Final

Note

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

Problems

1. Consider the following algorithm that, given a sorted sequence of *distinct* integers $a_1, a_2, \cdots, a_n$ (stored in an array A), determines whether there exists an index i such that $a_i = i$.

```
Algorithm Special_Binary_Search $(A, n)$;
begin
    $Position := Special\_Find(1, n)$;
end

function Special_Find $(Left, Right) : integer$;
begin
    if $Left = Right$ then
        if $A[Left] = Left$ then $Special\_Find := Left$
        else $Special\_Find := 0$
    else
        $Middle := \lfloor \frac{Left+Right}{2} \rfloor$;
        if $A[Middle] < Middle$ then
            $Special\_Find := Special\_Find(Middle + 1, Right)$
        else
            $Special\_Find := Special\_Find(Left, Middle)$
end
```

Draw a decision tree of the algorithm for the case of input size six, i.e., $n = 6$.

2. Consider the following algorithm that computes the square of the input number by using only addition and substraction.

```
Algorithm mySquare $(n)$;
begin
    // assume that $n \geq 0$
    $x := n$;
    $y := 0$;
    while $x > 0$ do
        $y := y + 2x - 1$;
        $x := x - 1$;
    od
    ...
end
```

Let $Inv(n, x, y)$ denote the assertion:

$$x \geq 0 \wedge y \geq 0 \wedge y = (n^2 - x^2).$$

Claim: $Inv(n, x, y)$ is a loop invariant of the while loop, assuming that $n \geq 0$. (The invariant is sufficient to deduce that, when the while loop terminates, $x = 0$ and $y = n^2$.)

Prove the claim.

3. Below is the KMP Algorithm.

```
Algorithm String_Match $(A, n, B, m)$;
begin
    $j := 1$;  $i := 1$;
    $Start := 0$;
    while $Start = 0$ and $i \leq n$ do
        if $B[j] = A[i]$ then
            $j := j + 1$;  $i := i + 1$
        else
            $j := next[j] + 1$;
            if $j = 0$ then
                $j := 1$;  $i := i + 1$;
        if $j = m + 1$ then $Start := i - m$
end
```

```
Algorithm Compute_Next $(B, m)$;
begin
    $next[1] := -1$;  $next[2] := 0$;
    for $i := 3$ to $m$ do
        $j := next[i - 1] + 1$;
        while $B[i - 1] \neq B[j]$ and $j > 0$ do
            $j := next[j] + 1$;
        $next[i] := j$
end
```

Explain why its running time is $O(n)$ (assuming *next* has been pre-computed).

4. 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
```

5. Given as input a connected undirected graph G, a spanning tree T of G, and a vertex v, design an algorithm to determine whether T is a valid DFS tree of G rooted at v. In other words, determine whether T can be the output of DFS under some order of the edges starting with v. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. Explain why the algorithm is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you get for this problem.

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 *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.

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 can these two cases be distinguished and how does the algorithm 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. 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.

9. 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:

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

   (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.

10. To prove that "P = NP" (which seems unlikely though), it suffices to show that some NP-complete problem is in P. Why? Please explain.