B1 Algorithms (25 points)

Do all three parts.

a) [6 points] Describe the "median of medians" approach to finding the median of n numbers. Analyze the time complexity of the algorithm using the recurrence relation

b) [6 points] Carefully explain Dijkstra's shortest path algorithm and the implementation needed for it to run in time $O(e + n \log n)$.

c) [13 points] Disjoint-set data structure.

i. [3 points] What are the operations for the data structure and the worst-case running time of m operations on n objects if implemented as linked lists? What is the amortized time for an operation in such an implementation?

ii. [3 points] Still implemented as linked lists but with the weighted-union heuristic, what are your answers to the above questions now? Exhibit how you get them.

iii. [4 points] Explain the heuristics of 'union by rank' and 'path compression' when the data structure is implemented as a forest of rooted trees. What is the worst-case running time of m operations on n objects in such an implementation with the heuristic of union-by-rank alone? Give a simple proof to your answer. If combining union by rank and path compression, what is the worst-case running time?

iv. [3 points] Give an example of an algorithm that uses Union-Find on disjoint sets – explain how it is used there.

B2 Algorithms (25 points)

Do all four parts.

a) [8 points] First, explain the main ideas behind Strassen's matrix multiplication algorithm. Second, state the number of scalar multiplications used by Strassen's algorithm when multiplying two 32 by 32 matrices and how you derived that value.

b) [9 points] What is the Fast Fourier Transform (FFT)? Also, explain how to implement the FFT in time O(n log n).

c) [3 points] State the basic principles of dynamic programming.

d) [5 points] Give an $O(n^2)$ algorithm for finding the longest strictly increasing consecutive subsequence of a sequence of n numbers.

Example: For the sequence 7, 8, 10, 2, -3, 3, 6, 7, 55, 41, 76, the desired subsequence has length 5 and is -3, 3, 6, 7, 55

B3 Algorithms (25 points)

Do all three parts.

a) [8 points] Define the following terms:

- P
- NP
- polynomially reducible
- NP-complete

b) [8 points] For each of the following problems, state whether or not it is NP-complete. If it is not NP-complete, then sketch (two or three lines) a polynomial time algorithm for solving the problem. If it is NP-complete, explain why.

i) [2 points] Given an undirected graph G, does G contain a clique of 5 vertices?

ii) [2 points] Given an undirected graph G, does G contain an independent set of 5 vertices? Here, an independent set is a set of vertices without any edges at all between them.

iii) [2 points] Given an undirected graph G and an integer k, does G contain a circuit of length k where no vertex appears more than once?

iv) [2 points] Given an undirected graph G, does G contain a path where every edge of G appears exactly once?

c) [9 points] Give a complete proof that the following problem is NP-complete.

3-EQUAL-PARTS (3EP)

INSTANCE: n real numbers

QUESTION: Does there exist a partition of the n numbers into exactly 3 sets such that the sum of the numbers in each set is the same.

Example: If the instance is 1, 1, 1, 2, 5, 6, 2, 3, then a solution does exist, namely:
{1, 6}, {1, 1, 2, 3}, {2, 5} – clearly here each set sums to 7.

B4 Algorithms (25 points)

Do all three parts.

a) [6 points] Carefully explain the key ideas in the Knuth-Morris-Pratt algorithm. Be sure to state the worst-case running time.

b) [6 points] Carefully describe Prim's minimum spanning tree algorithm. Explain the running time and how it is related to the implementation.

c) [13 points] Fibonacci heaps.

i. [3 points] Describe the data structure of a Fibonacci Heap and how an **Insert** operation is performed.

ii. [5 points] State the worst case and amortized running times for these operations in a Fibonacci Heap: **Insert**, **Min**, **Delete_min**, **Union**, and **Decrease_Key**.

iii. [5 points] Give and analyze a way to add a new type of operations **Increase_Key**(h, i, d), which increases the value of element i by $d > 0$ in a Fibonacci heap h of n elements and runs in amortized time O(log n). Your implementation should *not* change the amortized time for other operations in the data structure.