**International Institute of Professional Studies**

**Devi Ahilya University, Indore**

**Integrated M.Tech. and M.C.A. Programme**

**Algorithm Analysis and Design**

**Dr. Shaligram Prajapat**

[*https://sites.google.com/site/shaligramiipsdavvindore/Core-Activities/analysis-and-design-of-algorithms/*](https://sites.google.com/site/shaligramiipsdavvindore/Core-Activities/analysis-and-design-of-algorithms/)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Course Type: - Core Compulsory**

**Course Credits: - 4 Theory 0 Practical**

##### 

The study of **"Algorithm Analysis and Design"**, covers various topics such as:

* Introduction to Algorithms
* Divide and Conquer
* Dynamic Programming
* Greedy Algorithms
* Graph Algorithms
* NP-Completeness, and
* Advanced Topics.

The learning outcomes for each topic include understanding the basic concepts, identifying and implementing algorithms, analyzing the complexity of algorithms, and applying them to real-world problems. The **\*\*\* assignments and projects** involve implementing, analyzing, and developing algorithms for specific problems, evaluating their efficiency and effectiveness, and comparing and contrasting different algorithmic approaches. The advanced topics cover randomized algorithms, parallel algorithms, and their applications in machine learning, cryptography, and bioinformatics.

**Course Objectives**

* To understand the importance of algorithms in computer science and their basic concepts and notation.
* To analyze the complexity of algorithms using mathematical notation and techniques.
* To understand the divide-and-conquer strategy and apply it in algorithm design.
* To implement divide-and-conquer algorithms such as binary search, merge sort, and quicksort.
* To analyze the complexity of divide-and-conquer algorithms using mathematical notation and techniques.
* To understand the dynamic programming approach and apply it in algorithm design.
* Implement dynamic programming algorithms for problems such as the Fibonacci sequence, shortest path, and knapsack problem.
* To analyze the complexity of dynamic programming algorithms using mathematical notation and techniques.
* To understand the greedy strategy and apply it in algorithm design.
* To implement greedy algorithms such as Huffman coding and minimum spanning tree.
* To analyze the complexity of greedy algorithms using mathematical notation and techniques.
* To understand graph algorithms and their applications in computer science and other fields.
* To implement graph algorithms such as depth-first search, breadth-first search, and Dijkstra's algorithm.
* To analyze the complexity of graph algorithms using mathematical notation and techniques.
* To understand the concept of NP-completeness and its importance in algorithm design.
* To identify examples of NP-complete problems such as Boolean satisfiability and the travelling salesman problem.
* To apply approximation algorithms and heuristics to deal with NP-complete problems.
* To understand the concept of randomized algorithms and their applications in computer science.
* To understand the concept of parallel algorithms and their applications in computer science.
* To understand the applications of advanced algorithms in fields such as machine learning, cryptography, and bioinformatics.

##### **Course Content with Course Outcome**

| **Unit#** | **Topics** | **Min. Duration** |
| --- | --- | --- |
| *Unit-1:Introduction to Algorithms:* | * + Overview of algorithms and their importance in computer science   + Basic algorithmic concepts , notation   + Analysis of algorithms and their complexity | 5 Hr |
| *Learning Outcome* | * + Understand the importance of algorithms in computer science.   + Identify basic algorithmic concepts and notation.   + Analyze the complexity of algorithms using mathematical notation and techniques. |  |
| *Unit-2: Divide and Conquer:* | * + Overview of the divide-and-conquer strategy   + Examples of divide-and-conquer algorithms: binary search, merge sort, quicksort, etc.   + Analysis of divide-and-conquer algorithms and their complexity | 5 Hr |
| *Learning Outcome* | * + Understand the divide-and-conquer strategy and its application in algorithm design.   + Identify and implement divide-and-conquer algorithms such as binary search, merge sort, and quicksort.   + Analyze the complexity of divide-and-conquer algorithms using mathematical notation and techniques. |  |
| *Unit-3: Dynamic Programming:* | * + Overview of the dynamic programming approach   + Examples of dynamic programming algorithms: Fibonacci sequence, shortest path, knapsack problem, etc.   + Analysis of dynamic programming algorithms and their complexity | 6 Hr |
| *Learning Outcome* | * + Understand the dynamic programming approach and its application in algorithm design.   + Identify and implement dynamic programming algorithms for problems such as the Fibonacci sequence, shortest path, and knapsack problem.   + Analyze the complexity of dynamic programming algorithms using mathematical notation and techniques. |  |
| *Unit-4: Greedy Algorithms:* | * + Overview of the Greedy strategy   + Examples of greedy algorithms: Huffman coding, minimum spanning tree, etc.   + Analysis of greedy algorithms and their complexity | 6 Hr |
| *Learning Outcome* | * + Understand the greedy strategy and its application in algorithm design.   + Identify and implement greedy algorithms such as Huffman coding and minimum spanning tree.   + Analyze the complexity of greedy algorithms using mathematical notation and techniques. |  |
| *Unit-5: Graph Algorithms:* | * + Overview of graph algorithms and their applications   + Examples of graph algorithms: depth-first search, breadth-first search, Dijkstra's algorithm, etc.   + Analysis of graph algorithms and their complexity | 6Hr |
| *Learning Outcome* | * + Understand the applications of graph algorithms in computer science and other fields.   + Identify and implement graph algorithms such as depth-first search, breadth-first search, and Dijkstra's algorithm.   + Analyze the complexity of graph algorithms using mathematical notation and techniques. |  |
| *Unit-6: NP-Completeness:* | * + Introduction to NP-completeness and its importance in algorithm design   + Examples of NP-complete problems: Boolean satisfiability, traveling salesman problem, etc.   + Techniques for dealing with NP-complete problems: approximation algorithms, heuristics, etc. | 6Hr |
| *Learning Outcome* | * + Understand the concept of NP-completeness and its importance in algorithm design.   + Identify examples of NP-complete problems such as Boolean satisfiability and the traveling salesman problem.   + Apply approximation algorithms and heuristics to deal with NP-complete problems. |  |
| *Unit-7: Advanced Topics:* | * + Introduction to advanced topics in algorithm design: Randomized algorithms, parallel algorithms, Distributed and Decentralised Approaches etc.   + Applications of advanced algorithms in areas such as machine learning, cryptography and Information Security and bioinformatics | 6 Hr |
| *Learning Outcome* | * + Understand the concept of randomized algorithms and their applications in computer science.   + Understand the concept of parallel algorithms and their applications in computer science.   + Understand the applications of advanced algorithms in fields such as machine learning, cryptography, and bioinformatics. |  |
| **Recommended References:**   * 1. "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein.   2. "Algorithm Design" by Kleinberg and Tardos.   3. "The Design of Approximation Algorithms" by Williamson and Shmoys. | | |
| **Online Courses**   * 1. NPTEL lectures on "*Design and Analysis of Algorithms*" by Prof. S. Arun Kumar.   2. NPTEL lectures on "*Advanced Algorithms*" by Prof. Sundar Vishwanathan.   3. "*Algorithms*" by Jeff Erickson. This is a comprehensive online textbook that covers all the topics included in the syllabus.   4. "*Algorithmic Thinking, Peak Finding*" by Prof. Eric Grimson and Prof. John Guttag. | | |

##### 

##### 

##### CO-PO Mapping

**Course Outcomes**

CO1 – Learn core concepts for Algorithm,design and analysis.

CO2 – Learn various approaches of algorithm design and corresponding complexity analysis.

CO3 – Understand state of art approaches of algorithm, investigate their scope for future requirements and applications.

contexts, from office productivity software to programming languages.

CO4 – Understand computability theory and decision problems.Exploring their solution using advanced techniques, processor (single , multiple, parallel) for large problem size

|  | Program Specific Outcomes | CO1 | CO2 | CO3 | CO4 |
| --- | --- | --- | --- | --- | --- |
| PSO1 | Conceptual knowledge | ✔ | ✔ | ✔ | ✔ |
| Practical skills | ✔ | ✔ | ✔ | ✔ |
| Innovative skills |  |  | ✔ | ✔ |
| Research skills |  |  | ✔ | ✔ |
| Reflective thinking |  |  |  |  |
| PSO2 | Disciplinary knowledge | ✔ | ✔ | ✔ | ✔ |
| Critical thinking | ✔ |  | ✔ | ✔ |
| Programming skills | ✔ | ✔ |  |  |
| PSO3 | Problem solving skills | ✔ | ✔ | ✔ | ✔ |
| Analytical reasoning skills | ✔ | ✔ | ✔ | ✔ |
| Research skills |  |  | ✔ | ✔ |
| PSO4 | Team work and cooperation |  |  |  |  |
| Decision making skills |  | ✔ | ✔ |  |
| Managerial competencies |  | ✔ | ✔ |  |
| Communication skills |  |  |  |  |
| Leadership qualities |  |  |  |  |
| PSO5 | Ethical awareness |  |  |  |  |
| Moral values |  |  |  |  |
| Technological ethical practices |  |  |  |  |
| PSO6 | Modern technologies |  |  | ✔ | ✔ |
| Lifelong learning |  |  | ✔ | ✔ |
| Technological knowledge | ✔ | ✔ |  |  |

##### **Unit Wise Objective, Learning Outcomes and Assignment**

##### 

##### Unit-1: Introduction to Algorithms:

##### 

##### Understand the importance of algorithms in computer science.

##### Identify basic algorithmic concepts and notation.

##### Analyze the complexity of algorithms using mathematical notation and techniques.

##### Unit-2: Divide and Conquer:

##### 

##### Understand the divide-and-conquer strategy and its application in algorithm design.

##### Identify and implement divide-and-conquer algorithms such as binary search, merge sort, and quicksort.

##### Analyze the complexity of divide-and-conquer algorithms using mathematical notation and techniques.

##### Unit-3: Dynamic Programming:

##### 

##### Understand the dynamic programming approach and its application in algorithm design.

##### Identify and implement dynamic programming algorithms for problems such as the Fibonacci sequence, shortest path, and knapsack problem.

##### Analyze the complexity of dynamic programming algorithms using mathematical notation and techniques.

##### Unit-4: Greedy Algorithms:

##### 

##### Understand the greedy strategy and its application in algorithm design.

##### Identify and implement greedy algorithms such as Huffman coding and minimum spanning tree.

##### Analyze the complexity of greedy algorithms using mathematical notation and techniques.

##### Unit-5: Graph Algorithms:

##### 

##### Understand the applications of graph algorithms in computer science and other fields.

##### Identify and implement graph algorithms such as depth-first search, breadth-first search, and Dijkstra's algorithm.

##### Analyze the complexity of graph algorithms using mathematical notation and techniques.

##### Unit-6: NP-Completeness:

##### 

##### Understand the concept of NP-completeness and its importance in algorithm design.

##### Identify examples of NP-complete problems such as Boolean satisfiability and the traveling salesman problem.

##### Apply approximation algorithms and heuristics to deal with NP-complete problems.

##### Unit-7: Advanced Topics:

##### 

##### Understand the concept of randomized algorithms and their applications in computer science.

##### Understand the concept of parallel algorithms and their applications in computer science.

##### Understand the applications of advanced algorithms in fields such

##### 

**Unit-1: Introduction to Algorithms:**

*Learning Outcomes:*

* Understand the importance of algorithms in computer science.
* Identify basic algorithmic concepts and notation.
* Analyze the complexity of algorithms using mathematical notation and techniques.

*\*\*\* Assignments / Projects:*

* Implement and analyze the efficiency of algorithms for common tasks such as searching and sorting.
* Analyze the performance of different algorithms using big-O notation and empirical testing.
* Develop algorithms for specific problems and evaluate their efficiency and effectiveness.

**Unit-2: Divide and Conquer:**

*Learning Outcomes:*

* Understand the divide-and-conquer strategy and its application in algorithm design.
* Identify and implement divide-and-conquer algorithms such as binary search, merge sort, and quicksort.
* Analyze the complexity of divide-and-conquer algorithms using mathematical notation and techniques.

*\*\*\* Assignments/Projects:*

* Implement and analyze the efficiency of divide-and-conquer algorithms for specific problems.
* Develop new divide-and-conquer algorithms for complex problems and evaluate their efficiency and effectiveness.
* Compare and contrast divide-and-conquer algorithms with other algorithmic approaches.

**Unit-3: Dynamic Programming:**

*Learning Outcomes:*

* Understand the dynamic programming approach and its application in algorithm design.
* Identify and implement dynamic programming algorithms for problems such as the Fibonacci sequence, shortest path, and knapsack problem.
* Analyze the complexity of dynamic programming algorithms using mathematical notation and techniques.

*\*\*\* Assignments and Projects:*

* Implement and analyze the efficiency of dynamic programming algorithms for specific problems.
* Develop new dynamic programming algorithms for complex problems and evaluate their efficiency and effectiveness.
* Compare and contrast dynamic programming algorithms with other algorithmic approaches.

**Unit-4: Greedy Algorithms:**

Learning Outcomes:

* Understand the greedy strategy and its application in algorithm design.
* Identify and implement greedy algorithms such as Huffman coding and minimum spanning tree.
* Analyze the complexity of greedy algorithms using mathematical notation and techniques.

\*\*\* Assignments/Projects:

* Implement and analyze the efficiency of greedy algorithms for specific problems.
* Develop new greedy algorithms for complex problems and evaluate their efficiency and effectiveness.
* Compare and contrast greedy algorithms with other algorithmic approaches.

**Unit-5: Graph Algorithms:**

*Learning Outcomes:*

* Understand the applications of graph algorithms in computer science and other fields.
* Identify and implement graph algorithms such as depth-first search, breadth-first search, and Dijkstra's algorithm.
* Analyze the complexity of graph algorithms using mathematical notation and techniques.

*\*\*\* Assignments / Projects:*

* Implement and analyze the efficiency of graph algorithms for specific problems.
* Develop new graph algorithms for complex problems and evaluate their efficiency and effectiveness.
* Apply graph algorithms to real-world problems such as network analysis and social network analysis.

**Unit-6 :NP-Completeness:**

*Learning Outcomes:*

* Understand the concept of NP-completeness and its importance in algorithm design.
* Identify examples of NP-complete problems such as Boolean satisfiability and the travelling salesman problem.
* Apply approximation algorithms and heuristics to deal with NP-complete problems.

*\*\*\* Assignments / Projects:*

* Analyze the complexity of NP-complete problems using mathematical notation and techniques.
* Develop approximation algorithms and heuristics for specific NP-complete problems.
* Evaluate the effectiveness of approximation algorithms and heuristics for specific NP-complete problems.

**Unit - 7: Advanced Topics:**

*Learning Outcomes:*

* Understand the concept of randomized algorithms and their applications in computer science.
* Understand the concept of parallel algorithms and their applications in computer science.
* Understand the applications of advanced algorithms in fields such as machine learning, cryptography, and bioinformatics.

\*\*\* Assignments /Projects:

* Implement and analyze the efficiency of randomized algorithms and parallel algorithms for specific problems.
* Apply advanced algorithms to real-world problems in fields such as machine learning, cryptography, and bioinformatics.
* Develop new advanced algorithms for complex problems and evaluate their efficiency and effectiveness.

##### Assignments : Assignment #1

**Unit-1: Introduction to Algorithms:**

• Implement and analyze the efficiency of algorithms for common tasks such as searching and sorting.

• Analyze the performance of different algorithms using big-O notation and empirical testing.

• Develop algorithms for specific problems and evaluate their efficiency and effectiveness.

**Unit-2: Divide and Conquer:**

• Implement and analyze the efficiency of divide-and-conquer algorithms for specific problems.

• Develop new divide-and-conquer algorithms for complex problems and evaluate their efficiency and effectiveness.

• Compare and contrast divide-and-conquer algorithms with other algorithmic approaches.

**Unit-3: Dynamic Programming:**

• Implement and analyze the efficiency of dynamic programming algorithms for specific problems.

• Develop new dynamic programming algorithms for complex problems and evaluate their efficiency and effectiveness.

• Compare and contrast dynamic programming algorithms with other algorithmic approaches.

**Unit-4: Greedy Algorithms:**

• Implement and analyze the efficiency of greedy algorithms for specific problems.

• Develop new greedy algorithms for complex problems and evaluate their efficiency and effectiveness.

• Compare and contrast greedy algorithms with other algorithmic approaches.

**Unit-5: Graph Algorithms:**

• Implement and analyze the efficiency of graph algorithms for specific problems.

• Develop new graph algorithms for complex problems and evaluate their efficiency and effectiveness.

• Apply graph algorithms to real-world problems such as network analysis and social network analysis.

**Unit-6 :NP-Completeness:**

• Analyze the complexity of NP-complete problems using mathematical notation and techniques.

• Develop approximation algorithms and heuristics for specific NP-complete problems.

• Evaluate the effectiveness of approximation algorithms and heuristics for specific NP-complete problems.

**Unit - 7: Advanced Topics:**

• Implement and analyze the efficiency of randomized algorithms and parallel algorithms for specific problems.

• Apply advanced algorithms to real-world problems in fields such as machine learning, cryptography, and bioinformatics.

• Develop new advanced algorithms for complex problems and evaluate their efficiency and effectiveness.

## Programming Assignments : Assignment

**Unit-1: Introduction to Algorithms:**

Programming exercises related to implementing and analyzing the efficiency of algorithms for common tasks such as searching and sorting, analyzing the performance of different algorithms using big-O notation and empirical testing, and developing algorithms for specific problems and evaluating their efficiency and effectiveness.

###### C Programming Exercises:

1. Implement a binary search algorithm for an array of integers.

2. Implement a bubble sort algorithm for an array of integers.

3. Implement a selection sort algorithm for an array of integers.

4. Implement an insertion sort algorithm for an array of integers.

5. Implement a merge sort algorithm for an array of integers.

6. Implement a quick sort algorithm for an array of integers.

7. Implement a linear search algorithm for an array of integers.

8. Implement a hash table data structure using separate chaining collision resolution.

9. Implement a hash table data structure using linear probing collision resolution.

10. Implement a heap data structure and use it to sort an array of integers.

###### Python Programming Exercises:

11. Implement a binary search algorithm for a list of integers.

12. Implement a bubble sort algorithm for a list of integers.

13. Implement a selection sort algorithm for a list of integers.

14. Implement an insertion sort algorithm for a list of integers.

15. Implement a merge sort algorithm for a list of integers.

16. Implement a quick sort algorithm for a list of integers.

17. Implement a linear search algorithm for a list of integers.

18. Implement a hash table data structure using separate chaining collision resolution.

19. Implement a hash table data structure using linear probing collision resolution.

20. Implement a heap data structure and use it to sort a list of integers.

###### Flowchart Exercises:

21. Create a flowchart for a binary search algorithm.

22. Create a flowchart for a bubble sort algorithm.

23. Create a flowchart for a selection sort algorithm.

24. Create a flowchart for an insertion sort algorithm.

25. Create a flowchart for a merge sort algorithm.

26. Create a flowchart for a quick sort algorithm.

27. Create a flowchart for a linear search algorithm.

28. Create a flowchart for a hash table data structure using separate chaining collision resolution.

29. Create a flowchart for a hash table data structure using linear probing collision resolution.

30. Create a flowchart for a heap data structure and use it to sort an array of integers.

These exercises can help you practice and develop your skills in implementing and analyzing the efficiency of algorithms for common tasks such as searching and sorting, analyzing the performance of different algorithms using big-O notation and empirical testing, and developing algorithms for specific problems and evaluating their efficiency and effectiveness.

**Unit-2: Divide and Conquer:**

Complete following programming exercises related to implementing and analyzing the efficiency of divide-and-conquer algorithms for specific problems, developing new divide-and-conquer algorithms for complex problems and evaluating their efficiency and effectiveness, and comparing and contrasting divide-and-conquer algorithms with other algorithmic approaches.

###### C Programming Exercises:

1. Implement a recursive algorithm for computing the factorial of a given positive integer using divide-and-conquer.

2. Implement a recursive algorithm for computing the Fibonacci sequence using divide-and-conquer.

3. Implement a recursive algorithm for computing the greatest common divisor of two integers using divide-and-conquer.

4. Implement a recursive algorithm for computing the Euclidean distance between two points in 2D space using divide-and-conquer.

5. Implement a divide-and-conquer algorithm for finding the maximum subarray sum of a given array of integers.

6. Implement a divide-and-conquer algorithm for merging two sorted arrays.

7. Implement a divide-and-conquer algorithm for finding the kth largest element in an unsorted array of integers.

8. Implement a divide-and-conquer algorithm for multiplying two large integers.

9. Implement a divide-and-conquer algorithm for finding the closest pair of points in 2D space.

10. Implement a divide-and-conquer algorithm for sorting a linked list.

###### Python Programming Exercises:

11. Implement a recursive algorithm for computing the factorial of a given positive integer using divide-and-conquer.

12. Implement a recursive algorithm for computing the Fibonacci sequence using divide-and-conquer.

13. Implement a recursive algorithm for computing the greatest common divisor of two integers using divide-and-conquer.

14. Implement a recursive algorithm for computing the Euclidean distance between two points in 2D space using divide-and-conquer.

15. Implement a divide-and-conquer algorithm for finding the maximum subarray sum of a given array of integers.

16. Implement a divide-and-conquer algorithm for merging two sorted arrays.

17. Implement a divide-and-conquer algorithm for finding the kth largest element in an unsorted array of integers.

18. Implement a divide-and-conquer algorithm for multiplying two large integers.

19. Implement a divide-and-conquer algorithm for finding the closest pair of points in 2D space.

20. Implement a divide-and-conquer algorithm for sorting a linked list.

###### Flowchart Exercises:

21. Create a flowchart for a recursive algorithm for computing the factorial of a given positive integer using divide-and-conquer.

22. Create a flowchart for a recursive algorithm for computing the Fibonacci sequence using divide-and-conquer.

23. Create a flowchart for a recursive algorithm for computing the greatest common divisor of two integers using divide-and-conquer.

24. Create a flowchart for a recursive algorithm for computing the Euclidean distance between two points in 2D space using divide-and-conquer.

25. Create a flowchart for a divide-and-conquer algorithm for finding the maximum subarray sum of a given array of integers.

26. Create a flowchart for a divide-and-conquer algorithm for merging two sorted arrays.

27. Create a flowchart for a divide-and-conquer algorithm for finding the kth largest element in an unsorted array of integers.

28. Create a flowchart for a divide-and-conquer algorithm for multiplying two large integers.

29. Create a flowchart for a divide-and-conquer algorithm for finding the closest pair of points in 2D space.

30. Create a flowchart for a divide-and-conquer algorithm for sorting a linked list.

These exercises can help you practice and develop your skills in implementing and analyzing the efficiency of divide-and-conquer algorithms for specific problems, developing new divide-and-conquer algorithms for complex problems and evaluating their efficiency and effectiveness, and comparing and contrasting divide-and-conquer algorithms with other algorithmic approaches.

**Unit-3: Dynamic Programming:**

###### C/Python /Flowchart

31. Implement the Fibonacci sequence using dynamic programming in C or Python. Analyze the time complexity of your implementation and compare it with the time complexity of a recursive implementation.

32. Implement the longest common subsequence problem using dynamic programming in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a naive recursive implementation.

33. Implement the knapsack problem using dynamic programming in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a greedy algorithm.

34. Implement the rod-cutting problem using dynamic programming in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a naive recursive implementation.

35. Implement the matrix chain multiplication problem using dynamic programming in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a naive recursive implementation.

36. Develop a dynamic programming algorithm for the edit distance problem in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a naive recursive implementation.

37. Develop a dynamic programming algorithm for the coin change problem in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a greedy algorithm.

38. Implement a dynamic programming algorithm to find the maximum sum subarray in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a naive brute-force approach.

39. Develop a dynamic programming algorithm to find the longest increasing subsequence in C or Python. Analyze the time and space complexity of your implementation and compare it with the time and space complexity of a brute-force approach.

40. Create a flowchart for a dynamic programming algorithm that solves the longest common substring problem. Analyze the time and space complexity of the algorithm and compare it with the time and space complexity of a brute-force approach.

These exercises will help you understand and implement dynamic programming algorithms, analyze their time and space complexity, and compare them with other algorithmic approaches.

**Unit-4: Greedy Algorithms:**

###### C/Python /Flowchart

41. Implement the Huffman coding algorithm to compress a given string in C or Python.

42. Implement Kruskal's algorithm to find the minimum spanning tree of a given graph in C or Python.

43. Implement the Dijkstra algorithm to find the shortest path between two vertices in a graph in C or Python.

44. Implement the activity selection problem using a greedy approach in C or Python.

45. Implement the knapsack problem using a greedy approach in C or Python.

46. Implement the coin change problem using a greedy approach in C or Python.

47. Develop a new greedy algorithm to solve the job scheduling problem and analyze its efficiency and effectiveness in C or Python.

48. Implement Prim's algorithm to find the minimum spanning tree of a given graph in flowchart.

49. Implement the interval scheduling problem using a greedy approach in flowchart.

50. Implement the fractional knapsack problem using a greedy approach in flowchart.

These exercises will help you practice implementing and analyzing the efficiency of greedy algorithms for specific problems, developing new greedy algorithms for complex problems, and comparing and contrasting greedy algorithms with other algorithmic approaches.

**Unit-5: Graph Algorithms:**

###### C/Python /Flowchart

51. Implement Breadth First Search (BFS) algorithm to find the shortest path between two nodes in a graph in C or Python.

52. Implement Depth First Search (DFS) algorithm to traverse a graph and find connected components in C or Python.

53. Implement Dijkstra's algorithm to find the shortest path between two nodes in a weighted graph in C or Python.

54. Implement Bellman-Ford algorithm to find the shortest path between two nodes in a graph with negative weight edges in C or Python.

55. Implement the Floyd-Warshall algorithm to find the shortest path between all pairs of nodes in a graph in C or Python.

56. Implement the Prim's algorithm to find the minimum spanning tree of a given graph in C or Python.

57. Develop a new graph algorithm to solve the traveling salesman problem and evaluate its efficiency and effectiveness in C or Python.

58. Implement the PageRank algorithm to rank web pages in a network in Python.

59. Implement the Girvan-Newman algorithm to find communities in a social network in Python.

60. Apply graph algorithms to analyze the network structure and connectivity of a real-world social network, such as Facebook or Twitter.

These exercises will help you practice implementing and analyzing the efficiency of graph algorithms for specific problems, developing new graph algorithms for complex problems, and applying graph algorithms to real-world problems such as network analysis and social network analysis.

**Unit-6 :NP-Completeness:**

###### C/Python /Flowchart

61. Implement a dynamic programming algorithm for the Longest Common Subsequence Problem (LCS), and analyze its time and space complexity using Big-O notation.

62. Develop a greedy algorithm for the Job Scheduling Problem, and evaluate its effectiveness on a variety of input instances.

63. Implement a branch and bound algorithm for the Traveling Salesman Problem, and analyze its running time complexity.

64. Develop an approximation algorithm for the Set Cover Problem, and analyze its approximation ratio and running time complexity.

65. Implement a genetic algorithm for the Knapsack Problem, and evaluate its performance on a set of large-scale instances.

66. Develop a local search algorithm for the Maximum Cut Problem, and compare its performance to that of a randomized algorithm on a set of random graphs.

67. Implement a randomized algorithm for the Vertex Cover Problem, and analyze its expected running time using probabilistic techniques.

68. Develop an approximation algorithm for the Steiner Tree Problem, and analyze its approximation ratio and running time complexity.

69. Implement a simulated annealing algorithm for the Graph Coloring Problem, and evaluate its performance on a set of benchmark instances.

70. Develop a heuristic algorithm for the Maximum Clique Problem, and compare its performance to that of the exact algorithm on a set of benchmark instances.

71. Implement a brute force algorithm for the Traveling Salesman Problem (TSP), and analyze its time complexity using Big-O notation.

72. Develop an approximation algorithm for the Vertex Cover Problem, and evaluate its effectiveness on a variety of input graphs.

73. Implement a greedy algorithm for the Knapsack Problem, and analyze its approximation ratio.

74. Develop a heuristic algorithm for the Maximum Clique Problem, and compare its performance to that of the exact algorithm on a set of benchmark instances.

75. Implement a randomized algorithm for the Subset Sum Problem, and analyze its expected running time using probabilistic techniques.

76. Develop a genetic algorithm for the Quadratic Assignment Problem, and evaluate its performance on a set of large-scale instances.

77. Implement a local search algorithm for the Graph Coloring Problem, and compare its performance to that of a randomized algorithm on a set of randomly generated graphs.

78. Develop an approximation algorithm for the Set Cover Problem, and analyze its approximation ratio and running time complexity.

79. Implement a simulated annealing algorithm for the Traveling Salesman Problem, and evaluate its performance on a set of benchmark instances.

80. Develop a tabu search algorithm for the Maximum Cut Problem, and compare its performance to that of a greedy algorithm on a set of random graphs.

##### **Unit-7 : Advanced Topics**

###### C/Python /Flowchart

81. Implement a parallel algorithm for matrix multiplication, and analyze its speedup and efficiency on a multicore processor or GPU.

82. Develop a randomized algorithm for the k-Median Problem, and analyze its expected running time and approximation ratio.

83. Apply a deep learning algorithm such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) to a real-world problem such as image recognition or natural language processing, and evaluate its performance on a dataset.

84. Implement a parallel algorithm for the Breadth-First Search (BFS) traversal of a graph, and analyze its speedup and efficiency on a cluster or supercomputer.

85. Develop a new algorithm for the Minimum Spanning Tree (MST) Problem, and evaluate its efficiency and effectiveness on a set of benchmark instances.

86. Apply a cryptographic algorithm such as RSA or AES to a real-world problem such as secure communication or data encryption, and analyze its security and performance.

87. Implement a parallel algorithm for the Traveling Salesman Problem, and analyze its speedup and efficiency on a distributed computing environment such as a cluster or cloud platform.

88. Develop a new algorithm for the Maximum Flow Problem, and evaluate its efficiency and effectiveness on a set of benchmark instances.

89. Apply a bioinformatics algorithm such as Smith-Waterman or Needleman-Wunsch to a real-world problem such as DNA sequencing or protein folding, and analyze its accuracy and efficiency.

90. Implement a randomized algorithm for the Minimum Cut Problem, and analyze its expected running time and approximation ratio.

For each of these exercises, it would be helpful to create a flowchart or pseudocode to help plan out the implementation before diving into the actual code. Additionally, for exercises involving real-world applications, it may be necessary to acquire or generate appropriate datasets for testing and evaluation.

##### **Additional Assignment**

Compare the (serial, parallel ) algorithms implementation on (single core, multi core) processors CPU and GPU and plot the performance on various input size for following data structures:

1. Arrays
2. Linked Lists
3. Stacks
4. Queues
5. Trees
6. Hash Tables
7. Graphs
8. Heaps
9. Tries
10. Sets
11. Maps
12. Vectors
13. Matrices
14. Doubly Linked Lists
15. Priority Queues