

UCR - CS&E 141 Final Exam 002 solutions

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

The running time of the following algorithm is $\Theta(n^2)$:

True False

```
1. recurse(unsigned int n) {
2.   if (n <= 0) return 1;
3.   for (int i = 0; i < n; ++i)
4.     for (int j = 0; j < n; ++j) std::cout << "Hi\n";
5.   recurse(n/2);
6. }
```

Time is $O(\sum_{i=0}^{\log_2 n} (n/2^i)^2)$. This is a geometric sum, so time is proportional to largest term.

The running time of the following algorithm is $\Theta(n^2 \log n)$:

True False

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

Running time is $\sum_{i=0}^{n \log n} i = \Theta((n \log n)^2)$.

The running time of the following algorithm is $\Theta(n^2)$:

True False

```
1. HashTable<int,int> cache;
2.
3. int mystery1(unsigned int n) {
4.   if (! cache.exists(n) {
5.     if (n <= 1) cache[n] = 1;
6.     else cache[n] = mystery1(n-1)+MYSTERY2(n);
7.   }
8.
9.   int MYSTERY2(unsigned int n) {
10.    if (n <= 1) return 1;
11.    return MYSTERY2(n-1)+n/2;
12. }
```

First, the time for MYSTERY2(n) is $O(n)$. This follows from looking at the recursion tree for MYSTERY2(n), which has depth n and one child per node, with constant work per node.

Next, $\text{mystery1}(n)$ results in the recursive calls $\text{mystery1}(n-1), \text{mystery1}(n-2), \text{mystery1}(n-3)$, etc. (The caching makes no difference.) Thus, the recursion tree for mystery1 has depth n , and one child per node, with work $\Theta(i)$ at level i (due to the call to $\text{MYSTERY2}(i)$). Thus, the total work is $\Theta(\sum_{i=0}^n i) = \Theta(n^2)$.

The largest number printed by $\text{mystery}(n, 0)$ is $O(n)$:

True False

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

Yes, the depth of the recursion tree is $O(\log n)$. A recursive call at depth d prints d . So the largest number printed is $O(\log n)$. Since big- O is an upper bound, the largest number printed is also $O(n)$.

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

True False

This question was not stated as I intended, so I will accept either 'true' or 'false' as a correct answer. (Though in fact the answer is 'false'.)

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

True False

The total work is $1 + 3 + 3^2 + 3^3 + \dots + 3^{\log_3 n} = \Theta(n)$.

Suppose $T(0) = 1$ and $T(n) = 3 * T(n/3) + 1$ for $n > 0$.

Then $T(n) = \Theta(n \log n)$.

True False

The recursion tree has depth n , and the number of nodes at depth d is 2^d . Thus, the number of nodes total is $\Theta(2^n)$. Constant work is done at each node, so the total work is $\Theta(2^n)$.

For every $n = 3, 4, 5, \dots$, there exists a connected graph with n vertices such that every path in the graph has at least 2 edges.

True False

No, any connected graph has some edge (u, v) which is, by itself, a path.

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 .

True or false: if a graph G has a cut vertex, then for some pair of vertices s and t , the graph G also has an s, t -cut vertex.

True False

True. Let v be a cut vertex. Then removing v from G separates some vertex s from some other vertex t .

Dijkstra's algorithm runs in $O(m \log n)$ time (in a graph with n vertices and m edges).

True False

I misstated this question, so I will accept either 'true' or 'false' as an answer. (I assumed without stating it that the graph was connected. In fact, Dijkstra's algorithm can be implemented to run in $O(m \log n)$ time even in a graph with fewer edges than nodes, but we did not discuss that in class.

In the growable array data structure from programming assignment 1, a single access to array element $A[M]$ can cause the data structure to take $\Omega(M)$ time.

True False

True. For example, in a newly allocated array, accessing $A[M]$ will cause the array to grow to size $\Theta(M)$, which takes $\Theta(M)$ time.

2. A) Draw an undirected graph with vertex set $\{a, b, c, d, e, f\}$ and 6 edges with the following property: the algorithm for identifying cut vertices will assign the same low number to every vertex.

B) Is there only one such graph? If no, about how many do you think there are?

A) Draw a 6-cycle containing all 6 vertices.

B) No, any 6-cycle will do. There are $(5 \cdot 4 \cdot 3 \cdot 2) / 2$ different 6 cycles using these vertices.

You may have interpreted any two 6-cycles using these 6 vertices as being the same graph. In this case the answer is yes, there is only one, and you will get credit if you stated your assumption clearly.

3. Prove or disprove the following claim: An undirected graph has at least three distinct cycles if and only if any DFS in the graph yields at least three back edges.

The claim is false. Consider the graph with edges $a-b-c-d-e$ and $a-e$ and $b-d$. Any DFS yields 4 tree edges, and so 2 back edges. Yet the graph has at least three distinct cycles.

4. Consider the following algorithm. The input is an undirected graph G with edge weights and a pair of vertices s and t .

1. Find a minimum spanning tree T in G .

2. Find the path p from s to t (in T).

3. Output the path p .

Prove or disprove: the above algorithm always finds a shortest path from s to t in any edge-weighted graph.

This is false. Consider a graph with edges $a-b-c$ and $c-a$. The algorithm can find $a-b-c$ as an MST, yet the shortest path from 'a' to 'c' has length 1.

5. Given a directed acyclic graph where every edge is colored either red or blue, and a pair of vertices s and t , describe an algorithm for deciding whether there exists a path from s to t that has the same number of red edges as blue edges.

Your algorithm should run in $O(n^2(n + m))$ time.

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).

For each vertex v and $k = -n, -n - 1, -n - 2, \dots, n$, define $X[v, k]$ to be the number of paths from s to v with k more red edges than blue edges. Then $X[v, k]$ satisfies the following recurrence:

$$X[s, 0] = 1,$$

$$X[s, k] = 0 \text{ for } k \neq 0,$$

$$X[v, k] = \sum_{w:(w,v) \in E, (w,v) \text{ red}} X[w, k - 1] + \sum_{w:(w,v) \in E, (w,v) \text{ blue}} X[w, k + 1].$$

The algorithm is

1. Initialize $X[s, 0] = 1$ and $X[s, k] = 0$ for $k \neq 0$.
2. If t is not reachable from s , return 'false'.
3. For each vertex v_1, v_2, \dots, v_n reachable from s , in topological order, do:
4. For each $k = -n, -n - 1, -n - 2, \dots, n$ do:
5. Set

$$X[v, k] = \sum_{w:(w,v) \in E; (w,v) \text{ is red}} X[w, k - 1] + \sum_{w:(w,v) \in E; (w,v) \text{ is blue}} X[w, k + 1].$$

6. Return 'true' if $X[t, 0] > 0$, else 'false'.

6. Given n , describe an algorithm for counting the number of distinct triangulations of n $\{1, 2, \dots, n\}$ lying on a circle.

Your algorithm should run in $O(n), O(n^2)$, or $O(n^3)$ time.

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).

The high-level idea is similar to the algorithm for finding a min-weight triangulation, except it counts the number of triangulations instead of finding the min-weight one.

1. $C[1] = C[2] = C[3] = 1$.
2. For $i = 4, 5, \dots, n$ do:
3. $C[i] = \sum_{k=2}^{i-1} C[k]C[i - k + 1]$.
4. return $C[n]$.

The algorithm runs in $\Theta(n^2)$ time.