

Name (first last): _____

Student ID: _____

- The exam will be closed book, closed notes, except for 1 page of notes.
- No electronic equipment allowed (cell phones, PDA's, computers...).
- Write legibly.
- Use pseudo-code or English to describe your algorithms.
- When designing an algorithm, you are allowed to use any algorithm or data structure we have covered in cs141 or in cs14 without giving its details, unless the question specifically asks for such details.
- If you have a question about the meaning of a question, raise your hand.

for grader use only:

1.	<input type="text"/>	/22
3.	<input type="text"/>	/10
5.	<input type="text"/>	/10

2.	<input type="text"/>	/10
4.	<input type="text"/>	/10
6.	<input type="text"/>	/10

total:	<input type="text"/>	/72
--------	----------------------	-----

1. +2 points for each correct answer, -1 points for each incorrect answer, 0 points for each question not answered.

The running time of the following algorithm is $O(2^n)$: True False

```
1. int recurse(unsigned int n) {
2.   if (n <= 0) return 1;
3.   return recurse(n-1) + 2*recurse(n-1);
4. }
```

Recursion tree has depth n , each internal node has 2 children. $O(2^n)$ nodes, constant work per node.

The running time of the following algorithm is $O(n \log n)$: True False

```
1. void loop(unsigned int n) {
2.   for (int i = 0; i < sqrt(n); ++i)
3.     for (int j = 0; j < i*log(i); ++i)
4.       print "Hi\n";
5. }
```

Running time is $\sum_{i=0}^{\sqrt{n}-1} i \log i = O(\sqrt{n}\sqrt{n} \log \sqrt{n}) = O(n \log n)$.

Note: in fact, this algorithm as typed above will never terminate (since j is never incremented)! If you pointed this out, you will get full credit.

The running time of the following algorithm is $O(n)$: True False

```
1. int fib(unsigned int n) {
2.   if (n <= 1) return n;
3.   return fib(n-1) + fib(n-2);
4. }
```

Every node in the recursion tree at depth $n/2$ or less has two children, so the number of nodes in the recursion tree is at least $2^{n/2}$.

The largest number printed by `mystery(n, 0)` is $\Theta(\log n)$: True False

```
1. void mystery(int n, int d) {
2.   std::cout << d << std::endl;
3.   if (n <= 1) return;
4.   mystery(n-1, d+1);
5. }
```

Mystery(n,0) outputs 0, 1, 2, ..., n.

Suppose $T(0) = T(1) = T(2) = 1$ and $T(n) = T(n-1) + T(n-2) + T(n-3)$ for $n \geq 3$. The recursion tree for T has $O(2^n)$ nodes. True **False**

Each node in the recursion tree has depth at most n and at most two children, so the number of nodes at depth d is at most 2^d . Thus, the number of nodes total is at most $\sum_{d=0}^n 2^d = O(2^n)$.

Suppose $T(0) = T(1) = T(2) = 1$ and $T(n) = T(n-1) + T(n-2) + T(n-3)$ for $n \geq 3$. The recursion tree for T has $\Omega(3^{n/3})$ nodes. **True** False

Each node at depth less than $n/3$ has three children, so the number of nodes is at least $\sum_{d=0}^{n/3} 3^d = \Omega(3^{n/3})$.

Suppose $T(0) = 1$ and $T(n) = 2 * T(n/2) + 1$ for $n > 0$. Then $T(n) = \Theta(n \log n)$. True **False**

Recursion tree is a complete binary tree of depth $\log_2 n$, work done per node is constant, so total work is $O(\#nodes) = O(\sum_{i=0}^{\log_2 n} 2^i) = O(2^{\log_2 n}) = O(n)$.

For every $n = 3, 4, 5, \dots$, there exists a graph with n vertices such that every spanning tree in the graph is contained in a cycle. **True** False

Yes, consider the graph with n nodes that consists of a single cycle of n nodes and n edges.

Recall that an s, t -cut vertex in an undirected graph is a vertex other than s or t whose removal separates s from t . If DFS on a graph yields a back edge from t to s , then that graph cannot have an s, t -cut vertex. **True** False

True. If s and t are connected by a direct edge, the graph has no s, t -cut vertices, because no vertex other than s or t is on that edge.

Kruskal's algorithm runs in linear time. True **False**

No, review Kruskal's algorithm.

In the union-find data structure with union-by-size, the set id associated with an element can change $\Omega(\sqrt{m})$ times, where m is the total number of elements. True **False**

No, each time the element changes its set id, the size of the set it is in at least doubles. If this happens k times, then the element ends up in a set of size at least 2^k . Since no set can be larger than m , we know $2^k \leq m$, so $k = O(\log m)$.

2. A) Draw a graph with vertex set $\{a, b, c, d, e, f\}$ such that a DFS of the graph starting at a can give DFS post-order-numbering $f = 1, e = 2, d = 3, c = 4, b = 5, a = 6$.
- B) Is there only one graph with this vertex set such that a DFS of the graph can give this post-order numbering?
- A) $a-b-c-d-e-f$
- B) No, consider the graph with edge set $\{(a, b), (a, c), (a, d), (a, e), (a, f)\}$.
3. Prove or disprove each of the following claims:
- (A) A directed graph has at least two distinct cycles if and only if any DFS in the graph yields at least two back edges.
- (B) An undirected graph has at least two distinct cycles if and only if any DFS in the graph yields at least two back edges.
- A) This is false, because of cross edges. Consider the graph with vertices $\{a, b, c, d\}$ and edges $\{(a, b), (b, c), (c, a), (a, d), (d, c)\}$.
- B) This is true. If there are two back edges e and e' , then each back edge is on a cycle formed by the back edge and tree edges, and the two cycles are different because one contains e and the other doesn't.
- Conversely, if there are no back edges, then there are no cycles. And if there is only one back edge, there is only one cycle.
4. Prove or disprove that the following algorithm correctly solves the subset-sum problem. The input is a set of integers S and a target integer T .
1. Let s_1, s_2, \dots, s_m be the numbers in S , sorted in decreasing order (so s_1 is the largest number).
 2. Let $C = \emptyset$. (initialize C to the empty set.)
 3. For $i = 1, 2, \dots, m$ do:
 4. Add s_i to C unless doing so would make the total of the numbers in C more than T .
 5. If the total of the numbers in C equals T , then print 'yes, there is a subset of S summing to T .' Otherwise, print 'no, there is no subset of S summing to T .'
- The algorithm is incorrect. It fails on the following example: $S = \{4, 3, 2\}$ and $T = 5$.

5. Given a directed graph with non-negative integer edge weights, a pair of vertices s and t , and integers K and W , describe an algorithm for deciding whether there exists a path from s to t that has total weight W and uses exactly K edges.

Your algorithm should run in time $O((n + m) * W * K)$, where n is the number of vertices and m is the number of edges.

If you have time, illustrate the algorithm on a small example.

State the high-level idea of the algorithm, and explain the running time bound. To be sure your answer is precise and detailed enough, you may want to give pseudo-code (if you have time).

Hint: define $P[v, w, k]$ to be true if there is a path from s to v that has total weight w and uses exactly k edges. (For any vertex v and any integers w and k with $0 \leq w \leq W$ and $0 \leq k \leq K$.)

The following recurrence holds:

$P[v, 0, 0]$ is true for each vertex v .

$P[v, w, 0]$ is false for $w > 0$ and each vertex v .

For $k > 0$, $P[v, w, k]$ is true if and only if $P[u, w - wt(u, v), k - 1]$ is true for some edge (u, v) .

Based on the recurrence, here is an algorithm:

1. Set $P[v, 0, 0] = \text{true}$ for each vertex v .
2. Set $P[v, w, 0] = \text{false}$ for each vertex v and $w = 1, 2, \dots, W$.
3. For $k = 1, 2, \dots, K$ do:
4. For $w = 0, 1, 2, \dots, W$ do:
5. Set $P[v, w, k] = \text{true}$ if there is an edge (u, v) such that $P[u, w - wt(u, v), k - 1]$ is true (for each vertex v).
6. Return $P[t, W, K]$.

The outer loop executes K times, the inner loop executes W times (for each iteration of the outer loop), and implementing line 5 takes linear time.

6. Let $G = (V, E)$ be a graph with edge weights.

Recall that the *bottleneck* of a path p in G is the minimum weight of any edge on the path. Recall that the maximum bottleneck of any s, t -path is the maximum, over all paths p from s to t , of the bottleneck of p .

Prove or disprove: given any connected, undirected, edge-weighted graph, the algorithm below produces a tree T such that the bottleneck of the path p from s to t in T is the maximum bottleneck of any s, t -path in the original graph.

Here is the algorithm:

1. Sort the edges e_1, e_2, \dots, e_m in order of decreasing cost, so e_1 is a maximum-weight edge and e_m is a minimum-weight edge.
2. $T \leftarrow \emptyset$ (initialize T to the empty set)
3. For $i = 1, 2, \dots, m$ do:
4. Add e_i to T if doing so does not create a cycle in T .
5. Return T .

The algorithm is correct.

The algorithm produces a spanning tree T of the graph for the same reason that Kruskal's algorithm does.

At each point in time, each edge considered but not added by the algorithm creates a cycle with the edges added so far.

This implies that: () if you can get from a vertex u to a vertex w using just the edges considered so far, then you get get from u to w using just the edges added so far.*

Now consider the path p connecting s to t in T . Consider the last edge e_i on p added to T by the algorithm.

The property () above implies that there is no path from s to t using just the edges $\{e_1, e_2, \dots, e_{i-1}\}$. In other words, every path from s to t uses some edge in $\{e_i, e_{i+1}, \dots, e_m\}$.*

Keeping in mind that the edges are considered in order of decreasing weight, this implies that every path from s to t uses an edge of weight at most $wt(e_i)$. This implies that every path from s to t has bottleneck at most $wt(e_i)$.