Question 1

Specify the big-O running time of the following operations. For each, please specify the worst case running time. You may assume:

- A vector has \( n \) elements.
- A disjoint set instance has \( n \) elements, and you are implementing union by rank with path compression.
- A graph has \( V \) nodes and \( E \) edges.

**Part A:** Finding \( \text{fib}(n) \) using dynamic programming with memoization.

**Part B:** Determining the connected components in a graph.

**Part C:** Finding an augmenting path through a residual graph using the modified Dijkstra's algorithm.

**Part D:** Performing \( \text{union}() \) in disjoint sets.

**Part E:** Performing \( \text{find}() \) in disjoint sets.

**Part F:** Using Prim's algorithm to find a minimum spanning tree of a graph.

**Part G:** Sorting a vector using merge sort.

**Part H:** Sorting a vector using quicksort.

**Part I:** Finding the shortest path from node \( a \) to node \( b \) in an undirected, unweighted graph.

**Part J:** Doing the "coins" dynamic program with memoization. You have \( c \) denominations of coins, and you want to determine a collection of coins whose values sum to \( n \), which is composed of the minimum number of coins.

Question 2

Behold the following prototype to quicksort:

```c
void quick_sort(vector <double> &v, int start, int size);
```

The procedure will use quicksort to sort the \( \text{size} \) doubles in \( v \), starting with element \( \text{start} \). It assumes that \( v \) has at least \( \text{start}+\text{size} \) elements. It does not default to insertion sort below a certain size.

Below are three calls to \( \text{quick_sort()} \). For each, show me exactly what recursive calls \( \text{quick_sort()} \) makes. You don't have to show additional recursive calls -- just the ones made by that call to \( \text{quick_sort()} \). Use the median-based pivot selection algorithm.

Specify the recursive calls just as I do -- specifying \( v \), \( \text{start} \) and \( \text{size} \).

- Call #1: \( v = \{ 58, 25, 85, 10, 60, 1, 77 \} \), \( \text{start} = 0 \), \( \text{size} = 7 \).
- Call #2: \( v = \{ 46, 71, 12, 41, 18, 23, 93, 65, 19, 62, 55 \} \), \( \text{start} = 2 \), \( \text{size} = 5 \).
- Call #3: \( v = \{ 41, 28, 0, 77, 72, 12, 91, 65, 39, 99, 30, 75, 51, 13 \} \), \( \text{start} = 1 \), \( \text{size} = 11 \).

Question 3

Explain the classes \( P \), \( NP \) and \( NP\)-complete. How do you prove that a problem is \( NP\)-complete?

Question 4

Explain Dijkstra's algorithm. What problem does it solve? Exactly how does it work? What is the running time of each step? Give a small example of how it works on the graph with:

\[
V = \{ A, B, C, D \} \\
E = \{ (A,B,8), (A,C,20), (B,D,30), (C,D,6) \}
\]

Starting node \( A \). Ending node \( D \).