|  |
| --- |
| **Q.1) What is an algorithm? Explain in detail about various characteristics of an algorithm.** |
| **Ans.1)** |
| **Q.2) Explain in detail about various asymptomatic notations for analyzing algorithm.** |
| **Ans.2)** |
| **Q.3) Explain in detail Strassen’s Matrix Multiplication.** |
| **Ans.3)**  **Strassen's matrix multiplication algorithm is a divide-and-conquer algorithm for multiplying two matrices. It was first published by Volker Strassen in 1969. Strassen's algorithm is based on the observation that matrix multiplication can be broken down into smaller subproblems. The algorithm recursively solves these subproblems and then combines the solutions to obtain the final result.**  **The basic idea of Strassen's algorithm is to divide the two matrices to be multiplied into four smaller matrices of half the size. These smaller matrices are then multiplied using a combination of seven basic operations: addition, subtraction, multiplication, and division. The results of these multiplications are then combined to form the product of the original two matrices.**  **Strassen's algorithm has a time complexity of O(n log 7), which is asymptotically faster than the naive matrix multiplication algorithm, which has a time complexity of O(n^3). However, Strassen's algorithm is more complex to implement and may not be faster in practice for small matrices.**  **Here is a more detailed explanation of Strassen's algorithm:**   1. **The two matrices to be multiplied are divided into four smaller matrices of half the size.** 2. **The four smaller matrices are multiplied using the seven basic operations.** 3. **The results of the multiplications are combined to form the product of the original two matrices.**   **The following table shows the time complexity of Strassen's algorithm for different matrix sizes:**  **Code snippet**  **Matrix size | Time complexity**  **---|---**  **2x2 | O(1)**  **3x3 | O(2)**  **4x4 | O(7)**  **5x5 | O(49)**  **6x6 | O(343)**  **As you can see, the time complexity of Strassen's algorithm grows exponentially with the size of the matrices. This means that Strassen's algorithm is only practical for multiplying very small matrices.**  **In practice, the naive matrix multiplication algorithm is often faster than Strassen's algorithm for multiplying large matrices. This is because the naive algorithm is simpler to implement and can be optimized for specific hardware platforms.**  **However, Strassen's algorithm can be useful for multiplying matrices of intermediate size. In these cases, the speedup provided by Strassen's algorithm can outweigh the increased complexity of implementation.** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| **Q.4) Write the algorithm for Insertion Sort.** |
| **Ans.4)**  **The Algorithm for Insertion Sort:**    **Here is a breakdown of the algorithm:**   1. **The algorithm starts at the second element of the array.** 2. **The algorithm takes the current element and compares it to the elements before it.** 3. **If the current element is smaller than any of the elements before it, the algorithm swaps the current element with the element that is smaller than it.** 4. **The algorithm then repeats steps 2 and 3 for the next element in the array.** 5. **The algorithm continues this process until it has reached the end of the array.**   **Insertion sort is a simple sorting algorithm that is easy to understand and implement. However, it is not very efficient for large arrays. For large arrays, other sorting algorithms, such as quicksort or merge sort, are more efficient.**  **Here are some of the advantages of insertion sort:**   * **It is a simple algorithm to understand and implement.** * **It is a stable algorithm, which means that the relative order of elements with equal keys is preserved.** * **It is in-place, which means that it does not require any additional memory.**   **Here are some of the disadvantages of insertion sort:**   * **It is not very efficient for large arrays.** * **It is not a recursive algorithm, which means that it cannot be easily parallelized.** * **It is not very efficient for arrays that are already sorted.** |
| **Q.5) Derive worst case and best case run time for complexity for Insertion Sort.** |
| **Ans.5)** |
| **Q.7) State Dijkstra’s algorithm for solving Single Source multiple destination**  **shortest path problems? Illustrate your answer by stepwise execution by assuming suitable example.** |
| **Ans.7)**  **Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a weighted graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. The algorithm exists in many variants.**  **Here is the pseudocode for Dijkstra's algorithm:**  **function dijkstra(graph, source):**  **distances = {}**  **for v in graph.nodes():**  **distances[v] = float("inf")**  **distances[source] = 0**  **unvisited = set(graph.nodes())**  **while unvisited:**  **u = min(unvisited, key=distances.get)**  **unvisited.remove(u)**  **for v in graph.neighbors(u):**  **new\_distance = distances[u] + graph.edge(u, v).weight**  **if new\_distance < distances[v]:**  **distances[v] = new\_distance**  **return distances**  **Here is an example of how to use Dijkstra's algorithm to find the shortest paths from node A to all other nodes in a graph:**  **graph = {**  **"A": {"B": 5, "C": 10},**  **"B": {"A": 5, "C": 2},**  **"C": {"A": 10, "B": 2}**  **}**  **distances = dijkstra(graph, "A")**  **print(distances)**  **This will print the following output:**  **{'A': 0, 'B': 5, 'C': 10}**  **This means that the shortest path from node A to node A is 0, the shortest path from node A to node B is 5, and the shortest path from node A to node C is 10.**  **Dijkstra's algorithm is a greedy algorithm. This means that it always chooses the node that is closest to the source node. This can be inefficient for graphs with negative weights, as Dijkstra's algorithm may end up choosing a path that is not the shortest path.**  **Here is a table that summarizes the time complexity of Dijkstra's algorithm for different input conditions:**   |  |  | | --- | --- | | **Input condition** | **Time complexity** | | **Unweighted graph** | **O(** | | **Weighted graph** | **O(** | |