Algorithm :

1. **File Reading and Graph Construction:**
   Read a CSV file containing edge information.
   Construct an adjacency list representation of the graph.
2. **Dijkstra's Algorithm:**

```
function Dijkstra(Graph, src):
   dist[] := array of distances initialized to infinity, size V (number of vertices)
   parent[] := array of parents initialized to -1, size V
   dist[src] := 0

   pq := empty priority queue (min-heap) prioritized by dist[] values
   PriorityQueue.push((src, 0))

   while PriorityQueue is not empty:
      u := PriorityQueue.pop().node

      for each neighbor v of u in Graph:
         new_distance := dist[u] + weight(u, v)
         if new_distance < dist[v]:
            dist[v] := new_distance
            parent[v] := u
            PriorityQueue.push((v, dist[v]))

   return dist[], parent[]
```

3. **Drone Operations:**
   - Execute operations (`Takeoff`, `Survey`, `ReturnToHome`, `Land`, `Failure`) using threads.
   - Ensure synchronization such that operations complete in the expected order.

Time Complexity Analysis:

File Reading and Graph Construction (MissionPlanning constructor):

File Reading: Reading each line from the file involves a constant amount of work per line, so if there are `E` edges, this part would take `O(E)` time.

Graph Construction: After reading the file, constructing the adjacency matrix involves iterating over the edges again, which also takes `O(E)` time.

Therefore, the overall time complexity for file reading and graph construction is **O(E)**.

1. **Dijkstra's Algorithm (MissionPlanning::dijkstra)**

   Priority Queue Operations: Each node can be pushed and popped from the priority queue at most once. Since the priority queue operations (`push`, `pop`, `top`) take `O(log V)` time and we might perform up to `E` operations (in the worst case where all edges are processed), the total complexity for priority queue operations is `O(E log V)`.

   Edge Relaxations: Each edge is processed exactly once in the worst case, contributing `O(E)` time.

   Overall: The time complexity of Dijkstra's algorithm is dominated by the priority queue operations in sparse graphs, hence it is **O(E log V)**.

2. **Execution of Drone Operations (executeOperation function):**

   Each operation (`takeoff`, `survey`, `returnHome`, `land`) executes a single `cout` operation, which is constant time, `O(1)`.

3. **Threads and Join Operations:**

   Creating and managing threads (`std::thread`) involves a constant amount of work per thread creation, but the `join` operation waits until the thread completes, which depends on the operations running inside the thread.

   Joining Threads: The `join` operations on threads in the `main` function are sequential and wait for each thread to complete before moving to the next, but they do not add to the time complexity of the core operations significantly.

Overall Time Complexity: The main computational tasks (`file reading & graph construction` and `Dijkstra's algorithm`) are efficiently handled with time complexities of **O(E)** and **O(E log V)** respectively, where `E` is the number of edges and `V` is the number of vertices.