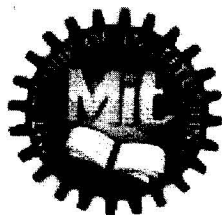


Moradabad Institute of Technology
Moradabad



In Pursuit Of Excellence

LAB MANUAL

DESIGN AND ANALYSIS OF ALGORITHM

(RCS 552)


Session: 2019-2020

prepared by-

Mr. Manish Gupta


Ms. Neha Gupta

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MORADABAD INSTITUTE OF TECHNOLOGY
MIT GROUP OF INSTITUTIONS
MORADABAD - 244001

 In Pursuit of Excellence	Course Evaluation Scheme	SESSION-2019-2020
		SEM- 5 th

**B. Tech. (CSE\CSIT)
FIFTH SEMESTER**

Sl No.	Subject Code	Subject Name	L-T-P	Theory/ Lab (ESE) Marks	Sessional		Total	Credi t
					Test	Assign/Att		
1	RAS501	MANAGERIAL ECONOMICS	3--0--0	70	20	10	100	3
2	RAS502/ RUCS01	INDUSTRIAL SOCIOLOGY /CYBER SECURITY	3--0--0	70	20	10	100	3
3	RCS-501	Database Management Systems	3--0--0	70	20	10	100	3
4	RCS-502	Design and Analysis of Algorithm	3--1--0	70	20	10	100	4
5	RCS-503	Principles of Programing Languages	3--0--0	70	20	10	100	3
6	CS-Elective-1	DEPTT ELECTIVE COURSE-1	3--1--0	70	20	10	100	4
7	RCS-551	Database Management Systems Lab	0--0--2	50	-	50	100	1
8	RCS-552	Design and Analysis of Algorithm Lab	0--0--2	50	-	50	100	1
9	RCS-553	Principles of Programing Languages Lab	0--0--2	50	-	50	100	1
10	RCS-554	Web Technologies Lab	0--0--2	50	-	50	100	1
	TOTAL						1000	24

 In Pursuit of Excellence	Course Outcome of Practical	SESSION-2019-2020
		SEM- 5 th

At the end of course, Students will be able to :


Course Code	CO	Course Outcomes(COs)	Cognitive Levels
RCS552	CO1	RCS552.1 Implement algorithm to solve problems by iterative approach.	Understand
	CO2	RCS552.2 Implement algorithm to solve problems by divide and conquer approach	Apply
	CO3	RCS552.3 Implement algorithm to solve problems by Greedy algorithm approach.	Apply
	CO4	RCS552.4 Implement algorithm to solve problems by Dynamic programming, backtracking, branch and bound approach.	Apply
	CO5	RCS552.5 Implement algorithm to solve problems by branch and bound approach.	Apply


CO-PO Mapping

Course Code	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
RCS 552	CO1	RCS 552.1	3	1	2		2			1	1		3
	CO2	RCS 552.2	3	1	2		2			1	1		2
	CO3	RCS 552.3	3	1	2		2			1	1		2
	CO4	RCS 552.4	3	1	2		2			1	1		2
	CO5	RCS 552.5	3	1	2		2			1	1		2

CO-PSO Mapping

Course Code	CO	PSO1	PSO2	
RCS 552	CO 1	RCS 552.1	3	3
	CO 2	RCS 552.2	3	3
	CO 3	RCS 552.3	3	3
	CO 4	RCS 552.4	3	3
	CO 5	RCS 552.5	3	3


Dr. Somesh Kumar
 Prof. & Head, CSE
 Moradabad Institute of Technology
 Moradabad-244001

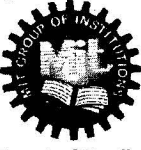
 In Pursuit of Excellence	Course Syllabus as per University	SESSION-2019-2020
		SEM-5 th

RCS-552 Design and Analysis of Algorithm Lab

List of Experiments

1. Program for Recursive Binary & Linear Search.
2. Program for Heap Sort.
3. Program for Merge Sort.
4. Program for Selection Sort.
5. Program for Insertion Sort.
6. Program for Quick Sort.
7. Knapsack Problem using Greedy Solution
8. Perform Travelling Salesman Problem
9. Find Minimum Spanning Tree using Kruskal's Algorithm
10. Implement N Queen Problem using Backtracking


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

 <p>In Pursuit of Excellence</p>	<p align="center"><u>List of Experiment with Enhancement by the Faculty Member</u></p>	<p>SESSION-2019-2020</p>
---	--	--------------------------

RCS-552 Design and Analysis of Algorithm Lab

Pre-requisites:

The student should have basic knowledge of programming skills.

List of Experiments

1. Program for Recursive Binary & Linear Search.
2. Program for Bubble Sort (**Beyond syllabus**)
3. Program for Heap Sort.
4. Program for Merge Sort.
5. Program for Selection Sort.
6. Program for Insertion Sort.
7. Program for Quick Sort.
8. Program for Counting Sort (**Beyond syllabus**)
9. Program for Radix Sort (**Beyond syllabus**)
10. Program for Bucket Sort (**Beyond syllabus**)
11. Program for Shell Sort (**Beyond syllabus**)
12. Knapsack Problem using Greedy Solution (0/1 knapsack)
13. Perform Travelling Salesman Problem
14. Find Minimum Spanning Tree using Kruskal's Algorithm
15. Implement N Queen Problem using Backtracking




In Pursuit of Excellence

Theory Syllabus

SESSION-2019-2020

RCS-502: Design and Analysis of Algorithm		3-1-0
Unit	Topic	Proposed Lecture
I	Introduction: Algorithms, Analyzing Algorithms, Complexity of Algorithms, Growth of Functions, Performance Measurements, Sorting and Order Statistics - Shell Sort, Quick Sort, Merge Sort, Heap Sort, Comparison of Sorting Algorithms, Sorting in Linear Time.	08
II	Advanced Data Structures: Red-Black Trees, B – Trees, Binomial Heaps, Fibonacci Heaps, Tries, Skip List	08
III	Divide and Conquer with Examples Such as Sorting, Matrix Multiplication, Convex Hull and Searching. Greedy Methods with Examples Such as: Optimal Reliability Allocation, Knapsack, Minimum Spanning Trees – Prim's and Kruskal's Algorithms, Single Source Shortest Paths - Dijkstra's and Bellman Ford Algorithms.	08
IV	Dynamic Programming with Examples Such as Knapsack. All Pair Shortest Paths – Warshal's and Floyd's Algorithms, Resource Allocation Problem. Backtracking, Branch and Bound with Examples Such as Travelling Salesman Problem, Graph Coloring, n-Queen Problem, Hamiltonian Cycles and Sum of Subsets.	08
V	Selected Topics: Algebraic Computation, Fast Fourier Transform, String Matching, Theory of NP-Completeness, Approximation Algorithms and Randomized Algorithms	08
References: <ol style="list-style-type: none">1. Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, "Introduction to Algorithms", Prentice 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, 2008.4. LEE "Design & Analysis of Algorithms (POD)", McGraw Hill5. Gajendra Sharma, Design & Analysis of Algorithms, Khanna Publishing House6. Richard E. Neapolitan "Foundations of Algorithms" Jones & Bartlett Learning7. Jon Kleinberg and Éva Tardos, Algorithm Design, Pearson, 2005.8. Michael T Goodrich and Roberto Tamassia, Algorithm Design: Foundations, Analysis, and Internet Examples, Second Edition, Wiley, 2006.9. Harry R. Lewis and Larry Denenberg, Data Structures and Their Algorithms, Harper Collins, 199710. Robert Sedgewick and Kevin Wayne, Algorithms, fourth edition, Addison Wesley, 2011.11. Harsh Bhasin, "Algorithm Design and Analysis", First Edition, Oxford University Press.12. Gilles Brassard and Paul Bratley, Algorithmics: Theory and Practice, Prentice Hall, 1995.		


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



In Pursuit of Excellence

**Vision & Mission of
Institute**

SESSION-
2019-2020

SEM- 5th

Vision of the Institute


To develop industry ready professionals with values and ethics for global needs.

Mission of the Institute

M1: To impart education through outcome based pedagogic principles.

M2: To provide conducive environment for personality development, training and entrepreneurial skills.

M3: To induct high professional ethics and accountability towards society in students.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

 In Pursuit of Excellence	Vision & Mission of Department	SESSION- 2019-2020
		SEM- 5 th

Vision of the Department

To develop globally recognized computer science and engineering graduates with ethical values for need of software industries.


Mission of the Department

M1: To impart knowledge through well defined instructional objectives in the field of computer science and engineering.

M2: To provide learning ambience for skills, innovation, leadership and overall personality development.

M3: To inculcate professional ethics, teamwork and responsiveness towards society.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

 In Pursuit of Excellence	Program Education Objectives	SESSION- 2019-2020
		SEM- 5 th


The graduates after 3-5 years of program completion will

PEO1: be having entrepreneurial and employable skills in software industries, by adapting themselves in the corporate world by utilizing the defined instructional objectives learnt in the program.

PEO2: engage in skill enhancement, that would help to work in their own area of interest, individually or in a team.

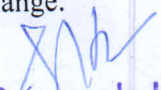
PEO3: demonstrate ownership and responsiveness towards the profession and the society.



Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

 In Pursuit of Excellence	Program Outcomes	SESSION- 2019-2020
		SEM- 5 th

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyses complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
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 information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
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 professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.


Dr. Somesh Kumar
 Prof. & Head, CSE
 Moradabad Institute of Technology
 Moradabad-244001

 In Pursuit of Excellence	Program Specific Outcomes	SESSION-2019-2020
		SEM- 5 th

Program Specific Outcomes

After completing their graduation, students of Computer Science and Engineering will be able to

1. Comprehend the core subjects of CSE and apply them to resolve domain specific tribulations.
2. Extrapolate the fundamental concepts in engineering and to apply latest technology with programming language skills to develop, test, implement and maintain software products.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

INSTRUCTIONS

1. Use comments, along with statements, in your program to make it clear and easy to understand.
2. Make the variable names meaningful so that they could convey the purpose of their use.
3. Make sure that all the allocated memory blocks have been deleted.
4. Instead of using nested if - else blocks use switch-case construct.
5. Write the description of your program in pseudo code, along with the code listing in the form of data dictionary.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

DEBUGGING

1. Check the pair of curly braces.
2. Check the data type of variables used.
3. Check that the memory is allocated to the static variables.
4. Check that the accessibility restrictions are not violated.
5. Check that the pointer variables are allocated the memory before any references is made.
6. Check that the loops are finite. Ensure that the iteration control variables are modified for each iteration.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. – 1

OBJECTIVE:

Write a Program for Recursive Binary & Linear Search.

PROGRAM LOGIC:-

Searching is the process of finding the location of given element in the array. The search is said to be successful if the given element is found i.e. the element does exist in the array; otherwise unsuccessful.

There are two approaches to search operation:-

- Linear Search
- Binary Search

(a) Linear Search:-

In Linear Search, we access each element of an array one by one sequentially and see whether desired element is present in the array or not. A search will be successful if all the elements are accessed and the desired element is found. In worst case, the desired element is not present in the array. The number of average case we may have to scan half of the size of the array ($n/2$). Therefore linear search can be defined as the technique which traverses the array sequentially to locate the given item.

Algorithm:

```
linear Search (A, n, K, i)
  While i < n
    if k = a[i]
      return i
    else
      return linear_search(A, n, K, i+1)
  return -1
```


(b) Binary Search:-

Binary search algorithm is a faster algorithm which reduces running time effectively when applied to a list of large number of elements. The precondition for the algorithm is that the list should be in sorted order, either increasing or decreasing, whatever the case may be.

Here is the algorithm which takes the list, the element to be searched as parameters and returns the place of searched element in the list if found else return -1.

Algorithm:

```
BinarySearch(A[0..N-1], value, low, high)
{
  // invariants: value > A[i] for all i < low
                value < A[i] for all i > high
  if (high < low)
```


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
    return not_found // value would be inserted at index "low"  
mid = (low + high) / 2  
if (A[mid] > value)  
    return BinarySearch(A, value, low, mid-1)  
else if (A[mid] < value)  
    return BinarySearch(A, value, mid+1, high)  
else  
    return mid  
}
```

LAB VIVA QUESTIONS:

- Q1: What is searching?
- Q2: How the searching is essential for database applications?
- Q3: Explain Linear and Binary Search?
- Q4: Differentiate between Linear and Binary Search
- Q5: What are the different searching techniques known to you? Explain one of them in detail with a suitable example.
- Q6: What is the time complexity of linear search and binary search?
- Q7: What is the data structure used to perform recursion?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 2

OBJECTIVE

Write a program to implement Bubble Sort

PROGRAM LOGIC:-

Bubble sort, sometimes shortened to bubble sort, also known as exchange sort, is a simple sorting algorithm. It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which means the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top (i.e. the beginning) of the list via the swaps. Because it only uses comparisons to operate on elements, it is a comparison sort. This is the easiest comparison sort to implement.

For example, using Bubble sort algorithms if our initial array is:

12, 9, 4, 99, 120, 1, 3, 10

The basic steps followed by algorithm:-

In the first step compare first two values 12 and 9.

12 9 4 99 120 1 3 10

As $12 > 9$ then we have to swap these values
Then the new sequence will be

9 12 4 99 120 1 3 10

In next step take next two values 12 and 4

9 12 4 99 120 1 3 10

Compare these two values .As $12 > 4$ then we have to swap these values.
Then the new sequence will be

9 4 12 99 120 1 3 10

we have to follow similar steps up to end of array. e.g.

9 4 12 99 120 1 3 10

9 4 12 99 120 1 3 10

9 4 12 99 1 120 3 10

9 4 12 99 1 120 3 10

9 4 12 99 1 3 120 10


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

9 4 12 99 1 3 10 120

When we reached at last index .Then restart same steps unlit the data is not sorted.

The output of this example will be :

1 3 4 9 10 12 99 120

ALGORITHM:

Step 1: for $i=1$ to $\text{length}[A]$

Step 2: do for $j = \text{length}[A]$ down to $i+1$

Step 3: do if $A[j] < A[j-1]$

Step 4: then exchange $A[j] \leftrightarrow A[j-1]$

LAB VIVA QUESTIONS:

Q1: What is sorting?

Q2: How the sorting is essential for database applications?

Q3: What is the time complexity of bubble sort?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. – 3

OBJECTIVE:

Write a Program for Heap Sort.

PROGRAM LOGIC:

Heap sort is one of the best general-purpose sorting algorithms, a comparison sort and part of the selection sort family. Although somewhat slower in practice on most machines than a good implementation of quick sort, it has the advantages of worst-case $O(n \log n)$ runtime. Heap sort is an in-place algorithm and is not a stable sort.

The heap sort is the slowest of the $O(n \log n)$ sorting algorithms, but unlike the merge and quick sorts it doesn't require massive recursion or multiple arrays to work. This makes it the most attractive option for *very* large data sets of millions of items.

The heap sort works as its name suggests - it begins by building a heap out of the data set, and then removing the largest item and placing it at the end of the sorted array. After removing the largest item, it reconstructs the heap and removes the largest remaining item and places it in the next open position from the end of the sorted array. This is repeated until there are no items left in the heap and the sorted array is full. Elementary implementations require two arrays - one to hold the heap and the other to hold the sorted elements.

Pros: In-place and non-recursive, making it a good choice for extremely large data sets.

Cons: Slower than the merge and quick sorts.

Maintaining the Heap property

MAX-HEAPIFY is an important routine for manipulating max-heaps. This procedure can be called in a bottom-up manner to convert an array $A[1 \dots n]$, where $n = \text{length}[A]$, into a max heap. When MAX-HEAPIFY is called it is assumed that the binary trees located at $\text{Left}(i)$ and $\text{Right}(i)$ are max-heaps, but that $A(i)$ may be smaller than its children, thus violating the max heap property.

MAX-HEAPIFY (A, i)

STEP 1: $l \leftarrow \text{Left}(i)$

STEP 2: $r \leftarrow \text{Right}(i)$


STEP 3: if $l \leq \text{Heap-size}[A]$ and $A[l] > A[i]$

STEP 4: then $\text{largest} \leftarrow l$

STEP 5: else $\text{largest} \leftarrow i$

STEP 6: if $r \leq \text{Heap-size}[A]$ and $A[r] > A[\text{largest}]$

STEP 7: then $\text{largest} \leftarrow r$


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

STEP 8: if largest! = i

STEP 9: then exchange $A[i] \leftrightarrow A[\text{largest}]$

STEP 10: MAX-HEAPIFY (A, largest)

Building a Heap

The procedure BUILD-MAX-HEAP goes through the nodes of the tree and runs MAX-HEAPIFY on each one.

BUILD-MAX-HEAP (A)

STEP 1: heap-size[A] \leftarrow length[A]

STEP 2: for $i \leftarrow$ length[A]/2 downto 1

STEP 3: do MAX_HEAPIFY (A, i)

THE HEAPSORT ALGORITHM

The heap sort algorithm starts by using BUILD-MAX-HEAP to build max heap on the input array A [1...n], where $n = \text{length}[A]$. Since the maximum element of the array is stored at $a[i]$, it can be put into its correct final position by exchanging it with $a[n]$. Now it discards the node n from the heap, remaining A [1... (N-1)] is converted into a max heap. The heap sort algorithm repeats this process for the max heap of size n-1 down to a heap of size 2.

HEAPSORT (A)

STEP 1: BUILD-MAX-HEAP (A)

STEP 2: for $1 \leftarrow$ length [A] down to 2

STEP 3: do exchange A [1] \leftrightarrow A[I]

STEP 4: heap-size [A] \leftarrow heap-size [A]-1

STEP 5: MAX-HEAPIFY (A, I)

LAB VIVA QUESTIONS:

1. What is the running time of Heap sort?
2. What technique is used to sort elements in heap sort?
3. Is heap sort in place sorting algorithm?
4. Define stable sort algorithm.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. - 4

OBJECTIVE:

Write a Program for Merge Sort.

PROGRAM LOGIC

Merge sort is based on the divide-and-conquer paradigm. Its worst-case running time has a lower order of growth than insertion sort. Since we are dealing with sub problems, we state each sub problem as sorting a sub array $A[p .. r]$. Initially, $p = 1$ and $r = n$, but these values change as we recurs through sub problems.

To sort $A[p .. r]$:

1. **Divide Step:** If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split $A[p .. r]$ into two subarrays $A[p .. q]$ and $A[q + 1 .. r]$, each containing about half of the elements of $A[p .. r]$. That is, q is the halfway point of $A[p .. r]$.
2. **Conquer Step:** Conquer by recursively sorting the two sub arrays $A[p .. q]$ and $A[q + 1 .. r]$.
3. **Combine Step:** Combine the elements back in $A[p .. r]$ by merging the two sorted sub arrays $A[p .. q]$ and $A[q + 1 .. r]$ into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

Note that the recursion bottoms out when the sub array has just one element, so that it is trivially sorted.

ALGORITHM:


To sort the entire sequence $A[1 .. n]$, make the initial call to the procedure MERGE-SORT ($A, 1, n$).

MERGE-SORT (A, p, r)

- | | | |
|----|------------------------------------|------------------------|
| 1. | IF $p < r$ | // Check for base case |
| 2. | THEN $q = \text{FLOOR}[(p + r)/2]$ | // Divide step |
| 3. | MERGE (A, p, q) | // Conquer step. |
| 4. | MERGE ($A, q + 1, r$) | // Conquer step. |
| 5. | MERGE (A, p, q, r) | // Conquer step. |

MERGE (A, p, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays $L[1 .. n_1 + 1]$ and $R[1 .. n_2 + 1]$
4. FOR $i \leftarrow 1$ TO n_1
5. DO $L[i] \leftarrow A[p + i - 1]$
6. FOR $j \leftarrow 1$ TO n_2



Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
7.   DO R[j] ← A[q + j]
8.   L[n1 + 1] ← ∞
9.   R[n2 + 1] ← ∞

10.  i ← 1
11.  j ← 1
12.  FOR k ← p TO r
13.    DO IF L[i] ≤ R[j]
14.      THEN A[k] ← L[i]
15.         i ← i + 1
16.      ELSE A[k] ← R[j]
17.         j ← j + 1
```

LAB VIVA QUESTIONS:

1. What is the running time of merge sort?
2. What technique is used to sort elements in merge sort?
3. Is merge sort in place sorting algorithm?
4. Define stable sort algorithm.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 5

OBJECTIVE:

Write a program to implement Selection Sort

PROGRAM LOGIC:

Selection sort is conceptually the most simplest sorting algorithm. This algorithm will first find the **smallest** element in the array and swap it with the element in the **first** position, then it will find the **second smallest** element and swap it with the element in the **second** position, and it will keep on doing this until the entire array is sorted.

It is called selection sort because it repeatedly **selects** the next-smallest element and swaps it into the right place.

ALGORITHM:

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

LAB VIVA QUESTIONS:

1. What is the time complexity of selection sort.
2. Is selection sort unstable?.
3. What are the steps in selection sorting?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. – 6

OBJECTIVE:

Write a Program for Insertion Sort.

PROGRAM LOGIC:

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert' in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort. The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where n is the number of items.

ALGORITHM:

```
for j ← 2 to length[A]
  do key ← A[j]
     i ← j-1
  while i > 0 and A[i] > key
    do A[i+1] ← A[i]
       i ← i-1
  A[i+1] ← key
```

LAB VIVA QUESTIONS:

1. Define incremental Approach.
2. Define Complexity.
3. Discuss time complexity of Insertion Sort.
4. Where in real world we use the same logic?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. – 7

OBJECTIVE:

Write a Program for Quick Sort.

PROGRAM LOGIC

Quicksort is a sorting algorithm developed by C. A. R. Hoare that, on average, makes $O(n \log n)$ (big O notation) comparisons to sort n items. In the worst case, it makes $O(n^2)$ comparisons, though if implemented correctly this behavior is rare. Typically, quicksort is significantly faster in practice than other $O(n \log n)$ algorithms, because its inner loop can be efficiently implemented on most architectures, and in most real-world data, it is possible to make design choices that minimize the probability of requiring quadratic time. Additionally, quicksort tends to make excellent usage of the memory hierarchy, taking perfect advantage of virtual memory and available caches. Coupled with the fact that quicksort is an in-place sort and uses no temporary memory, it is very well suited to modern computer architectures.

Quick sort (also known as "partition-exchange sort") is a comparison sort and, in efficient implementations, is not a stable sort. The complexity of algorithm is as follows:

Worst case performance $O(n^2)$
Best case performance $O(n \log n)$
Average case performance $O(n \log n)$
Worst case space complexity $O(n)$

ALGORITHM:

QUICKSORT(S, P, r)

```
1 if p < r
2   then q ← PARTITION(S, p, r)
3     QUICKSORT(S, p, q-1)
4     QUICKSORT(S, q+1, r)
```

PARTITION(S, p, r)

```
1 x ← S[r]
2 i ← p-1
3 for j ← p to r-1
4   do if S[j] ≤ x
5     then i ← i+1
6       swap S[i] ↔ S[j]
7 swap S[i+1] ↔ S[r]
8 return i+1
```


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

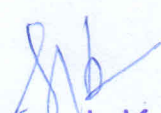
LAB VIVA QUESTIONS:

Q1: What is sorting?

Q2: How the sorting is essential for database applications?

Q3: What is the time complexity of quick sort?

Q4: What do you mean by asymptotic notations?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 8

OBJECTIVE

Write a program to implement Counting Sort

PROGRAM LOGIC

Counting sort is a linear time sorting algorithm used to sort items when they belong to a fixed and finite set. Integers which lie in a fixed interval, say k_1 to k_2 , are examples of such items.

The algorithm proceeds by defining an ordering relation between the items from which the set to be sorted is derived (for a set of integers, this relation is trivial). Let the set to be sorted be called A . Then, an auxiliary array with size equal to the number of items in the superset is defined, say B . For each element in A , say e , the algorithm stores the number of items in A smaller than or equal to e in $B(e)$. If the sorted set is to be stored in an array C , then for each e in A , taken in reverse order, $C[B[e]] = e$. After each such step, the value of $B(e)$ is decremented.

The algorithm makes two passes over A and one pass over B . If size of the range k is smaller than size of input n , then time complexity = $O(n)$. Also, note that it is a stable algorithm, meaning that ties are resolved by reporting those elements first which occur first.

ALGORITHM:

COUNTING SORT(A, B, k)

1. for $i \leftarrow 1$ to k do
2. $c[i] \leftarrow 0$
3. for $j \leftarrow 1$ to n do
4. $c[A[j]] \leftarrow c[A[j]] + 1$
5. // $c[i]$ now contains the number of elements equal to i
6. for $i \leftarrow 2$ to k do
7. $c[i] \leftarrow c[i] + c[i-1]$
8. // $c[i]$ now contains the number of elements $\leq i$
9. for $j \leftarrow n$ downto 1 do
10. $B[c[A[j]]] \leftarrow A[j]$
11. $c[A[j]] \leftarrow c[A[j]] - 1$

LAB VIVA QUESTIONS:

1. What is the running time of counting sort?
2. What technique is used to sort elements in counting sort?
3. Is counting sort in place sorting algorithm?
4. Define stable sort algorithm.


Dr. Somesh Kumar
Prof. & Head, CSE
Morebad Institute of Technology
Morebad-244001

Program No. 9

OBJECTIVE:

Write a program to implement Radix Sort

PROGRAM LOGIC:

Radix sort is one of the linear sorting algorithms for integers. It functions by sorting the input numbers on each digit, for each of the digits in the numbers. However, the process adopted by this sort method is somewhat counterintuitive, in the sense that the numbers are sorted on the least-significant digit first, followed by the second-least significant digit and so on till the most significant digit.

To appreciate Radix Sort, consider the following analogy: Suppose that we wish to sort a deck of 52 playing cards (the different suits can be given suitable values, for example 1 for Diamonds, 2 for Clubs, 3 for Hearts and 4 for Spades). The 'natural' thing to do would be to first sort the cards according to suits, then sort each of the four separate piles, and finally combine the four in order. This approach, however, has an inherent disadvantage. When each of the piles is being sorted, the other piles have to be kept aside and kept track of. If, instead, we follow the 'counterintuitive' approach of first sorting the cards by value, this problem is eliminated. After the first step, the four separate piles are combined in order and then sorted by suit. If a stable sorting algorithm (i.e. one which resolves a tie by keeping the number obtained first in the input as the first in the output) it can be easily seen that correct final results are obtained.


As has been mentioned, the sorting of numbers proceeds by sorting the least significant to most significant digit. For sorting each of these digit groups, a stable sorting algorithm is needed. Also, the elements in this group to be sorted are in the fixed range of 0 to 9. Both of these characteristics point towards the use of Counting Sort as the sorting algorithm of choice for sorting on each digit (If you haven't read the description on Counting Sort already, please do so now).

The time complexity of the algorithm is as follows: Suppose that the n input numbers have maximum k digits. Then the Counting Sort procedure is called a total of k times. Counting Sort is a linear, or $O(n)$ algorithm. So the entire Radix Sort procedure takes $O(kn)$ time. If the numbers are of finite size, the algorithm runs in $O(n)$ asymptotic time.

Example:

Following example shows how radix sort operates on seven 3-digit numbers-

Input	1 st Pass	2 nd Pass	3 rd Pass
329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

ALGORITHM:

RADIX-SORT(A, d)

1 for $i - 1$ to d

2 do use a stable sort to sort Array A on digit i

LAB VIVA QUESTIONS:

1. What is the running time of radix sort?
2. What technique is used to sort elements in radix sort?
3. Is radix sort in place sorting algorithm?
4. Define stable sort algorithm.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 10

OBJECTIVE:

Write a program to implement Bucket Sort

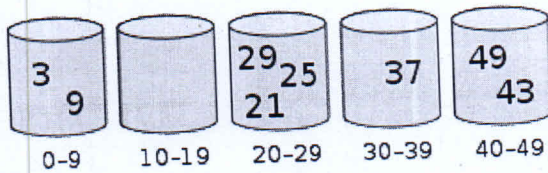
PROGRAM LOGIC:

Bucket sort, or bin sort, is a sorting algorithm that works by distributing the elements of an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm.

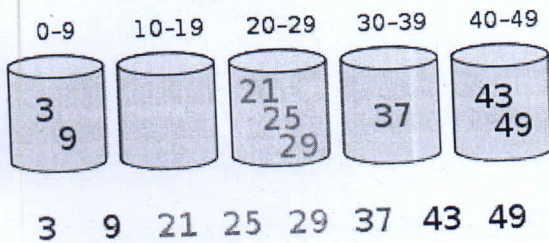
The computational complexity depends on the algorithm used to sort each bucket, the number of buckets to use, and whether the input is uniformly distributed.

Elements are distributed among bins:

29 25 3 49 9 37 21 43



Then, elements are sorted within each bin:




ALGORITHM:

```
function bucketSort(array, k) is
    buckets ← new array of k empty lists
    M ← the maximum key value in the array
    for i = 1 to length(array) do
        insert array[i] into buckets[floor(k × array[i] / M)]
    for i = 1 to k do
        nextSort(buckets[i])
    return the concatenation of buckets[1], ..., buckets[k]
```

LAB VIVA QUESTIONS:

1. What is the use of Bucket Sort?
2. How do you make a bucket sort stable?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 11

OBJECTIVE:

Write a program to implement Shell Sort

PROGRAM LOGIC:

The shell sort, sometimes called the “diminishing increment sort,” improves on the insertion sort by breaking the original list into a number of smaller sublists, each of which is sorted using an insertion sort.

This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

This algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as **interval**. This interval is calculated based on Knuth's formula as –

$$h = h * 3 + 1$$

where –

h is interval with initial value 1

ALGORITHM:

Step 1 – Initialize the value of h

Step 2 – Divide the list into smaller sub-list of equal interval h

Step 3 – Sort these sub-lists using insertion sort

Step 3 – Repeat until complete list is sorted

LAB VIVA QUESTIONS:

1. What sort does shell sort use?
2. Is shell sort stable?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 12

OBJECTIVE:

Write a Program to Implement the 0/1 Knapsack problem using (a) Dynamic Programming method (b) Greedy method

PROGRAM LOGIC:

0-1 knapsack problem The setup is the same, but the items may not be broken into smaller pieces, so we may decide either to take an item or to leave it (binary choice), but may not take a fraction of an

Let i be the highest-numbered item in an optimal solution S for W dollars. Then $S' = S - \{i\}$ is an optimal solution for $W - w_i$ dollars and the value to the solution S is V_i plus the value of the sub-problem.

We can express this fact in the following formula: define $c[i, w]$ to be the solution for items $1, 2, \dots, i$ and the maximum weight w .

The algorithm takes the following inputs

- The maximum weight W
- The number of items n
- The two sequences $v = \langle v_1, v_2, \dots, v_n \rangle$ and $w = \langle w_1, w_2, \dots, w_n \rangle$

The set of items to take can be deduced from the table, starting at $c[n, w]$ and tracing backwards where the optimal values came from.

If $c[i, w] = c[i-1, w]$, then item i is not part of the solution, and we continue tracing with $c[i-1, w]$. Otherwise, item i is part of the solution, and we continue tracing with $c[i-1, w-w_i]$.

This dynamic-0-1-knapsack algorithm takes $\theta(nw)$ times, broken up as follows: $\theta(nw)$ times to fill the c -table, which has $(n+1)(w+1)$ entries, each requiring $\theta(1)$ time to compute. $O(n)$ time to trace the solution, because the tracing process starts in row n of the table and moves up 1 row at each step.

Given a Knapsack of max. capacity W and 'n' number of items each having some weight and value such as $w_1, w_2, w_3, \dots, w_n$ and $v_1, v_2, v_3, \dots, v_n$.

Our aim is to choose those elements such that total weight is less than or equal to max. weight (W) and also gaining the max. profit.

ALGORITHM:

```
void knapsack(W, n, w, V)
int i, w;
int V[n+1][W+1]; // matrix having n+1 rows and W+1 columns
```


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
for w = 0 to W
V[0, w] = 0
for i = 0 to n
V[i, 0] = 0
for i = 1 to n
for w = 1 to W
    if w[i] ≤ w
        then V[i, w] = max ( V[i-1, w], V[i] + V[i-1, w-w[i] ] )
    else
        V[i, w] = V[i-1, w]
return V[n, W]
```

LAB VIVA QUESTIONS:

1. Define knapsack problem.
2. Define principle of optimality.
3. What is the optimal solution for knapsack problem?
4. What is the time complexity of knapsack problem?


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 13

OBJECTIVE

Write programs to implement Travelling Salesman problem using Dynamic programming.

PROGRAM LOGIC:

Theory:

The Traveling Salesman Problem (TSP) is a deceptively simple combinatorial problem. It can be stated very simply: A salesman spends his time visiting n cities (or nodes) cyclically. In one tour he visits each city just once, and finishes up where he started. In what order should he visit them to minimize the distance traveled?

Many TSP's are symmetric - that is, for any two cities A and B, the distance from A to B is the same as that from B to A. In this case you will get exactly the same tour length if you reverse the order in which they are visited - so there is no need to distinguish between a tour and its reverse, and you can leave off the arrows on the tour diagram.

If there are only 2 cities then the problem is trivial, since only one tour is possible. For the symmetric case a 3 city TSP is also trivial. If all links are present then there are $(n-1)!$ Different tours for an n city asymmetric TSP. To see why this is so, pick any city as the first - then there are $n-1$ choices for the second city visited, $n-2$ choices for the third, and so on. For the symmetric case there are half as many distinct solutions - $(n-1)!/2$ for an n city TSP. In either case the number of solutions becomes extremely large for large n , so that an exhaustive search is impractical.


Basic Steps -

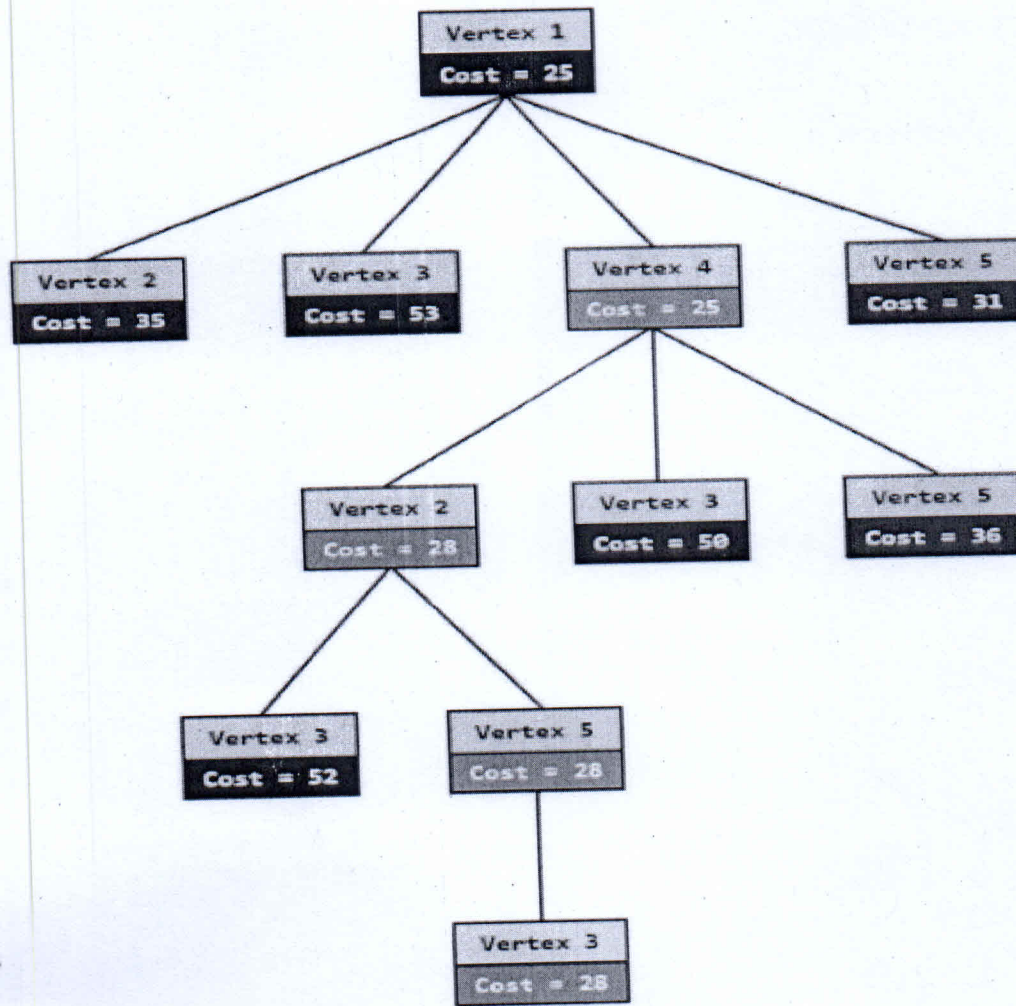
Let $G(V, E)$ be a direct graph defining an instance of the TSP.

1. This graph is first represented by a cost matrix where
$$c_{ij} = \begin{cases} \text{the cost of edge, if there is a path between } i \text{ and } j \\ \infty & \text{if there is no path} \end{cases}$$
2. Convert cost matrix into reduced matrix i.e every row and column should contain atleast one '0' entry.
3. Cost of the reduced matrix is the sum of elements that are subtracted from rows and columns of cost matrix to make it reduced.
4. Make the state space tree for reduced matrix.
5. To find the next E-node, find the least cost valued node by calculating the reduced cost matrix with every node.
6. If $\langle i, j \rangle$ edge is to be included, then there are 3 conditions to accomplish this task:
 - i. Change all entries in row i and column j of A to ∞
 - ii. Set $A[j, i] = \infty$ (**i is the root node**)
 - iii. Reduce all rows and columns in resulting matrix except for rows and columns containing ∞
7. Calculate the cost of the matrix where
$$\text{cost} = L + \text{cost}(i, j) + r$$
where L = cost of original reduced cost matrix, and
 r = new reduced cost matrix

Repeat the above steps for all the nodes until all the nodes are generated and we get a path.

Example:


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



Thus,
Optimal path is:
 $1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3$
Cost of Optimal path = 28 units

LAB VIVA QUESTIONS:

1. Define dynamic programming.
2. What is memorization in dynamic programming?
3. What are the uses of dynamic programming?


Dr. Somesh Kumar
 Prof. & Head, CSE
 Moradabad Institute of Technology
 Moradabad-244001

Program No. 14

OBJECTIVE:

Write a Program to Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.

PROGRAM LOGIC:

Kruskal's Algorithm: This is a greedy algorithm. A greedy algorithm chooses some local optimum (ie. picking an edge with the least weight in a MST). Kruskal's algorithm works as follows: Take a graph with 'n' vertices, keep adding the shortest (least cost) edge, while avoiding the creation of cycles, until $(n - 1)$ edges have been added. (NOTE: Sometimes two or more edges may have the same cost. The order in which the edges are chosen, in this case, does not matter. Different MSTs may result, but they will all have the same total cost, which will always be the minimum cost)

The steps for implementing Kruskal's algorithm are as follows:

1. Remove all self loops and parallel edges
2. Make a forest containing all the vertices
3. Sort the edges in non decreasing order of their weight.
4. Pick smallest edge, check if it forms a cycle with spanning tree formed so far. If cycle not formed, include this edge . Else discard it.
5. Repeat step 4, until there are $(|V|-1)$ edges in spanning tree. (V is set of vertices)

ALGORITHM

$A = \emptyset$

for each vertex $v \in V[G]$

 MAKE-SET(v)

Sort the edges of $E(G)$ into non decreasing order by weight w

for each edge $(u, v) \in E(G)$ taken in non decreasing ordered by weight

 if FIND-SET(u) \neq FIND-SET(v)

$A = A \cup \{(u, v)\}$

 UNION(u, v)

return A


Note:

FIND-SET(u) and FIND-SET(v) are used to check whether cycle if created after adding edge (u,v) or not.

MAKE-SET(v) is used to create a forest from all the vertices.

LAB VIVA QUESTIONS:

1. What is the time complexity of Kruskal's algorithm.
2. Define spanning tree.
3. Define minimum cost spanning tree.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program No. 15

OBJECTIVE:

Write a Program to Implement N Queen Problem using Backtracking

PROGRAM LOGIC:

Backtracking is kind of solving a problem by trial and error. However, it is a well organized trial and error. We make sure that we never try the same thing twice. We also make sure that if the problem is finite we will eventually try all possibilities (assuming there is enough computing power to try all possibilities).

The n Queens problem: Given is a board of n by n squares. Is it possible to place n queens (that behave exactly like chess queens) on this board, without having any one of them attack any other queen?

- 1) Start in the leftmost column
- 2) If all queens are placed return true
- 3) Try all rows in the current column. Do following for every tried row.
 - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
 - b) If placing queen in [row, column] leads to a solution then return true.
 - c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
- 4) If all rows have been tried and nothing worked, return false to trigger Backtracking.

ALGORITHM

```
NQueen (k, n)
for i = 1 to n do
{
  If Place (k, i) then
  {
    x[k] = i; // place kth queen to ith column
    If (k = n) // if all queens placed
      then write (x[1: n]);
    else NQueens (k+1, n);
  }
}
Place (k, i) // k is the queen number and i is the column no.
for j = 1 to k-1 do
{
  If ((x[j] = i) or (abs(x[j] - i) = abs(j - k)))
    then return false
}
return true
}
```

Note: Place (k, i) return true if a queen can be placed in the kth row and ith column otherwise return is false. It tests both whether i is distinct from all previous values of x_1, x_2, \dots, x_{k-1} and whether there is no other queen on the same diagonal.


Dr. Somesh Kumar
Prof. & Head, CSE
Maradabad Institute of Technology
Maradabad-214001

LAB VIVA QUESTIONS:

1. Define backtracking.
2. Define live node, dead node.
3. Define implicit and explicit constraints.
4. What is the time complexity of n-queens problem.


Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001