Final

Note

This is a closed-book exam. Each problem accounts for 10 points, unless otherwise marked.

Problems

1. The partition procedure in the Quicksort algorithm chooses an element as the pivot and divides the input array $A[1..n]$ into two parts such that, when the pivot is properly placed in $A[i]$, the entries in $A[1..(i-1)]$ are less than or equal to $A[i]$ and the entries in $A[(i+1)..n]$ are greater than $A[i]$. The entries that are equal to the pivot may be scattered anywhere in $A[1..(i-1)]$. Please design a refinement of the partition procedure so that it places all entries that are equal to the pivot in consecutive locations. That is, after the refined partition, the entries equal to the pivot are placed in locations from $A[i]$ through $A[j]$ for some i and j, where $1 \leq i \leq j \leq n$, such that the entries in $A[1..(i-1)]$ are less than the pivot and the entries in $A[(j+1)..n]$ are greater than the pivot.

   Please present your refinement in adequate pseudocode and make assumptions wherever necessary. Give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

2. Give a binary de Bruijn sequence of 2^4 bits, which is a cyclic sequence of 2^4 bits $a_1 a_2 \cdots a_{2^4}$ such that each binary sequence of size 4 appears somewhere in the sequence. Explain how you can systematically produce the sequence.

3. Consider Dijkstra's algorithm for single-source shortest paths as shown below.

```
Algorithm Single_Source_Shortest_Paths(G, v);
begin
    for all vertices w do
        w.mark := false;
        w.SP := ∞;
    v.SP := 0;
    while there exists an unmarked vertex do
        let w be an unmarked vertex s.t. w.SP is minimal;
        w.mark := true;
        for all edges (w, z) such that z is unmarked do
            if w.SP + length(w, z) < z.SP then
                z.SP := w.SP + length(w, z)
end
```

The values of SP for all vertices may be stored in either an array or a heap. How do these two implementations compare in terms of time complexity? Please explain.

4. Please explain, using an example, why Dijkstra's algorithm does not work for graphs that contain edges with a negative weight.

5. What is wrong with the following algorithm for computing the minimum-cost spanning tree of a given weighted undirected graph (assumed to be connected)?

   > If the input is just a single-node graph, return the single node. Otherwise, divide the graph into two subgraphs, recursively compute their minimum-cost spanning trees, and then connect the two spanning trees with an edge between the two subgraphs that has the minimum weight.

6. Let $G = (V, E)$ be a connected weighted undirected graph and T be a minimum-cost spanning tree (MCST) of G. Suppose that the cost of one edge $\{u, v\}$ in G is *increased*; $\{u, v\}$ may or may not belong to T. Design an algorithm either to find a new MCST or to determine that T is still an MCST. The more efficient your algorithm is, the more points you will be credited for this problem. Explain why your algorithm is correct and analyze its time complexity.

7. Below is the algorithm discussed in class for determining the strongly connected components of a directed graph. The algorithm is based on depth-first search. During the exploration of the neighbors of a particular node v on which the SCC procedure is invoked, a neighboring node w may be found to have been visited. The neighbor w is reached from v either via a cross edge or a back edge. How are these two cases distinguished and how does the algorithm correctly handle these two cases? Please explain.

```
Algorithm Strongly_Connected_Components(G, n);
begin
   for every vertex v of G do
       v.DFS_Number := 0;
       v.component := 0;
   Current_Component := 0;  DFS_N := n;
   while v.DFS_Number = 0 for some v do
       SCC(v)
end

procedure SCC(v);
begin
   v.DFS_Number := DFS_N;
   DFS_N := DFS_N - 1;
   insert v into Stack;
   v.high := v.DFS_Number;
   for all edges (v, w) do
      if w.DFS_Number = 0 then
```

```
            SCC(w);
            v.high := max(v.high, w.high)
        else if w.DFS_Number > v.DFS_Number
                and w.component = 0 then
            v.high := max(v.high, w.DFS_Number)
    if v.high = v.DFS_Number then
        Current_Component := Current_Component + 1;
        repeat
            remove x from the top of Stack;
            x.component := Current_Component
        until x = v
end
```

8. In the proof (discussed in class) of the NP-hardness of the clique problem by reduction from the SAT problem, we convert an arbitrary boolean expression in CNF (input of the SAT problem) to an input graph of the clique problem.

   (a) Please illustrate the conversion by drawing the graph that will be obtained from the following boolean expression:

   $$(\overline{x} + \overline{z}) \cdot (\overline{w} + x + y + \overline{z}) \cdot (\overline{x} + \overline{y} + z) \cdot (w + y + z).$$

   (b) The original boolean expression is satisfiable. As a demonstration of how the reduction works, please use the resulting graph to argue that it is indeed the case.

9. Let $A \leq_p B$ denote that Problem A is polynomially reducible to Problem B. Prove that, for any pair of NP-complete problems A and B, $A \leq_p B$ and $B \leq_p A$.

10. Solve one of the following two problems. (Note: if you try to solve both problems, I will randomly pick one of them to grade.)

    (a) The traveling salesman problem is as follows.

    > Given a weighted complete graph $G = (V, E)$ (representing a set of cities and the distances between all pairs of cities) and a number D, does there exist a circuit (traveling-salesman tour) that includes all the vertices (cities) and has a total length $\leq D$?

    Prove that the traveling salesman problem is NP-complete.

    (b) The (standard) knapsack problem is as follows.

    > Given a set X, where each element $x \in X$ has an associated size $s(x)$ and value $v(x)$, and two other numbers S and V, is there a subset $B \subseteq X$ whose total size is $\leq S$ and whose total value is $\geq V$?

    Prove that the knapsack problem is NP-complete.

Appendix

- Below is an alternative algorithm for partition in the Quicksort algorithm:

  Partition $(X, \mathit{Left}, \mathit{Right})$;
  begin
  $\mathit{pivot} := X[\mathit{left}]$;
  $i := \mathit{Left}$;
  for $j := \mathit{Left} + 1$ **to** Right **do**
  if $X[j] \leq \mathit{pivot}$ **then** $i := i + 1$;
  $\mathit{swap}(X[i], X[j])$;
  $\mathit{Middle} := i$;
  $\mathit{swap}(X[\mathit{Left}], X[\mathit{Middle}])$
  end

- The Hamiltonian cycle problem: given an undirected graph G, does G have a Hamiltonian cycle? (A Hamiltonian cycle in a graph is a cycle that contains each vertex, except the starting vertex of the cycle, exactly once.)

  The Hamiltonian cycle problem is NP-complete.

- The partition problem: given a set X where each element $x \in X$ has an associated size $s(x)$, is it possible to partition the set into two subsets with exactly the same total size?

  The partition problem is NP-complete.