K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: **Roll No.:**

Experiment No._____

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Study, Implementation, and Performance Comparison of Selection Sort, Insertion Sort to analyze sorting strategies.

Objective: To analyse performance of sorting methods

CO to be achieved:

CO 1 Analyze the asymptotic running time and space complexity of algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Insertion_sort**
4. **http://www.sorting-algorithms.com/insertion-sort**
5. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion_sort.html**
6. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm**
7. **http://en.wikipedia.org/wiki/Selection_sort**
8. **http://www.sorting-algorithms.com/selection-sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm**
10. **http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/SelectionCardSort/selectioncardsort.html**

Pre Lab/ Prior Concepts:

Data structures, sorting techniques.

Historical Profile:

There are various methods to sort the given list. As the size of input changes, the performance of these strategies tends to differ from each other. In such case, the priori analysis can helps the engineer to choose the best algorithm.

New Concepts to be learned:
Space complexity, time complexity, size of input, order of growth.

Topic: Sorting Algorithms

Theory: Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, 1< k ≤n, yielding k sub problems. These sub problems must be solved and then a method must be found to combine sub solutions into a solution of the whole. If the sub problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied. Often the sub problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases the reapplication of the divide-and-conquer principle is naturally expressed by a recursive algorithm. Now smaller and smaller sub problems of the same kind are generated until eventually sub problems that are small enough to be solved without splitting are produced.

Algorithm Insertion Sort

INSERTION_SORT (*A*,*n*)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR j ← 2 TO length[A]
    DO  key ← A[j]
        {Put A[j] into the sorted sequence A[1 . . j − 1]}
            i ← j − 1
            WHILE i > 0 and A[i] > key
                DO A[i +1] ← A[i]
                        i ← i − 1
             A[i + 1] ← key
```

Algorithm Selection Sort

SELECTION_SORT (A,n)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR i ← 1 TO n-1 DO
  min j ← i;
```

```
min x ← A[i]
FOR j ← i + 1 to n do
    IF A[j] < min x then
        min j ← j
        min x ← A[j]
A[min j] ← A [i]
A[i] ← min x
```

The space complexity of Insertion sort:

The space complexity of Selection sort:

Time complexity for Insertion sort:

Time complexity for selection sort:

Graphs for varying input sizes: (Insertion Sort & Selection sort)

CONCLUSION:

Batch:	**Roll No.:**
Experiment No._____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation, and Comparative Analysis of Strassen's matrix multiplication.

Objective: To learn the divide and conquer strategy of solving the problems of different types

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Binary_search_algorithm**
4. **https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Binary_search_algorithm.html**
5. **http://video.franklin.edu/Franklin/Math/170/common/mod01/binarySearchAlg.html**
6. **http://xlinux.nist.gov/dads/HTML/binarySearch.html**
7. **https://www.cs.auckland.ac.nz/software/AlgAnim/searching.html**

Pre Lab/ Prior Concepts:

Data structures

Historical Profile:

Strassen's Algorithm is a groundbreaking algorithm in computer science and mathematics that introduced a faster method for matrix multiplication compared to the traditional method. It has a rich history, being one of the first major breakthroughs in computational complexity for matrix operations.Matrix multiplication is a fundamental operation in linear algebra with applications in computer graphics, scientific computing, machine learning, and more.Strassen's

algorithm is an advanced technique for matrix multiplication, introduced by Volker Strassen in 1969, which significantly improves the time complexity of traditional matrix multiplication algorithms.

Traditional Matrix Multiplication:
Complexity: O(n3) for multiplying two n×n matrices using the standard algorithm.

Strassen's Matrix Multiplication:Reduces the number of multiplications required in the divide-and-conquer approach from 8 to 7.Complexity: Approximately $O(n^{2.81})$.

New Concepts to be learned:
Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.

Algorithm :

Input : Two n×n matrices A and B, where n is a power of 2 (if not, pad the matrices with zeros).

Step 1: Divide the Matrices

A=[A11 A12

A21 A22] ,

B=[B11 B12

B21 B22]

Step 2: Compute Seven Intermediate Products

Define seven products based on specific combinations of additions and subtractions of submatrices:

1. M1=(A11+A22)*(B11+B22)
2. M2=(A21+A22)*B11
3. M3=A11*(B12−B22)
4. M4=A22*(B21−B11)
5. M5=(A11+A12)*B22
6. M6=(A21−A11)*(B11+B12)
7. M7=(A12−A22)*(B21+B22)

Step 3: Combine Results. (Use the seven intermediate products to compute the resulting matrix C)

C=[C11 C21

C12 C22] where:

- C11=M1+M4
- C12=M3+M5
- C21=M2+M4
- C22=M1−M2+M3+M6

The space complexity:

The Time complexity:

CONCLUSION:

Batch:	**Roll No.:**
Experiment No._____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation, and Comparative Analysis of Merge Sort and Quick Sort.

Objective: To learn the divide and conquer strategy of solving the problems of different types

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyze Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Quicksort**
4. **https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html**
5. **http://www.cs.rochester.edu/~gildea/csc282/slides/C07-quicksort.pdf**
6. **http://www.sorting-algorithms.com/quick-sort**
7. **http://www.cse.ust.hk/~dekai/271/notes/L01a/quickSort.pdf**
8. **http://en.wikipedia.org/wiki/Merge_sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm**
10. **http://www.sorting-algorithms.com/merge-sort**
11. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Merge_sort.html**

Pre Lab/ Prior Concepts:

Data structures, various sorting techniques

Historical Profile:

Quicksort and merge sort are divide-and-conquer sorting algorithm in which division is dynamically carried out. They are one the most efficient sorting algorithms.

New Concepts to be learned:
Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving vs Divide-and-Conquer problem solving.

Algorithm Recursive Quick Sort:

```
void quicksort( Integer A[ ], Integer left, Integer right)
//sorts A[left.. right] by using partition() to partition A[left.. right], and then //calling itself //
twice to sort the two subarrays.
{ IF ( left < right ) then
        {       q = partition( A, left, right);
                quicksort( A, left, q–1);
                quicksort( A, q+1, right);

        }

}
```

Integer *partition(integer A*T[], Integer *left*, Integer *right*)
//This function rarranges *A*[*left*..*right*] and finds and returns an integer *q*, such that *A*[*left*], ..., //*A*[*q*–1] <~ *pivot*, *A*[*q*] = *pivot*, *A*[*q*+1], ..., *A*[*right*] > *pivot*, where *pivot* is the first element of //a[left…right], before partitioning**.**

```
{
pivot = A[left]; lo = left+1; hi = right;
WHILE ( lo ≤ hi)
{       WHILE (A[hi] > pivot)                          hi = hi – 1;
        WHILE ( lo ≤ hi and A[lo] <~pivot)             lo = lo + 1;
        IF ( lo ≤ hi) then                             swap( A[lo], A[hi]);
}
swap(pivot, A[hi]);
 RETURN hi;

}
```

The space complexity of Quick Sort:

Derivation of best case and worst-case time complexity (Quick Sort)

Algorithm Merge Sort

MERGE-SORT (*A*, *p*, *r*)

// To sort the entire sequence A[1 .. n], make the initial call to the procedure MERGE-SORT (*A*, //1, *n*). Array *A* and indices *p*, *q*, *r* such that $p \leq q \leq r$ and sub array *A*[*p* .. *q*] is sorted and sub array //*A*[*q* + 1 .. *r*] is sorted. By restrictions on *p*, *q*, *r*, neither sub array is empty.

//OUTPUT: The two sub arrays are merged into a single sorted sub array in *A*[*p* .. *r*].

```
IF p < r                                  // Check for base case
   THEN q = FLOOR [(p + r)/2]               // Divide step
        MERGE (A, p, q)                     // Conquer step.
        MERGE (A, q + 1, r)                 // Conquer step.
        MERGE (A, p, q, r)                  // Conquer step.
```

```
MERGE (A, p, q, r)
{
   n1 ← q − p + 1
   n2 ← r − q
   Create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
   FOR i ← 1 TO n1
       DO L[i] ← A[p + i − 1]
    FOR j ← 1 TO n2
       DO R[j] ← A[q + j ]
   L[n1 + 1] ← ∞
   R[n2 + 1] ← ∞
  i ← 1
  j ← 1
  FOR k ← p TO r
     DO IF L[i ] ≤ R[ j]
          THEN A[k] ← L[i]
               i ← i + 1
          ELSE A[k] ← R[j]
               j ← j + 1

}
```

The space complexity of Merge sort:

Derivation of best case and worst-case time complexity (Merge Sort)

Example for quicksort/Merge tree for merge sort:

CONCLUSION:

Batch:	**Roll No.:**
Experiment No._____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation and Analysis of Single Source Shortest Path

Objective: To learn the Greedy strategy of solving the problems for different types of problems

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **1. Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **https://www.mpi-inf.mpg.de/~mehlhorn/ftp/ShortestPathSeparator.pdf**
4. **en.wikipedia.org/wiki/Shortest_path_problem**
5. **www.cs.princeton.edu/~rs/AlgsDS07/15ShortestPaths.pdf**

Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

Historical Profile:

Sometimes the problems have more than one solution. With the size of the problem, every time it's not feasible to solve all the alternative solutions and choose a better one. The greedy algorithms aim at choosing a greedy strategy as solutioning method and proves how the greedy solution is better one.

Though greedy algorithms do not guarantee optimal solution, they generally give a better and feasible solution.

The path finding algorithms work on graphs as input and represent various problems in the real world.

New Concepts to be learned: Application of algorithmic design strategy to any problem, Greedy method of problem solving Vs other methods of problem solving, optimality of the solution

Topic: GREEDY METHOD

Theory: The greedy method suggests that one can devise an algorithm that work in stages, considering one input at a time. At each stage, a decision is made regarding whether a particular input is in an optimal solution. This is done by considering the inputs in an order determined by some selection procedure. If the inclusion of the next input into the partially constructed optimal solution will result in an infeasible solution, then this input is not added to the partial solution. Otherwise, it is added. The selection procedure itself is based on some optimization measures may be plausible for a given problem. Most of these, however, will result in algorithms that generate suboptimal solutions. This version of the greedy technique is called the **subset paradigm**.

Control Abstraction:

```
SolType Greedy (Type s [], int n)

// a[1:n] contains the n inputs.

{SolType solution = EMPTY;
        // Initialize the solution.
        For (int i=1; I<=n; i++) {
                Type x = Select (a);
        If Feasible (solution, x)
        Solution = Union (solution, x) ;

}
return solution;
}
```

Algorithm:

```
1   Algorithm ShortestPaths(v, cost, dist, n)
2   // dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
3   // path from vertex v to vertex j in a digraph G with n
4   // vertices. dist[v] is set to zero. G is represented by its
5   // cost adjacency matrix cost[1 : n, 1 : n].
6   {
7       for i := 1 to n do
8       { // Initialize S.
9           S[i] := false; dist[i] := cost[v, i];
10      }
11      S[v] := true; dist[v] := 0.0; // Put v in S.
12      for num := 2 to n − 1 do
13      {
14          // Determine n − 1 paths from v.
15          Choose u from among those vertices not
16          in S such that dist[u] is minimum;
17          S[u] := true; // Put u in S.
18          for (each w adjacent to u with S[w] = false) do
19              // Update distances.
20              if (dist[w] > dist[u] + cost[u, w])) then
21                  dist[w] := dist[u] + cost[u, w];
22      }
23  }
```

Example Graph:

Solution:

Time Complexity for single source shortest path

Conclusion:

Batch:	**Roll No.:**
Experiment No._____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation, and Analysis of Job Sequencing with Deadlines.

Objective: To learn the Greedy strategy of solving the problems for different types of problems

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://lcm.csa.iisc.ernet.in/dsa/node184.htm**
4. **http://students.ceid.upatras.gr/~papagel/project/kruskal.htm**
5. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/kruskalAlgor.html**
6. **http://lcm.csa.iisc.ernet.in/dsa/node183.html**
7. **http://students.ceid.upatras.gr/~papagel/project/prim.htm**
8. **http://www.cse.ust.hk/~dekai/271/notes/L07/L07.pdf**

Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

Historical Profile:The Job Sequencing with Deadlines problem is a classic optimization problem where the goal is to schedule jobs to maximize profit, ensuring each job is completed before its deadline. The algorithm follows a greedy approach.The Job Sequencing with Deadlines Problem is a foundational optimization problem in scheduling theory and computer science. Its historical development is closely tied to advancements in algorithm design, combinatorial optimization, and practical applications in operations research and industrial processes.

Origins and Early Development

Scheduling Problems in Industry (Mid-20th Century):The problem emerged during the industrial revolution and later in the mid-20th century with the rise of operational research.Scheduling tasks to maximize efficiency, minimize delays, or maximize profits became critical for industries like manufacturing and logistics.

Development of Mathematical Formulation:Researchers began formalizing scheduling problems, including constraints like deadlines and profit maximization, laying the groundwork for job sequencing problems.The focus shifted from heuristic or ad-hoc methods to formal algorithmic solutions.

New Concepts to be learned:
Application of algorithmic design strategy to any problem, Greedy method of problem solving Vs other methods of problem solving, optimality of the solution, knapsack problem and their applications

---

Algorithm:

Input: A set of n jobs, where each job i has:

1. pi: Profit if the job is completed.
2. di: Deadline (maximum time slot by which it should be completed).

Output: A schedule of jobs such that:

1. Each job is completed within its deadline.
2. The profit is maximized.

Step 1: Sort Jobs by Profit. Sort all jobs in descending order of their profit pi. Jobs with higher profits are prioritized.

Step 2: Allocate Time Slots

1. Iterate through the sorted jobs.
2. For each job: Find the latest available time slot $t \leq di$ (the deadline of the job).If a time slot is available, schedule the job in that slot and update the slot as occupied.

Step 3: Output the Job Schedule.(Return the scheduled jobs and the total profit.)

Example:

Analysis of algorithm:

Conclusion:

Batch:	**Roll No.:**
Experiment No._7_	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation and Analysis of All Pair Shortest Path.

Objective To learn the All-Pair Shortest Path using Floyd-Warshall's algorithm

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://users.cecs.anu.edu.au/~Alistair.Rendell/Teaching/apac_comp3600/module4/all_pairs_shortest_paths.xhtml**
4. **https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/**
5. **http://www.cs.bilkent.edu.tr/~atat/502/AllPairsSP.ppt**

Theory:

It aims to figure out the shortest path from each vertex v to every other u.

1. In all pair shortest path, when a weighted graph is represented by its weight matrix W then objective is to find the distance between every pair of nodes.
2. Apply dynamic programming to solve the all pairs shortest path.
3. In all pair shortest path algorithm, we first decomposed the given problem into sub problems.
4. In this principle of optimally is used for solving the problem.
5. It means any sub path of shortest path is a shortest path between the end nodes.

Algorithm:

```
Algorithm All_pair(W, A)
{
For i = 1 to n do
For j = 1 to n do
A [i , j] = W [i , j]
For k = 1 to n do
        {
        For i = 1 to n do
                {
                For j = 1 to n do
                {
                A [i , j] = min(A [i, j], A [i, k] + A [k, j])
                }
                }}}
```

Example :

Solution for the example:

Analysis of algorithm:

CONCLUSION:

Batch:	**Roll No.:**
Experiment No._____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title:Study, Implementation, and Analysis of 0/1 Knapsack Problem.

Objective: To learn 0/1 Knapsack Problem using Dynamic Programming Approach. (Maximize the total value of selected items while ensuring their total weight does not exceed W.)

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf**
4. **http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/**
5. **http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf**
6. **https://class.coursera.org/algo2-2012-001/lecture/181**
7. **http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming**
8. **www.cse.hcmut.edu.vn/~dtanh/download/Appendix_B_2.ppt**
9. **www.ms.unimelb.edu.au/~s620261/powerpoint/chapter9_4.ppt**

Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

Historical Profile:

Dynamic Programming (DP) is used heavily in optimization problems (finding the maximum and the minimum of something). Applications range from financial models and operation research to biology and basic algorithm research. So the good news is that

understanding DP is profitable. However, the bad news is that DP is not an algorithm or a data structure that you can memorize. It is a powerful algorithmic design technique.

The 0/1 Knapsack Problem is one of the most studied problems in computer science, operations research, and combinatorial optimization. Its origins and development reflect the evolution of mathematical optimization and computational techniques.

Historical Background

Origins in Resource Allocation:The knapsack problem has its roots in resource allocation problems dating back centuries, where individuals needed to maximize their gain (value) from limited resources (capacity).The term "knapsack" derives from the idea of filling a knapsack with items to achieve the greatest benefit without exceeding its weight limit.

Early Formalization: The problem was first mathematically formalized in the early 20th century as part of broader studies in combinatorics and optimization. It gained attention as a theoretical challenge in discrete mathematics.

Greedy Algorithms and Limitations:Researchers also explored greedy approaches for simplified variants (e.g., fractional knapsack).The greedy algorithm does not work for the 0/1 knapsack problem due to its inability to handle binary choices optimally.

New Concepts to be learned:

Application of algorithmic design strategy to any problem, dynamic Programming method of problem solving Vs other methods of problem solving, optimality of the solution, Optimal Binary Search Tree Problems and their applications

---

Theory:

Algorithm:

Example:

Solution for the example:

Analysis of algorithm:

CONCLUSION:

Topic: Backtracking

Theory: In many applications of the backtrack method, the desired solution is expressible as an n-tuple *(x1,...,Xn),* where the x*i* are chosen from some finite set Si. Often the problem to be solved calls for finding one vector that maximizes (or minimizes or satisfies) a *criterion function P(x1,..... , xn). Sometime*s it seeks all vectors that satisfy *P.* For example, sorting the array of integers in. *a[1* : n] is a problem whose solution is expressible by an *n- tuple, w*here x*i* is the index in *a* of the ith smallest element. The criterion function P is the inequality *a[xi]* ≤ *a[xi+1]* for l ≤ i < *n.* The set *Si* is finite and includes the integers 1 through *n.* Though sorting is not usually one of the problems solved by backtracking, it is one example of a familiar problem whose solution can be formulated as an n-tuple.

Control abstraction:

```
void Backtrack( int k )

// This is a schema that describes the backtracking process //using recursion. On entering, the first k-1 values x[1], x[2], //…., x[k-1] of the solution vector  x[1:n] have been //assigned. x[] and n are global.
{
        for (each x[k] such that x[k] Є T(x[1], …, x[k-1])
    {

            if (Bk  (x[1], x[2], …, x[k]))
             {

                if (x[1], x[2], …, x[k] is a path to an answer node)

                            output x[1:k];

                if (k < n) Backtrack(k+1);
             }
        }

    }
```

Batch: B3 Roll No.: 1611124

Experiment No. _____

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Study, Implementation, and Analysis of Graph Coloring Problem.

Objective: To learn the Backtracking strategy of problem solving for Graph Coloring Problem.

CO to be achieved:

CO 2 Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**

Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

Historical Profile:The Graph Coloring Problem is a classical problem in graph theory and combinatorics with origins rooted in practical applications and mathematical curiosity. It has a rich history, spanning over two centuries, and remains a vibrant area of research due to its theoretical significance and real-world applications.

Origins and Early History

Map Coloring and the Four Color Theorem (1852):The problem of graph coloring originated from an attempt to color regions on maps so that no two adjacent regions share the same color. In 1852, Francis Guthrie, a British mathematician, conjectured the Four Color Theorem, stating that four colors are sufficient to color any map in a plane.

Graph Representation of Maps:In 1879, Arthur Cayley formulated the map coloring problem in terms of graph theory, representing regions of a map as vertices and adjacency as edges.

New Concepts to be learned:

Algorithm:

Backtracking Algorithm

The backtracking approach finds the optimal solution by trying all possible color assignments.

Steps:

1. **Try assigning each vertex a color from 1 to k, where k is the number of colors.**
2. **Backtrack if an assignment leads to a conflict (two adjacent vertices having the same color).**
3. **Continue until all vertices are colored or all possibilities are exhausted.**

Implementation(Code):

Output:

Example sum of subset Problem along with state space tree:

Analysis of Backtracking solution for :

Conclusion:

Batch: B3 Roll No.: 1611124 **Experiment No. ___8___** **Grade: AA / AB / BB / BC / CC / CD /DD**   **Signature of the Staff In-charge with date**

Title: Study, Implementation and Analysis of 8-Queens problem.

Objective: To learn the Backtracking strategy of problem solving for 8-Queens problem

CO to be achieved:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. **http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**
5. **http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking**
6. **http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**
7. **http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:

The **N-Queens puzzle** is the problem of placing N queens on an N×N chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

New Concepts to be learned:

Application of algorithmic design strategy to any problem, Backtracking method of problem-solving Vs other methods of problem solving, 8- Queens problem and its applications.

Algorithm N Queens Problem: -

```
void NQueens(int k, int n)
// Using backtracking, this procedure prints all possible placements of n queens on an n X n
chessboard so that they are nonattacking.
{       for (int i=1; i<=n; i++)
      {
            if (Place(k, i))
             {
               x[k] = i;
               if (k==n)
                        for (int j=1;j<=n;j++)          Print  x[j] ;
               else NQueens(k+1, n);
             }
        }
}

Boolean Place(int k, int i)
// Returns true if a queen can be placed in k^th row and i^th column.  Otherwise it returns false.
// x[] is a global array whose first (k-1) values have been set. abs(r) returns absolute value of
r.
{
for (int j=1; j < k; j++)
        if ((x[j] == i)  // Two in the same column
      || (abs(x[j]-i) == abs(j-k)))                 // or in the same diagonal
          return(false);
return(true);
 }
```

Example 8-Queens Problem:

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other i.e. no two queens share the same row, column, or diagonal.

Solution Using Backtracking Approach:

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

State Space tree for N-Queens (Solution):

Implementation (Code):

OUTPUT:

Algorithm:

Analysis of Backtracking solution:

CONCLUSION:

Batch: **Roll No.:**

Experiment No. ____

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Study, Implementation, and Analysis of the Longest Common Subsequence Algorithm.

Objective: To compute longest common subsequence for the given two strings.

CO to be achieved:

CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.
CO 3	Analyze and solve problems for different string matching algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:
Given 2 sequences, $X = x1, ..., xm$ and $Y = y1, ..., yn$, find a subsequence common to both whose length is longest. A subsequence doesn't have to be consecutive, but it has to be in order.

New Concepts to be learned:
String matching algorithm, Dynamic programming approach for LCS, Applications of LCS.

Recursive **Formulation:**
Define $c[i, j]$ = length of LCS of Xi and $Y j$.

Final answer will be computed with $c[m, n]$.

c[i, j]= 0
if i=0 or j=0.
c[i, j]= c[i − 1, j − 1] + 1
if i,j>0 and xi=yj

c[i, j]= max(c[i − 1, j], c[i, j − 1])
if i, j > 0 and x_i <> y_j

Algorithm: Longest Common Subsequence

Compute length of optimal solution-

LCS-LENGTH *(X , Y, m, n)*
for $i \leftarrow 1$ **to** m
do $c[i, 0] \leftarrow 0$
for $j \leftarrow 0$ **to** n
do $c[0, j] \leftarrow 0$
for $i \leftarrow 1$ **to** m
do for $j \leftarrow 1$ **to** n
do if $xi = y\,j$
then $c[i, j] \leftarrow c[i - 1, j - 1] + 1$
$b[i, j] \leftarrow$ "≈"
else if $c[i - 1, j] \geq c[i, j - 1]$
then $c[i, j] \leftarrow c[i - 1, j]$
$b[i, j] \leftarrow$ "↑"
else $c[i, j] \leftarrow c[i, j - 1]$
$b[i, j] \leftarrow$ "←"
return *c* and *b*

Print the solution-
PRINT-LCS*(b, X , i, j)*
if $i = 0$ or $j = 0$
then return
if $b[i, j] =$ "≈"
then PRINT-LCS*(b, X , i − 1, j − 1)*
print *xi*
elseif $b[i, j] =$ "↑"
then PRINT-LCS*(b, X , i − 1, j)*
else PRINT-LCS*(b, X , i, j − 1)*

Initial call is PRINT-LCS*(b, X , m, n).*
$b[i, j]$ points to table entry whose subproblem we used in solving LCS of *Xi* and *Y j*.
When $b[i, j] =$ ≈, we have extended LCS by one character. So longest

com− mon subsequence = entries with ≈ in them.

Example: LCS computation

Analysis of LCS computation

Output:

Algorithm:

CONCLUSION: