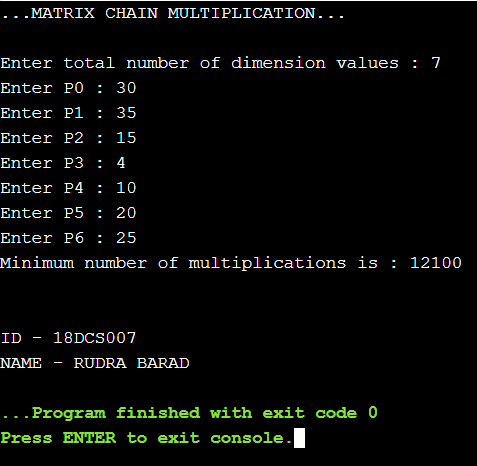
**int length=sizeof(arr)/sizeof(arr[0]);**

**cout << "Minimum number of multiplications is : "<<MatrixMultiplication(arr,length)<<endl;**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**}**

**OUTPUT**

**5.4 AIM: Implement a PROGRAM CODE to print the longest common subsequence for the following strings:**

**Check the PROGRAM CODE for following test cases:**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **String1** | **String2** |
| 1 | ABCDAB | BDCABA |
| 2 | EXPONENTIAL | POLYNOMIAL |
| 3 | LOGARITHM | ALGORITHM |

**PROGRAM CODE**

**#include<iostream>**

**#include<string.h>**

**using namespace std;**

**int max(int a, int b);**

**int lcs( char \*X, char \*Y, int m, int n )**

**{ if (m == 0 || n == 0)**

**return 0;**

**if (X[m-1] == Y[n-1])**

**return 1 + lcs(X, Y, m-1, n-1);**

**else**

**return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));} int max(int a, int b)**

**{**

**return (a > b)? a : b;**

**}**

**int main()**

**{ cout<<"...LONGEST COMMON SUBSEQUENCE..."<<endl;**

**char X[100],Y[100];**

**cout<<"Enter 1st string sequence : ";cin>>X;**

**cout<<"Enter 2nd string sequence : ";cin>>Y;**

**int m = strlen(X);int n = strlen(Y);**

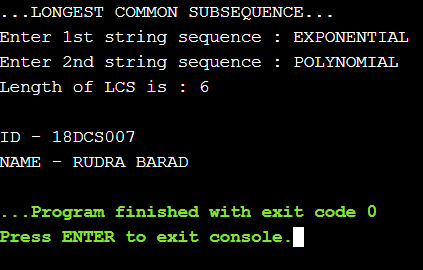
**cout<<"Length of LCS is : "<<lcs(X,Y,m,n);**

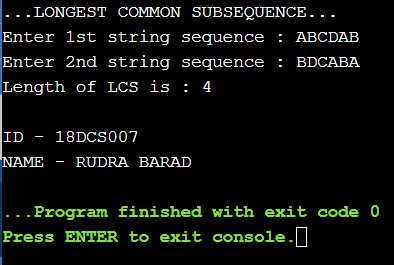
**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

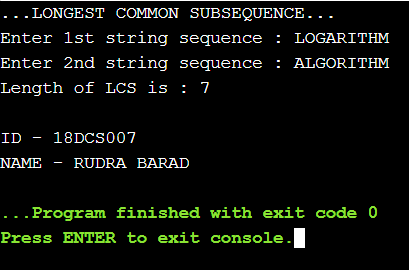
**return 0;**

**}**

**OUTPUT**







**CONCLUSION**

**Time Complexity of the above implementation is O(mn) where m and n is length of string1 and string2.And, it is observed to be much better than the worst-case time complexity of Naive Recursive implementation.**

**PRACTICAL - 6**

**GRAPH**

**6.1 AIM: Write a PROGRAM CODE to detect cycles in an directed graph.**

**PROGRAM CODE**

**#include<iostream>**

**#include <list>**

**#include <limits.h>**

**using namespace std;**

**class Graph**

**{**

**int V;**

**list<int> \*adj;**

**bool isCyclicUtil(int v, bool visited[], bool \*rs);**

**public:**

**Graph(int V);**

**void addEdge(int v, int w);**

**bool isCyclic();**

**};**

**Graph::Graph(int V)**

**{**

**this->V = V;**

**adj = new list<int>[V];**

**}**

**void Graph::addEdge(int v, int w)**

**{**

**adj[v].push\_back(w);**

**}**

**bool Graph::isCyclicUtil(int v, bool visited[], bool \*recStack)**

**{**

**if(visited[v] == false)**

**{**

**visited[v] = true;**

**recStack[v] = true;**

**list<int>::iterator i;**

**for(i = adj[v].begin(); i != adj[v].end(); ++i)**

**{**

**if ( !visited[\*i] && isCyclicUtil(\*i, visited, recStack) )**

**return true;**

**else if (recStack[\*i])**

**return true;**

**}**

**}**

**recStack[v] = false;**

**return false;**

**}**

**bool Graph::isCyclic()**

**{**

**bool \*visited = new bool[V];**

**bool \*recStack = new bool[V];**

**for(int i = 0; i < V; i++)**

**{**

**visited[i] = false;**

**recStack[i] = false;**

**}**

**for(int i = 0; i < V; i++)**

**if (isCyclicUtil(i, visited, recStack))**

**return true;**

**return false;**

**}**

**int main()**

**{**

**Graph g(4);**

**g.addEdge(0, 1);**

**g.addEdge(0, 2);**

**g.addEdge(1, 2);**

**g.addEdge(2, 0);**

**g.addEdge(2, 3);**

**g.addEdge(3, 3);**

**if(g.isCyclic())**

**cout << "Graph contains cycle";**

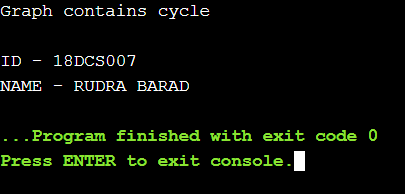
**else**

**cout << "Graph doesn't contain cycle";**

**cout<<"\n\nID - 18DCS007\nNAME - RUDRA BARAD";**

**return 0;**

**OUTPUT**



**CONCLUSION**

**Depth First Traversal can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. Time Complexity: O(V + E)**

**6.2 AIM: From a given vertex in a weighted graph, implement a PROGRAM CODE to find shortest paths to other vertices using Dijkstra’s algorithm.**

**PROGRAM CODE**

**#include<stdio.h>**

**#define INFINITY 9999**

**#define MAX 10**

**void dijkstra(int G[MAX][MAX],int n,int startnode);**

**int main()**

**{ int G[MAX][MAX],i,j,n,u;**

**printf("Enter no. of vertices:");**

**scanf("%d",&n);**

**printf("\nEnter the adjacency matrix:\n");**

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

**for(j=0;j<n;j++)**

**scanf("%d",&G[i][j]);**

**printf("\nEnter the starting node:");**

**scanf("%d",&u);**

**dijkstra(G,n,u);**

**printf("\n\nID - 18DCS007\nNAME - RUDRA BARAD");**

**return 0;**

**}**

**void dijkstra(int G[MAX][MAX],int n,int startnode)**

**{ int cost[MAX][MAX],distance[MAX],pred[MAX];**

**int visited[MAX],count,mindistance,nextnode,i,j;**

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

**for(j=0;j<n;j++)**

**if(G[i][j]==0)**

**cost[i][j]=INFINITY;**

**else**

**cost[i][j]=G[i][j];**

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

**{**

**distance[i]=cost[startnode][i];**

**pred[i]=startnode;**

**visited[i]=0;**

**}**

**distance[startnode]=0;**

**visited[startnode]=1;**

**count=1;**

**while(count<n-1)**

**{**

**mindistance=INFINITY;**

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

**if(distance[i]<mindistance&&!visited[i])**

**{**

**mindistance=distance[i];**

**nextnode=i;**

**}**

**visited[nextnode]=1;**

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

**if(!visited[i])**

**if(mindistance+cost[nextnode][i]<distance[i])**

**{ distance[i]=mindistance+cost[nextnode][i];**

**pred[i]=nextnode;**

**}**

**count++;**

**}**

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

**if(i!=startnode)**

**{ printf("\nDistance of node%d=%d",i,distance[i]);**

**printf("\nPath=%d",i);**

**j=i;**

**do**

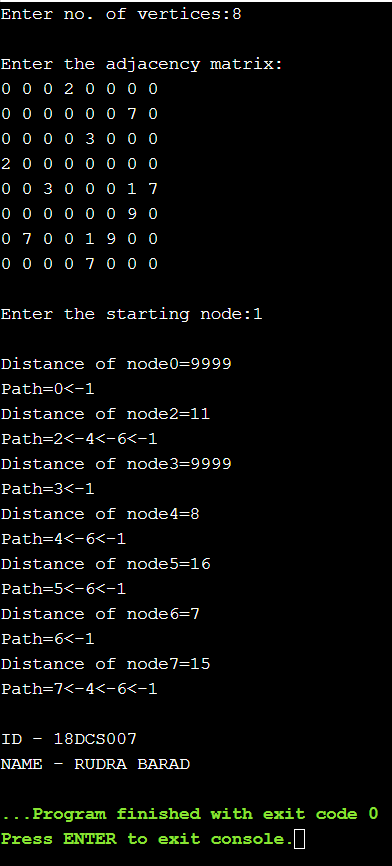
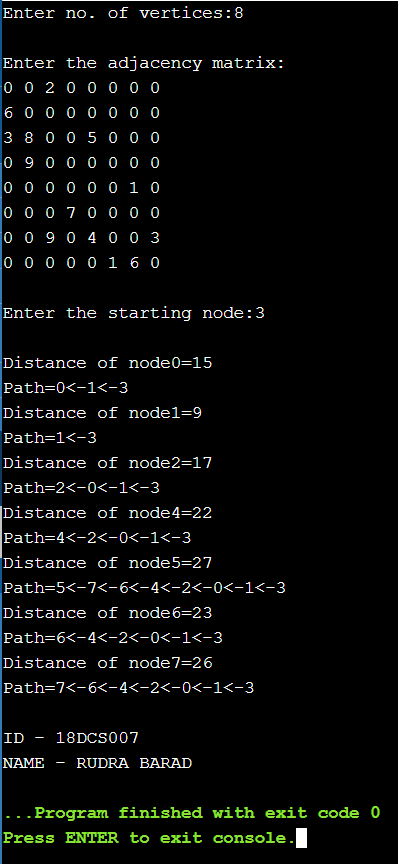
**{ j=pred[j];**

**printf("<-%d",j);**

**}while(j!=startnode);**

**}}**

**OUTPUT**



**CONCLUSION**

**Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.Time Complexity: O(ElogV)**

**6.3 AIM: Find Minimum Cost spanning tree of a given undirected graph using Prim’s algorithm.**

**PROGRAM CODE**

**#include<stdio.h>**

**#include<stdlib.h>**

**#define infinity 9999**

**#define MAX 20**

**int G[MAX][MAX],spanning[MAX][MAX],n;**

**int prims();**

**int main()**

**{ int i,j,total\_cost;**

**printf("Enter no. of vertices:");**

**scanf("%d",&n);**

**printf("\nEnter the adjacency matrix:\n");**

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

**for(j=0;j<n;j++)**

**scanf("%d",&G[i][j]);**

**total\_cost=prims();**

**printf("\nspanning tree matrix:\n");**

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

**{**

**printf("\n");**

**for(j=0;j<n;j++)**

**printf("%d\t",spanning[i][j]); }**

**printf("\n\nTotal cost of spanning tree=%d",total\_cost);**

**printf("\n\nID - 18DCS007\nNAME - RUDRA BARAD");**

**return 0; }**

**int prims()**

**{ int cost[MAX][MAX];**

**int u,v,min\_distance,distance[MAX],from[MAX];**

**int visited[MAX],no\_of\_edges,i,min\_cost,j;**

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

**for(j=0;j<n;j++)**

**{ if(G[i][j]==0)**

**cost[i][j]=infinity;**

**else**

**cost[i][j]=G[i][j];**

**spanning[i][j]=0;**

**}**

**distance[0]=0;**

**visited[0]=1;**

**for(i=1;i<n;i++)**

**{ distance[i]=cost[0][i];**

**from[i]=0;**

**visited[i]=0;**

**}**

**min\_cost=0;**

**no\_of\_edges=n-1;**

**while(no\_of\_edges>0)**

**{ min\_distance=infinity;**

**for(i=1;i<n;i++)**

**if(visited[i]==0&&distance[i]<min\_distance)**

**{ v=i;**

**min\_distance=distance[i]; }**

**u=from[v];**

**spanning[u][v]=distance[v];**

**spanning[v][u]=distance[v];**

**no\_of\_edges--;**

**visited[v]=1;**

**for(i=1;i<n;i++)**

**if(visited[i]==0&&cost[i][v]<distance[i])**

**{ distance[i]=cost[i][v];**

**from[i]=v;**

**}**

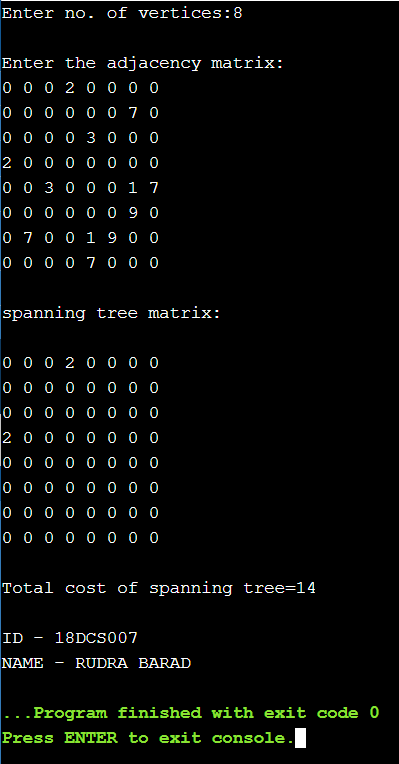
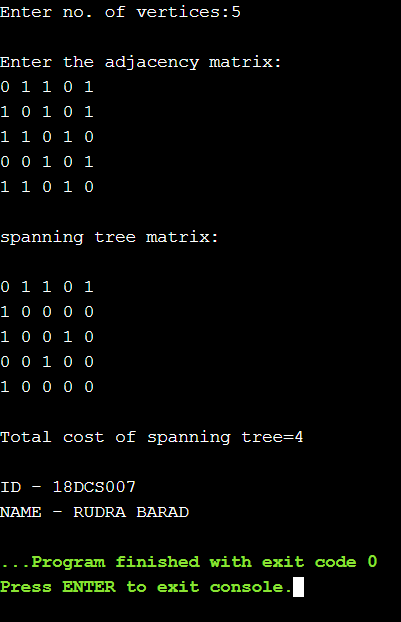
**min\_cost=min\_cost+cost[u][v];**

**}**

**return(min\_cost);**

**}**

**OUTPUT**



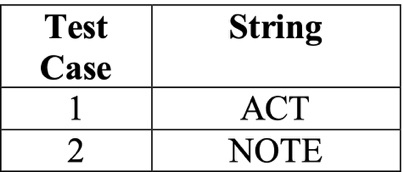
**CONCLUSION**

**Prim's algorithm is very similar to Kruskal's: whereas Kruskal's "grows" a forest of trees, Prim's algorithm grows a single tree until it becomes the minimum spanning tree.Time Complexity: O(ElogV)**

**PRACTICAL - 7**

**BACKTRACKING**

**AIM: Implement a PROGRAM CODE to print all permutations of a given string.**



**PROGRAM CODE**

**#include <stdio.h>**

**#include <string.h>**

**void swap(char \*x, char \*y)**

**{**

**char temp;**

**temp = \*x;**

**\*x = \*y;**

**\*y = temp;**

**}**

**void permute(char \*a, int l, int r)**

**{**

**int i;**

**if (l == r)**

**printf("%s\n", a);**

**else**

**{**

**for (i = l; i <= r; i++)**

**{**

**swap((a+l), (a+i));**

**permute(a, l+1, r);**

**swap((a+l), (a+i)); //backtrack**

**} } }**

**int main()**

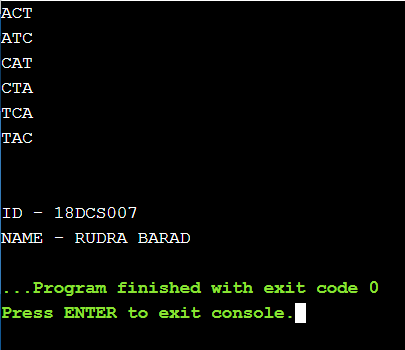
**{ char str[] = "NOTE"; //ACT**

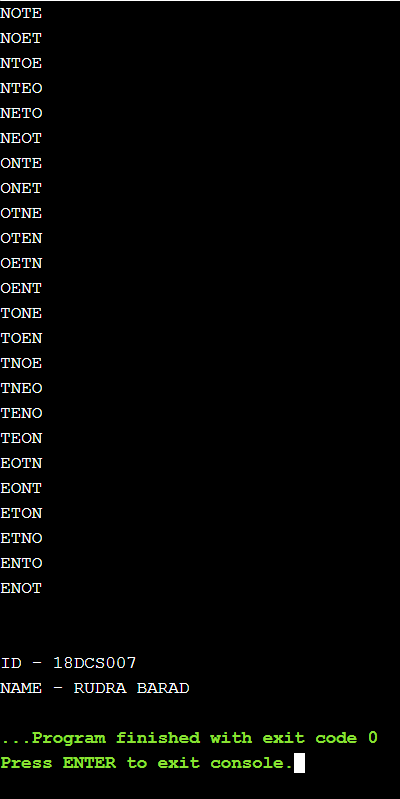
**int n = strlen(str);**

**permute(str, 0, n-1);**

**printf("\n\nID - 18DCS007\nNAME - RUDRA BARAD");**

**return 0;**

**OUTPUT**



**PRACTICAL - 8**

**String Matching Algorithm**

**8.1 AIM: Suppose you are given a source string S[0 ..n − 1] of length n, consisting of symbols a and b. Suppose that you are given a pattern string P[0 ..m − 1] of length m < n, consisting of symbols a, b, and \*, representing a pattern to be found in string S. The symbol \* is a “wild card” symbol, which matches a single symbol, either a or b. The other symbols must match exactly. The problem is to output a sorted list M of valid “match positions”, which are positions j in S such that pattern P matches the substring S [j..j + |P|− 1]. For example, if S = ababbab and P = ab\*, then the output M should be [0, 2]. Implement a straightforward, naive algorithm to solve the problem.**

**PROGRAM CODE**

**#include<stdio.h>**

**#include<string.h>**

**int main()**

**{ char t[100],p[100];**

**int tn,pn,shift[20]={0},s=0,i,j=0,count=0,m=0;**

**printf("\n Enter The Text : ");**

**scanf("%s",t);**

**fflush(stdin);**

**printf("\n Enter The Pattern : ");**

**scanf("%s",p);**

**tn = strlen(t);**

**pn = strlen(p);**

**while(s!=(tn-pn+1))**

**{ j=0;**

**for(i=s;i<pn+s;i++)**

**{ if(p[j]==t[i])**

**{ count++;**

**if(count==pn)**

**{ count=0;**

**shift[m]=s;**

**m++;**

**} }**

**else**

**{ count=0;**

**break; }**

**j++;**

**}**

**s++; }**

**if(m>0)**

**{ printf("\n\n Valid Shifts : ");**

**for(i=0;i<m;i++) printf("%d \n",shift[i]);**

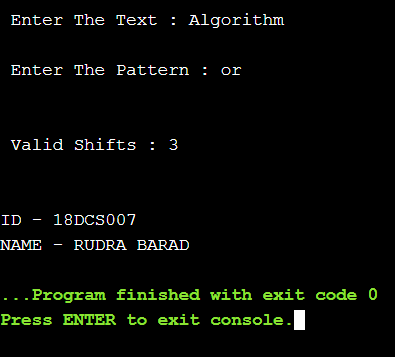
**}**

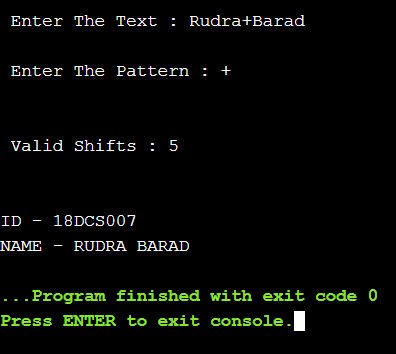
**else**

**{ printf("\n\n No Valid Shifts."); }**

**printf("\n\nID - 18DCS007\nNAME - RUDRA BARAD"); }**

**OUTPUT**

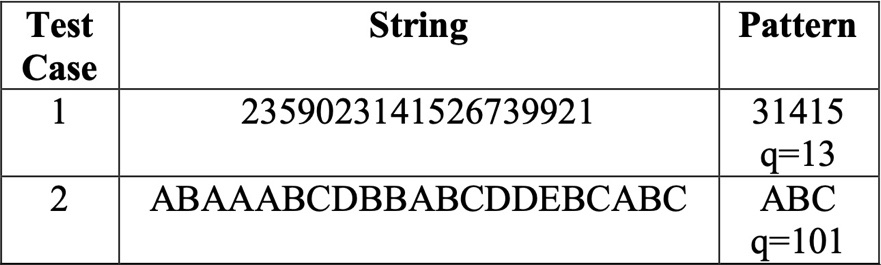




**CONCLUSION**

* **Naive pattern searching is the simplest method among other pattern searching algorithms.**
* **It checks for all character of the main string to the pattern.**
* **This algorithm is helpful for smaller texts. It does not need any pre-processing phases.**
* **The time complexity is O(m\*n). The m is the size of pattern and n is the size of the main string.**

**8.2 AIM: Implement Rabin karp algorithm and test it on the following test cases:**

****

**PROGRAM CODE**

**#include<stdio.h>**

**#include<string.h>**

**#define d 256**

**using namespace std;**

**void search(char pat[], char txt[], int q)**

**{**

**int M = strlen(pat);**

**int N = strlen(txt);**

**int i, j;**

**int p = 0; // hash value for pattern**

**int t = 0; // hash value for txt**

**int h = 1;**

**for (i = 0; i < M - 1; i++)**

**h = (h \* d) % q;**

**for (i = 0; i < M; i++)**

**{**

**p = (d \* p + pat[i]) % q;**

**t = (d \* t + txt[i]) % q;**

**}**

**for (i = 0; i <= N - M; i++)**

**{**

**if ( p == t )**

**{**

**for (j = 0; j < M; j++)**

**{**

**if (txt[i+j] != pat[j])**

**break;**

**}**

**if (j == M)**

**printf("Pattern found at index %d \n",i);**

**}**

**if ( i < N-M )**

**{**

**t = (d\*(t - txt[i]\*h) + txt[i+M])%q;**

**if (t < 0)**

**t = (t + q);**

**}**

**}**

**}**

**int main()**

**{**

**char txt[] = "2359023141526739921"; //ABAAABCDBBABCDDEBCABC**

**char pat[] = "31415"; //ABC**

**int q = 101;**

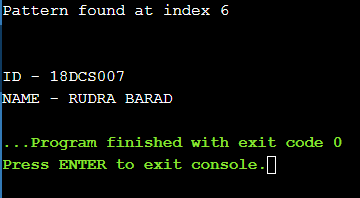
**search(pat, txt, q);**

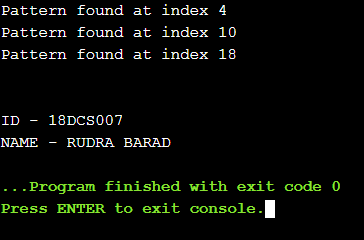
**printf("\n\nID - 18DCS007\nNAME - RUDRA BARAD");**

**return 0;**

**}**

**OUTPUT**





**CONCLUSION**

* **The Rabin–Karp algorithm is a form of**[**rolling hash**](https://en.wikipedia.org/wiki/Rolling_hash)**used in string searching**[**[1]**](https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm#cite_note-1)**to find any one of a set of pattern strings in a text.**
* **For text of length n and p patterns of combined length m, its average and best case time complexity is**[**O**](https://en.wikipedia.org/wiki/Big-O_notation)**(n+m) , but its worst-case time complexity is O(nm).**