

1. If the worst-case running time of an algorithm is  $O(n^2)$ , then the algorithm can finish in time  $O(n)$  on some inputs of size  $n$ . \_\_\_\_\_  True  False

If the worst-case running time of an algorithm is  $O(n)$ , then the algorithm can finish in time  $\Omega(n^2)$  on some inputs of size  $n$ . \_\_\_\_\_  True  False

$1 + 2 + 3 + \dots + (n - 1) + n = O(n)$  \_\_\_\_\_  True  False

$1^{10} + 2^{10} + 3^{10} + \dots + (n - 1)^{10} + n^{10} = \Theta(n^{11})$  \_\_\_\_\_  True  False

If  $\alpha = 1.001$ , then  $\alpha^0 + \alpha^1 + \alpha^2 + \dots + \alpha^n = \Omega(2^n)$  \_\_\_\_\_  True  False

Suppose  $T(0) = 0$  and  $T(n) = n + T(n/2)$  for  $n > 0$ .  
Then  $T(n) = O(n)$ . \_\_\_\_\_  True  False

Suppose  $T(0) = 1$  and  $T(n) = n + 2T(n/2)$  for  $n > 0$ .  
The recursion tree for  $T$  has depth  $O(\log n)$ . \_\_\_\_\_  True  False

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

Assuming the growable Array class is implemented as in programming assignment 1, consider the following code:

```

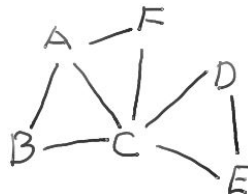
1. void test(int N) {
2.     Array<int> A;
3.     for (i = 0; i < N; ++i)
4.         A[i] = i;
5. }

```

Each time line 4 is executed, it takes  $O(1)$  time. \_\_\_\_\_  True  False

On any input  $N$ , test( $N$ ) takes time  $O(N)$ . \_\_\_\_\_  True  False

2.



Run depth-first search on the above graph. Start at A, and whenever you have a choice about which edge to explore next, choose the one that goes to the vertex that is earliest in the alphabet.

Below, draw the DFS tree, drawing the tree edges with solid lines and the non-tree edges using dashed lines.



How many distinct paths are there that go from A to F and through C? Give a value and explain how you got it.

*There are 4. There are two paths from A to C, and two paths from C to F. Each path from A to F through C is exactly formed by a path from A to C, then a path from C to F. Thus, the number of such paths is the product  $2*2 = 4$ .*

---

4. Let  $T(n)$  be the *time* taken by the following subroutine:

```
int mystery3(int n) {
    if (n <= 1) return 2;
    else return (2*mystery3(n-1) + 3*mystery3(n-2)) % 10;
}
```

Give a recurrence relation for  $T(n)$  (ignoring constant factors):

*The recurrence is*

- $T(n) = 1$  (for  $n \leq 1$ )
- $T(n) = T(n - 1) + T(n - 2)$  (for  $n > 1$ )

What is the depth of the deepest node in the recursion tree (in terms of  $n$ )?

*The deepest node has depth  $n - 1 = O(n)$ .*

Precisely describe an algorithm that runs in  $O(n)$  time and returns the same value as `mystery3(n)`.

*The algorithm is*

```
int mystery3(int n) {
    if (n <= 1) return 2;
    Array<int> A;
    A[0] = 2;
    A[1] = 2;
    for (int i = 2; i <= n; ++i)
        A[i] = (2*A[n-1] + 3*A[n-2]) % 10;
    return A[n];
}
```

.

---

5. Prove or disprove the following claim:

For any digraph  $G$  and any edge  $(u, w)$  on a cycle in  $G$ , it is *always* possible to run DFS on  $G$  in such a way that  $(u, w)$  is classified as a back edge.

(By “it is possible to run DFS on  $G$  in such a way that  $(u, w)$  is classified as a back edge” we mean that, if DFS chooses the start vertex and the neighbors to visit from each vertex in the right way, then  $(u, w)$  will end up as a back edge.)

*The claim is true. If the DFS starts at  $w$ ,  $u$  will be a descendant of  $w$  in the DFS tree, so the edge  $(w, u)$  will have to be a back edge.*

---

6. Describe a linear-time algorithm for the following problem:

Given a digraph  $G$  with no cycles, and two vertices  $s$  and  $t$ , count the number of even-length paths from  $s$  to  $t$ .

(Hint: dynamic programming.)

(An “even-length” path is one with an even number of edges.)

For each vertex  $v$ , define

$$EVEN[v] = \# \text{ even-length paths from } s \text{ to } v$$

$$ODD[v] = \# \text{ odd-length paths from } s \text{ to } v$$

Then these satisfy the following recurrence:

$$EVEN[s] = 1, \quad ODD[s] = 0$$

$$EVEN[v] = \sum_{w:(u,w) \in E} ODD[v], \quad ODD[v] = \sum_{w:(u,w) \in E} EVEN[v].$$

Here is the algorithm:

1. Set  $EVEN[s] = 1$ ,  $ODD[s] = 0$ .
2. Initialize  $EVEN[v] = ODD[v] = 0$  for all other vertices  $v$ .
3. Consider the vertices in topological order.
4. For each vertex  $v$ , reset  $EVEN[v]$  and  $ODD[v]$  according to the above recurrence.
5. Return  $EVEN[s]$ .