Implement Depth First Search, Breadth first search and Djikstra Algorithm for Graphs and also give the time complexity (O(?)) of the approached used.

```
from collections import defaultdict
import heapq

class Graph:
    def _init_(self):
        self.graph = defaultdict(list)

    def add_edge(self, u, v, weight=1):
        self.graph[u].append((v, weight))

    def dfs_util(self, vertex, visited):
        visited.add(vertex)
        print(vertex, end=" ")

        for neighbor, _ in self.graph[vertex]:
            if neighbor not in visited:
                self.dfs_util(neighbor, visited)

    def dfs(self, start):
        visited = set()
        self.dfs_util(start, visited)

    def bfs(self, start):
        visited = set()
        queue = [start]
        visited.add(start)

        while queue:
            vertex = queue.pop(0)
            print(vertex, end=" ")

            for neighbor, _ in self.graph[vertex]:
                if neighbor not in visited:
                    queue.append(neighbor)
                    visited.add(neighbor)

    def dijkstra(self, start):
        distances = {vertex: float('infinity') for vertex in self.graph}
```

```
        distances[start] = 0
        priority_queue = [(0, start)]

        while priority_queue:
            current_distance, current_vertex = heapq.heappop(priority_queue)

            if current_distance > distances[current_vertex]:
                continue

            for neighbor, weight in self.graph[current_vertex]:
                distance = current_distance + weight

                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    heapq.heappush(priority_queue, (distance, neighbor))

        return distances


g = Graph()
g.add_edge('A', 'B', 5)
g.add_edge('A', 'C', 3)
g.add_edge('B', 'D', 2)
g.add_edge('C', 'D', 1)
g.add_edge('C', 'E', 6)
g.add_edge('D', 'E', 4)

print("Depth First Search:")
g.dfs('A')
print("\nBreadth First Search:")
g.bfs('A')
print("\nDijkstra's Algorithm:")
print(g.dijkstra('A'))
```

Time complexities:

- Depth First Search (DFS): O(V + E), where V is the number of vertices and E is the number of edges.

- Breadth First Search (BFS): O(V + E), where V is the number of vertices and E is the number of edges.

- Dijkstra's Algorithm: O((V + E) * log V) using a priority queue implementation with a binary heap. In the worst case, each vertex and edge will be added to and extracted from the priority queue once, where heap operations take O(log V) time.