Final

Note

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

Problems

1. Below is the Mergesort algorithm in pseudocode:

```
Algorithm Mergesort $(X, n)$;
begin $M\_Sort(1, n)$ end

procedure M_Sort $(Left, Right)$;
begin
    if $Right - Left = 1$ then
      if $X[Left] > X[Right]$ then $swap(X[Left], X[Right])$
    else if $Left \neq Right$ then
            $Middle := \lceil \frac{1}{2}(Left + Right) \rceil$;
            $M\_Sort(Left, Middle - 1)$;
            $M\_Sort(Middle, Right)$;
            // the merge part
            $i := Left$;  $j := Middle$;  $k := 0$;
            while $(i \leq Middle - 1)$ and $(j \leq Right)$ do
                  $k := k + 1$;
                  if $X[i] \leq X[j]$ then
                     $TEMP[k] := X[i]$;  $i := i + 1$
                  else $TEMP[k] := X[j]$;  $j := j + 1$;
            if $j > Right$ then
               for $t := 0$ to $Middle - 1 - i$ do
                  $X[Right - t] := X[Middle - 1 - t]$
            for $t := 0$ to $k - 1$ do
               $X[Left + t] := TEMP[t]$
end
```

Given the array below as input, what are the contents of array *TEMP* after the merge part is executed for the first time and what are the contents of *TEMP* when the algorithm terminates? Assume that each entry of *TEMP* has been initialized to 0 when the algorithm starts.

1	2	3	4	5	6	7	8	9	10	11	12
12	5	3	6	11	7	8	2	1	9	10	4

2. Construct a Huffman code tree for a text composed from seven characters A, B, C, D, E, F, and G with frquencies 12, 5, 3, 6, 16, 4, and 8 respectively.

3. Below is an algorithm skeleton for depth-first search utilizing a stack; assume that the input graph is undirected and connected. Modify the algorithm so that it prints out (the edges of) a DFS tree of the input graph. You should try to make as few changes as possible, while maintaining the overall structure of the original algorithm.

```
**Algorithm Simple_Nonrecursive_DFS** $(G, v)$;
**begin**
    push $v$ to *Stack*;
    **while** *Stack* is not empty **do**
        pop vertex $w$ from *Stack*;
        **if** $w$ is unmarked **then**
            mark $w$;
            **for** all edges $(w, x)$ such that $x$ is unmarked **do**
                push $x$ to *Stack*
**end**
```

4. Below is a variant of Prim's algorithm for obtaining a minimum-cost spanning tree of a weighted undirected graph. Please give an analysis of its time complexity. Be explicit about the assumptions you make when doing the analysis.

```
**Algorithm Another_MST**$(G)$;
**begin**
    initially $T$ is the empty set;
    **for** all vertices $w$ **do**
        $w.mark := false$;  $w.cost := \infty$;
    $x.mark := true$; /* $x$ is an arbitrary vertex */
    **for** all edges $(x, z)$ **do**
        $z.edge := (x, z)$;  $z.cost := cost(x, z)$;
    **while** there exists an unmarked vertex **do**
        let $w$ be an unmarked vertex with minimal $w.cost$;
        **if** $w.cost = \infty$ **then**
            print "G is not connected";  halt
        **else**
            $w.mark := true$;
            add $w.edge$ to $T$;
            **for** all edges $(w, z)$ **do**
                **if** not $z.mark$ **then**
                    **if** $cost(w, z) < z.cost$ **then**
                        $z.edge := (w, z)$;
                        $z.cost := cost(w, z)$
**end**
```

5. 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 *decreased*; $\{u, v\}$ 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.

6. Finding a small vertex cover for an arbitrary undirected graph is difficult, but is much easier for trees; a *vertex cover* of a graph G is a set of vertices such that every edge in G is incident to at least one of these vertices. Design an efficient algorithm to find a minimum-size vertex cover for a given tree. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. The more efficient your algorithm is, the more points you will be credited for this problem. Explain why your algorithm is correct and give an analysis of its time complexity.

7. Below is a solution to the single-source shortest path problem using the dynamic programming approach, which we have discussed in class:

   Denote by $D^l(u)$ the length of a shortest path from v (the source) to u containing *at most* l edges; particularly, $D^{n-1}(u)$ is the length of a shortest path from v to u (with no restrictions).

$$D^1(u) = \begin{cases} \mathit{length}(v, u) & \text{if } (v, u) \in E \\ 0 & \text{if } u = v \\ \infty & \text{otherwise} \end{cases}$$

$$\begin{aligned} D^l(u) = \; & \min\{D^{l-1}(u), \min_{(u',u)\in E}\{D^{l-1}(u') + \mathit{length}(u', u)\}\}, \\ & 2 \le l \le n-1 \end{aligned}$$

   Please explain why the solution allows edges with a negative weight (as long as there is no cycle with a negative weight). How is this different from Dijkstra's algorithm? Please explain.

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:

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

   (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. In the proof (discussed in class) of the NP-hardness of the 3SAT problem by reduction from the SAT problem, we convert an arbitrary boolean expression in CNF (input of the SAT problem) to a boolean expression in 3CNF (where each clause has exactly three literals).

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

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

   (b) The original boolean expression is satisfiable. As a demonstration of why the reduction is correct, please use the resulting boolean expression to show that it is indeed the case.

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 subgraph isomorphism problem is as follows.

    Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, does G_1 have a subgraph that is isomorphic to G_2? (Two graphs are isomorphic if there exists a one-one correspondence between the two sets of vertices of the two graphs that preserves adjacency, i.e., if there is an edge between two vertices of the first graph, then there is also an edge between the two corresponding vertices in the second graph, and vice versa.)

    Prove that the subgraph isomorphism problem is NP-complete.

    (b) 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.

Appendix

- 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.