**MEERUT INSTITUTE OF ENGINEERING & TECHNOLOGY, MEERUT**

N.H. 58, Delhi-Roorkee Highway, Baghpat Road Bypass Crossing, Meerut-250005


**Department of Computer Science and Engineering**
**B. Tech. (Session 2018–19)**

**Lab Manual**

# Design And Analysis of Algorithm Lab

**(RCS-552)**

| L | T | P |
|---|---|---|
| 0 | 0 | 2 |

**Ms. Anjali Sharma**

**Mr. Baldey Mitra**

**Ms. Surbhi**

**Mr. Vivek Kumar**

**Mr. Vinod Kumar**

**MEERUT INSTITUTE OF ENGINEERING & TECHNOLOGY, MEERUT**
N.H. 58, Delhi-Roorkee Highway, Baghpat Road Bypass Crossing, Meerut-250005

**Computer Science & Engineering Department**
**B. Tech. 3rd Year/ 5th Semester**
**(Session: 2018 – 19)**

**DAA Lab. (RCS-552)**

| L | T | P |
|---|---|---|
| 0 | 0 | 2 |

1. Write a program to implement Insertion Sort on an unsorted array.
2. Write a program to implement Selection Sort on an unsorted array.
3. Write a program to implement Quick Sort on an unsorted array.
4. Write a program to implement Counting Sort on an unsorted array.
5. Write a program to Fractional Knapsack Problem.
6. Write a program to 0-1 Knapsack Problem.
7. Write a program to implement Strassen's Matrix Multiplication.
8. Write a program to implement Longest Common Subsequence.
9. Write a program to implement Job Sequencing.
10. Write a program to implement Naive Based String Matching.

**Value Addition:**

1. Write a program to implement Dijkstra's algorithm.
2. Write a program to implement Floyd warshall algorithm.
3. Write a program to implement N-Queens problem.

| | | |
|---|---|---|
| ……. | ……. | ……. |
| **Signature of Faculty Lab In-charge** | **Signature of Subject Coordinator** | **Head of Department** |

## Lab Course Outcome

- Design and implement various Linear and Polynomial Time Sorting algorithms.
- Employ various design strategies for problem solving and Implement various algorithms in a high level language.
- Synthesize divide-and-conquer algorithms.
- Identify the problem given and design the algorithm using various algorithm design techniques.
- Analyze the performance of various algorithms.
- Compare the performance of different algorithms for same problem.

Experiment 1

Object: Write a program to implement Insertion Sort on an unsorted array.

Algorithm:

```
insertionSort( A : array of items )
  int holePosition
  int valueToInsert

  for i = 1 to length(A) inclusive do:

    /* select value to be inserted */
    valueToInsert = A[i]
    holePosition = i

    /*locate hole position for the element to be inserted */

    while holePosition > 0 and A[holePosition-1] > valueToInsert do:
      A[holePosition] = A[holePosition-1]
      holePosition = holePosition -1
    end while

    /* insert the number at hole position */
    A[holePosition] = valueToInsert

  end for

end procedure
```

Experiment 2

Object:Write a program to implement Selection Sort on an unsorted array.

Algrothm:

1. Set MIN to location 0
2. Search the minimum element in the list\
3. Swap with value at location MIN
4. Increment MIN to point to next element
5. Repeat until list is sorted

**Pseudocode**

```
list  : array of items n      : size of list

  for i = 1 to n - 1
  /* set current element as minimum*/
    min = i

    /* check the element to be minimum */

    for j = i+1 to n
      if list[j] < list[min] then
         min = j;
      end if
    end for

    /* swap the minimum element with the current element*/
    if indexMin != i  then
      swap list[min] and list[i]
    end if
  end for
```

Experiment 3

Object:Write a program to implement Quick Sort on an unsorted array.

Alorithm:

```
partitionFunc(left, right, pivot)
  leftPointer = left
  rightPointer = right - 1

  while True do
    while A[++leftPointer] < pivot do
      //do-nothing
    end while

    while rightPointer > 0 && A[--rightPointer] > pivot do
      //do-nothing
    end while

    if leftPointer >= rightPointer
      break
    else
      swap leftPointer,rightPointer
    end if

  end while

  swap leftPointer,right
  return leftPointer

end function

procedure quickSort(left, right)

  if right-left <= 0
    return
  else
    pivot = A[right]
    partition = partitionFunc(left, right, pivot)
    quickSort(left,partition-1)
    quickSort(partition+1,right)
  end if
```

Experiment 4

Object:Write a program to implement Counting Sort on an unsorted array.

Algorithm:

```
counting sort(array, size)
Begin
  max = get maximum element from array.
  define count array of size [max+1]

  for i := 0 to max do
    count[i] = 0 //set all elements in the count array to 0
  done

  for i := 1 to size do
    increase count of each number which have found in the array
  done

  for i := 1 to max do
    count[i] = count[i] + count[i+1] //find cumulative frequency
  done

  for i := size to 1 decrease by 1 do
    store the number in the output array
    decrease count[i]
  done

  return the output array
```

Experiment5

Object: Write a program to Fractional Knapsack Problem.

Algorithm:

1. Compute the profit per weight density for each item using the formula $\mathbf{d_i = P_i / w_i}$.
2. Sort each item by its profit per weight density.
3. Maximize the profit i.e Take as much as possible of the profit per weight density item not already in the bag.

Experiment 6

Object:Write a program to 0-1 Knapsack Problem.

Algorithm:

Input :
{w1,w2,.....wn }, W , {V1 ,V2 ,……..Vn}

Output :
T[n,W]

```
 for i = 0 to W do
    T[0,i] = 0
 For i = 1 to n
{
     B[I,o] = 0
    For(j=1 to W) do
    If(wk<= j) and T[i-1,j-wk] + Vk> T[i-1,j]
    Then T[i,j] = T[i-1,j-wk] + Vk
    Else T[I,j] = T[i-1,j]
}
```

Experiment 7

Object:Write a program to implement Strassen's Matrix Multiplication.

Algorithm:

**D1 = (a11 + a22) (b11 + b22)**
**D2 = (a21 + a22).b11**
**D3 = (b12 – b22).a11**
**D4 = (b21 – b11).a22**
**D5 = (a11 + a12).b22**
**D6 = (a21 – a11) . (b11 + b12)**
**D7 = (a12 – a22) . (b21 + b22)**

**C11 = d1 + d4 – d5 + d7**
**C12 = d3 + d5**
**C21 = d2 + d4**
**C22 = d1 + d3 – d2 – d6**

**Algorithm for Strassen's matrix multiplication**

```
Algorithm Strassen(n, a, b, d)
begin
  If n = threshold then compute
        C = a * b is a conventional matrix.
  Else
        Partition a into four sub matrices  a11, a12, a21, a22.
        Partition b into four sub matrices b11, b12, b21, b22.
        Strassen ( n/2, a11 + a22, b11 + b22, d1)
        Strassen ( n/2, a21 + a22, b11, d2)
        Strassen ( n/2, a11, b12 – b22, d3)
        Strassen ( n/2, a22, b21 – b11, d4)
        Strassen ( n/2, a11 + a12, b22, d5)
        Strassen (n/2, a21 – a11, b11 + b22, d6)
        Strassen (n/2, a12 – a22, b21 + b22, d7)

        C = d1+d4-d5+d7     d3+d5
        d2+d4          d1+d3-d2-d6

  end if

  return (C)
end.
```

Experiment 8

Object:Write a program to implement Longest Common Subsequence.

Algorithm:

1. Let the input sequences be X[0..m-1] and Y[0..n-1] of lengths m and n respectively. And let L(X[0..m-1], Y[0..n-1]) be the length of LCS of the two sequences X and Y. Following is the recursive definition of L(X[0..m-1], Y[0..n-1]).
2. If last characters of both sequences match (or X[m-1] == Y[n-1]) then L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])
3. If last characters of both sequences do not match (or X[m-1] != Y[n-1]) then L(X[0..m-1], Y[0..n-1]) = MAX ( L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2]) )

Experiment 9

Object:Write a program to implement Job Sequencing.

Algorithm:

1 Sort all jobs in decreasing order of profit.
2 Initialize the result sequence as first job in sorted jobs.
3 Do following for remaining n-1 jobs
…….a) If the current job can fit in the current result sequence without missing the deadline, add current job to the result. Else ignore the current job.

Experiment 10

Object:Write a program to implement Naive Based String Matching.

Algorithm:

```
naivePatternSearch(pattern, text)
Input: The text and the pattern
Output: location, where patterns are present in the text
Begin
  patLen := pattern Size
  strLen := string size

  for i := 0 to (strLen - patLen), do
    for j := 0 to patLen, do
      if text[i+j] ≠ pattern[j], then
        break the loop
    done

    if j == patLen, then
      display the position i, as there pattern found
  done
End
```

**Value Addition:**

## Experiment 1

**Object:** Write a program to implement Dijkstra's algorithm.

**Algorithm:**

**Dijkstra's-Algorithm (G, w, s)**

```
for each vertex v Є G.V
   v.d := ∞
   v.∏ := NIL
s.d := 0
S := Φ
Q := G.V
while Q ≠ Φ
   u := Extract-Min (Q)
   S := S U {u}
   for each vertex v Є G.adj[u]
      if v.d > u.d + w(u, v)
         v.d := u.d + w(u, v)
         v.∏ := u
```

## Experiment 2

Object:Write a program to implement Floyd warshall algorithm.

Algorithm:

```
Begin
  for k := 0 to n, do
    for i := 0 to n, do
      for j := 0 to n, do
        if cost[i,k] + cost[k,j] < cost[i,j], then
          cost[i,j] := cost[i,k] + cost[k,j]
      done
    done
  done
  display the current cost matrix
End
```

## Experiment 3

Object: Write a program to implement N-Queens problem.

Algorithm:

**isValid(board, row, col)**

**Input:** The chess board, row and the column of the board.

**Output:** True when placing a queen in row and place position is a valid or not.

```
Begin
  if there is a queen at the left of current col, then
    return false
  if there is a queen at the left upper diagonal, then
    return false
  if there is a queen at the left lower diagonal, then
    return false;
  return true //otherwise it is valid place
End
```

**solveNQueen(board, col)**

**Input:** The chess board, the col where the queen is trying to be placed.

**Output:** The position matrix where queens are placed.

```
Begin
  if all columns are filled, then
    return true
  for each row of the board, do
    if isValid(board, i, col), then
      set queen at place (i, col) in the board
      if solveNQueen(board, col+1) = true, then
        return true
      otherwise remove queen from place (i, col) from board.
  done
  return false
End
```