

# 201A-MATH182-1 Midterm

ESHAN UNIYAL

TOTAL POINTS

**54 / 54**

## QUESTION 1

12 pts

### 1.1 5 / 5

- ✓ - **0 pts** Correct
  - **1 pts** Loop invariant not strong enough to prove correctness
  - **2 pts** Incorrect loop invariant or no loop invariant given
  - **1 pts** Error

### 1.2 5 / 5

- ✓ - **0 pts** Correct
  - **1 pts** Missing upper bound

### 1.3 2 / 2

- ✓ - **0 pts** Correct
  - **2 pts** No answer submitted
  - **0.5 pts** Click here to replace this description.
  - **1 pts** No justification given
  - **1 pts** Incorrect

## QUESTION 2

15 pts

### 2.1 5 / 5

- ✓ - **0 pts** Correct

### 2.2 5 / 5

- ✓ - **0 pts** Correct
  - **2 pts** Incorrect

### 2.3 5 / 5

- ✓ - **0 pts** Correct
  - **1 pts** Error
  - **2 pts** Incorrect

## QUESTION 3

3 5 / 5

- **0 pts** Correct
- ✓ - **0 pts** Assume a particular value for  $T(0)$  or  $T(1)$ 
  - **0.5 pts** Upper bound only
  - **1 pts** Derive consequence different than inductive hypothesis
  - **0.5 pts** Neglect constant term
  - **1.5 pts** Incorrect induction
  - **1 pts** Only prove lower bound by substitution
  - **1 pts** Recursion tree drawn but seemingly not used
  - **4 pts** Incorrect recursion tree and bound, no proof
  - **1 pts** Repeatedly substitute in recurrence instead of using substitution method
  - **2 pts** Repeated substitution; don't use recursion tree
- ①  $T(0) \neq 1$

## QUESTION 4

4 5 / 5

- ✓ - **0 pts** Correct
  - **2 pts** Incorrect
  - **1 pts** Bounds not tight

## QUESTION 5

5 5 / 5

- ✓ - **0 pts** Correct
  - **1 pts** Correct answer, gap in proof
  - **2 pts** Correct answer, error in proof
  - **3 pts** Correct answer, proof just reformulates the desired result and asserts truth
  - **3 pts** Correct answer, incorrect proof
  - **3 pts** Correct answer, no proof
  - **3 pts** Partially correct answer, reformulated and

asserted

- **1 pts** Incorrect answer from careless error
- **2 pts** Incorrect answer, hole in proof
- **4 pts** Incorrect answer, reformulated and asserted
- **4 pts** Incorrect answer, incorrect proof.
- **5 pts** Incorrect answer, no justification

#### QUESTION 6

12 pts

##### 6.1 2 / 2

- ✓ - **0 pts** Correct
- **2 pts** Incorrect
- **1 pts** No answer

##### 6.2 2 / 2

- ✓ - **0 pts** Correct
- **2 pts** Incorrect
- **1 pts** No answer

##### 6.3 2 / 2

- ✓ - **0 pts** Correct
- **2 pts** Incorrect
- **1 pts** No answer

##### 6.4 2 / 2

- ✓ - **0 pts** Correct
- **2 pts** Incorrect
- **1 pts** No answer

##### 6.5 2 / 2

- ✓ - **0 pts** Correct
- **2 pts** Incorrect
- **1 pts** No answer

##### 6.6 2 / 2

- ✓ - **0 pts** Correct
- **2 pts** Incorrect
- **1 pts** No answer

**MATH182 MIDTERM**  
**DUE July 10, 2020**

**Question 1.** Consider the following pseudo-code:

```
1  sum = A[1]
2  max = sum
3  for j = 2 to A.length
4      sum = sum + A[j]
5      if sum > max
6          max = sum
7  return max
```

This algorithm takes as input an array  $A[1..n]$  and outputs the value of the maximum subarray of the form  $A[1..j]$ , i.e., it outputs the number

$$\max \left\{ \sum_{i=1}^j A[i] : 1 \leq j \leq A.length \right\}$$

- (1) Give a proof of the correctness of this algorithm. Your proof should include: a precise statement of a loop invariant for the **for** loop, and a proof of this loop invariant. (5pts)
- (2) Analyze the running-time of this algorithm. This includes deducing a tight asymptotic bound. (5pts)
- (3) Is this algorithm asymptotically optimal (i.e., is there another algorithm with asymptotically smaller running time which can do the same thing this algorithm does)? Justify your answer. (2pts)

*Solution.*

(1) We first define  $T_J$  as

$$T_J := \left\{ \sum_{i=1}^k A[i] : 1 \leq k \leq J \right\}$$

The loop invariant for the **for** loop is then as follows:

**Loop Invariant:** After line 3 is run, **sum** is the sum of all elements in  $A[1..j-1]$ , and **max** is the sum of the maximal subarray of the form  $A[1..k]$  in  $A[j..j-1]$ , i.e.,

$$\text{sum} = \sum_{i=1}^{j-1} A[i] ; \text{max} = \max \left\{ \sum_{i=1}^k A[i] : 1 \leq k \leq j-1 \right\} \equiv \max T_{j-1}$$

We show this loop invariant holds.

*Initialisation:* Before the first iteration, we have **sum** =  $A[1]$ , **max** = **sum** =  $A[1]$ , and  $j = 2$ . Since the subarray  $A[j..j-1] = A[1..1]$  is a one-element array, **max** = **sum** =  $A[1]$  is (trivially) the sum of its only (and therefore maximal) subarray,  $A[1..1]$ . The loop invariant is therefore true in the initialisation step.

*Maintenance:* Assume the loop invariant is true before some iteration with  $j = j_0 \in [2 \dots A.length]$ . By assumption and definition of the loop invariant, we have

$$\text{sum} = \sum_{i=1}^{j_0-1} A[i] ; \text{max} = \max T_{j_0-1}$$

In line 4, we add  $A[j_0]$  sum, and therefore have  $\text{sum} = \sum_{i=1}^{j_0} A[i]$ . The first requirement of the loop invariant for  $j = j_0 + 1$  is therefore satisfied. Comparing  $\text{sum}$  to  $\text{max}$  in line 5, we have two cases:

*Case 1:* ( $\text{sum} > \text{max}$ ) By definition of  $\text{sum}$  and  $\text{max}$ , we have

$$\sum_{i=1}^{j_0} A[i] > \max T_{j_0-1}$$

Updating  $\text{max}$  to  $\text{sum}$  in line 6 therefore gives us

$$\text{max} = \sum_{i=1}^{j_0} A[i] = \max \left\{ T_{j_0-1} \cup \left\{ \sum_{i=1}^{j_0} A[i] \right\} \right\} = \max T_{j_0}$$

We see that this state of  $\text{max}$  is precisely the second requirement of the loop invariant for  $j = j_0 + 1$ .

*Case 2:* ( $\text{sum} \leq \text{max}$ ) In this case, condition of the **if** statement in line 5 fails and  $\text{max}$  is left unmodified. By definition of  $\text{sum}$  and  $\text{max}$ , we therefore have

$$\sum_{i=1}^{j_0} A[i] \leq \max T_{j_0-1} \implies \text{max} = \max \left\{ T_{j_0-1} \cup \left\{ \sum_{i=1}^{j_0} A[i] \right\} \right\} = \max T_{j_0}$$

We see that this state of  $\text{max}$  is also precisely the second requirement of the loop invariant for  $j = j_0 + 1$ .

The loop invariant is therefore true for  $j = j_0 + 1$ .

*Termination:* In the final iteration, we have  $j = A.length + 1$  and the loop terminates. From the loop invariant, we know

$$\text{max} = \max T_{A.length+1-1} = \max T_{A.length} = \max \left\{ \sum_{i=1}^k A[i] : 1 \leq k \leq A.length \right\}$$

which is precisely the sum of the maximal subarray of  $A$  of the form  $A[1 \dots j]$ , and is then returned in line 7. The algorithm is therefore correct. ■

(2) We consider the cost and number of times each line is run for  $n := A.length$ :

1	$sum = A[1]$	cost : $c_1$ times : 1
2	$max = sum$	cost : $c_2$ times : 1
3	<b>for</b> $j = 2$ <b>to</b> $n$	cost : $c_3$ times : $n$
4	$sum = sum + A[j]$	cost : $c_4$ times : $n$
5	<b>if</b> $sum > max$	cost : $c_5$ times : $n$
6	$max = sum$	cost : $c_6$ times : $k \in [0, n]$
7	<b>return</b> $max$	cost : $c_7$ times : 1

Since  $0 \leq k \leq n$ , we may define  $c_{6'} := k/n \in [0, 1]$ . We therefore have  $k = c_{6'}n$  for some  $c_{6'} \in [0, 1]$ . Let  $T(n)$  be the running time of the algorithm for an array of length  $n$ . Adding up the total cost of each line, we have

$$\begin{aligned} T(n) &= c_1 \cdot 1 + c_2 \cdot 1 + c_3 \cdot n + c_4 \cdot n + c_5 \cdot n + c_6 c_{6'} \cdot n + c_7 \cdot 1 \\ &= (c_3 + c_4 + c_5 + c_6 c_{6'}) \cdot n + (c_1 + c_2 + c_7) = \Theta(n) \end{aligned}$$

1.1 5 / 5

✓ - 0 pts Correct

- 1 pts Loop invariant not strong enough to prove correctness

- 2 pts Incorrect loop invariant or no loop invariant given

- 1 pts Error

*Maintenance:* Assume the loop invariant is true before some iteration with  $j = j_0 \in [2 \dots A.length]$ . By assumption and definition of the loop invariant, we have

$$\text{sum} = \sum_{i=1}^{j_0-1} A[i] ; \text{max} = \max T_{j_0-1}$$

In line 4, we add  $A[j_0]$  sum, and therefore have  $\text{sum} = \sum_{i=1}^{j_0} A[i]$ . The first requirement of the loop invariant for  $j = j_0 + 1$  is therefore satisfied. Comparing  $\text{sum}$  to  $\text{max}$  in line 5, we have two cases:

*Case 1:* ( $\text{sum} > \text{max}$ ) By definition of  $\text{sum}$  and  $\text{max}$ , we have

$$\sum_{i=1}^{j_0} A[i] > \max T_{j_0-1}$$

Updating  $\text{max}$  to  $\text{sum}$  in line 6 therefore gives us

$$\text{max} = \sum_{i=1}^{j_0} A[i] = \max \left\{ T_{j_0-1} \cup \left\{ \sum_{i=1}^{j_0} A[i] \right\} \right\} = \max T_{j_0}$$

We see that this state of  $\text{max}$  is precisely the second requirement of the loop invariant for  $j = j_0 + 1$ .

*Case 2:* ( $\text{sum} \leq \text{max}$ ) In this case, condition of the **if** statement in line 5 fails and  $\text{max}$  is left unmodified. By definition of  $\text{sum}$  and  $\text{max}$ , we therefore have

$$\sum_{i=1}^{j_0} A[i] \leq \max T_{j_0-1} \implies \text{max} = \max \left\{ T_{j_0-1} \cup \left\{ \sum_{i=1}^{j_0} A[i] \right\} \right\} = \max T_{j_0}$$

We see that this state of  $\text{max}$  is also precisely the second requirement of the loop invariant for  $j = j_0 + 1$ .

The loop invariant is therefore true for  $j = j_0 + 1$ .

*Termination:* In the final iteration, we have  $j = A.length + 1$  and the loop terminates. From the loop invariant, we know

$$\text{max} = \max T_{A.length+1-1} = \max T_{A.length} = \max \left\{ \sum_{i=1}^k A[i] : 1 \leq k \leq A.length \right\}$$

which is precisely the sum of the maximal subarray of  $A$  of the form  $A[1 \dots j]$ , and is then returned in line 7. The algorithm is therefore correct. ■

(2) We consider the cost and number of times each line is run for  $n := A.length$ :

1	$sum = A[1]$	cost : $c_1$ times : 1
2	$max = sum$	cost : $c_2$ times : 1
3	<b>for</b> $j = 2$ <b>to</b> $n$	cost : $c_3$ times : $n$
4	$sum = sum + A[j]$	cost : $c_4$ times : $n$
5	<b>if</b> $sum > max$	cost : $c_5$ times : $n$
6	$max = sum$	cost : $c_6$ times : $k \in [0, n]$
7	<b>return</b> $max$	cost : $c_7$ times : 1

Since  $0 \leq k \leq n$ , we may define  $c_{6'} := k/n \in [0, 1]$ . We therefore have  $k = c_{6'}n$  for some  $c_{6'} \in [0, 1]$ . Let  $T(n)$  be the running time of the algorithm for an array of length  $n$ . Adding up the total cost of each line, we have

$$\begin{aligned} T(n) &= c_1 \cdot 1 + c_2 \cdot 1 + c_3 \cdot n + c_4 \cdot n + c_5 \cdot n + c_6 c_{6'} \cdot n + c_7 \cdot 1 \\ &= (c_3 + c_4 + c_5 + c_6 c_{6'}) \cdot n + (c_1 + c_2 + c_7) = \Theta(n) \end{aligned}$$

The running time of this algorithm is therefore  $\Theta(n)$ .

(3) Yes, this algorithm is asymptotically optimal, since any algorithm searching for the maximal subarray of the form  $A[1..j]$  must examine each element of the array (of length  $n$ ) at least once and will therefore have running time  $\Omega(n)$ .

**Question 2.** Recall that for  $0 \leq k \leq n$ , the **binomial coefficient**  $\binom{n}{k}$  is defined by

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}$$

In particular, we have  $\binom{n}{0} = \binom{n}{n} = 1$  for every  $n$ .

(1) Prove for every  $0 < k < n$ :

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

You can use any valid method you know to prove this (from the definition, combinatorial, generating function, etc.) (5pts)

(2) Write pseudocode for a recursive algorithm BINOMIAL( $n, k$ ) which returns  $\binom{n}{k}$ . Your algorithm should use the above fact you proved in (1). (5pts)

(3) Give a proof of correctness of your algorithm in (2). You should prove the statement: “For every  $n \geq 0$  and for every  $0 \leq k \leq n$ , BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$ .” (5pts)

*Solution.*

(1) Let  $k$  and  $n$  be such that  $0 < k < n$ . Therefore, since  $k \leq n-1$  and  $k-1 \leq n-1$ , the binomial coefficients  $\binom{n-1}{k}$  and  $\binom{n-1}{k-1}$  are well-defined. We therefore have

$$\begin{aligned} \binom{n-1}{k} + \binom{n-1}{k-1} &= \frac{(n-1)!}{k!(n-k-1)!} + \frac{(n-1)!}{(k-1)!(n-k)!} \\ &= \frac{(n-1)!}{k!(n-k-1)!} \cdot \frac{n-k}{n-k} + \frac{(n-1)!}{(k-1)!(n-k)!} \cdot \frac{k}{k} \\ &= \frac{(n-k) \cdot (n-1)!}{k!(n-k)!} + \frac{k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n-k) \cdot (n-1)! + k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n \cdot (n-1)!)}{k!(n-k)!} = \frac{n!}{k!(n-k)!} = \binom{n}{k} \\ \therefore \binom{n}{k} &= \binom{n-1}{k} + \binom{n-1}{k-1} \quad \blacksquare \end{aligned}$$

(2) Following is pseudocode for a recursive BINOMIAL( $n, k$ ) algorithm based on the fact from (1):

BINOMIAL( $n, k$ ):

```

1 // base cases:  $k = 0$  or  $n = k$  imply  $\binom{n}{k} = 1$ 
2 if  $k == 0$  or  $n == k$ 
3   return 1
4 // recursive step
5 else return BINOMIAL( $n-1, k$ ) + BINOMIAL( $n-1, k-1$ )
```

(3) For  $n \geq 0$ , we define the property

1.2 5 / 5

✓ - 0 pts Correct

- 1 pts Missing upper bound



The running time of this algorithm is therefore  $\Theta(n)$ .

(3) Yes, this algorithm is asymptotically optimal, since any algorithm searching for the maximal subarray of the form  $A[1..j]$  must examine each element of the array (of length  $n$ ) at least once and will therefore have running time  $\Omega(n)$ .

**Question 2.** Recall that for  $0 \leq k \leq n$ , the **binomial coefficient**  $\binom{n}{k}$  is defined by

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}$$

In particular, we have  $\binom{n}{0} = \binom{n}{n} = 1$  for every  $n$ .

(1) Prove for every  $0 < k < n$ :

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

You can use any valid method you know to prove this (from the definition, combinatorial, generating function, etc.) (5pts)

(2) Write pseudocode for a recursive algorithm BINOMIAL( $n, k$ ) which returns  $\binom{n}{k}$ . Your algorithm should use the above fact you proved in (1). (5pts)

(3) Give a proof of correctness of your algorithm in (2). You should prove the statement: “For every  $n \geq 0$  and for every  $0 \leq k \leq n$ , BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$ .” (5pts)

*Solution.*

(1) Let  $k$  and  $n$  be such that  $0 < k < n$ . Therefore, since  $k \leq n-1$  and  $k-1 \leq n-1$ , the binomial coefficients  $\binom{n-1}{k}$  and  $\binom{n-1}{k-1}$  are well-defined. We therefore have

$$\begin{aligned} \binom{n-1}{k} + \binom{n-1}{k-1} &= \frac{(n-1)!}{k!(n-k-1)!} + \frac{(n-1)!}{(k-1)!(n-k)!} \\ &= \frac{(n-1)!}{k!(n-k-1)!} \cdot \frac{n-k}{n-k} + \frac{(n-1)!}{(k-1)!(n-k)!} \cdot \frac{k}{k} \\ &= \frac{(n-k) \cdot (n-1)!}{k!(n-k)!} + \frac{k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n-k) \cdot (n-1)! + k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n \cdot (n-1)!)}{k!(n-k)!} = \frac{n!}{k!(n-k)!} = \binom{n}{k} \\ \therefore \binom{n}{k} &= \binom{n-1}{k} + \binom{n-1}{k-1} \quad \blacksquare \end{aligned}$$

(2) Following is pseudocode for a recursive BINOMIAL( $n, k$ ) algorithm based on the fact from (1):

BINOMIAL( $n, k$ ):

```

1 // base cases:  $k = 0$  or  $n = k$  imply  $\binom{n}{k} = 1$ 
2 if  $k == 0$  or  $n == k$ 
3   return 1
4 // recursive step
5 else return BINOMIAL( $n-1, k$ ) + BINOMIAL( $n-1, k-1$ )
```

(3) For  $n \geq 0$ , we define the property

1.3 2 / 2

✓ - 0 pts Correct

- 2 pts No answer submitted

- 0.5 pts [Click here to replace this description.](#)

- 1 pts No justification given

- 1 pts Incorrect

The running time of this algorithm is therefore  $\Theta(n)$ .

(3) Yes, this algorithm is asymptotically optimal, since any algorithm searching for the maximal subarray of the form  $A[1..j]$  must examine each element of the array (of length  $n$ ) at least once and will therefore have running time  $\Omega(n)$ .

**Question 2.** Recall that for  $0 \leq k \leq n$ , the **binomial coefficient**  $\binom{n}{k}$  is defined by

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}$$

In particular, we have  $\binom{n}{0} = \binom{n}{n} = 1$  for every  $n$ .

(1) Prove for every  $0 < k < n$ :

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

You can use any valid method you know to prove this (from the definition, combinatorial, generating function, etc.) (5pts)

(2) Write pseudocode for a recursive algorithm BINOMIAL( $n, k$ ) which returns  $\binom{n}{k}$ . Your algorithm should use the above fact you proved in (1). (5pts)

(3) Give a proof of correctness of your algorithm in (2). You should prove the statement: “For every  $n \geq 0$  and for every  $0 \leq k \leq n$ , BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$ .” (5pts)

*Solution.*

(1) Let  $k$  and  $n$  be such that  $0 < k < n$ . Therefore, since  $k \leq n-1$  and  $k-1 \leq n-1$ , the binomial coefficients  $\binom{n-1}{k}$  and  $\binom{n-1}{k-1}$  are well-defined. We therefore have

$$\begin{aligned} \binom{n-1}{k} + \binom{n-1}{k-1} &= \frac{(n-1)!}{k!(n-k-1)!} + \frac{(n-1)!}{(k-1)!(n-k)!} \\ &= \frac{(n-1)!}{k!(n-k-1)!} \cdot \frac{n-k}{n-k} + \frac{(n-1)!}{(k-1)!(n-k)!} \cdot \frac{k}{k} \\ &= \frac{(n-k) \cdot (n-1)!}{k!(n-k)!} + \frac{k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n-k) \cdot (n-1)! + k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n \cdot (n-1)!)}{k!(n-k)!} = \frac{n!}{k!(n-k)!} = \binom{n}{k} \\ \therefore \binom{n}{k} &= \binom{n-1}{k} + \binom{n-1}{k-1} \quad \blacksquare \end{aligned}$$

(2) Following is pseudocode for a recursive BINOMIAL( $n, k$ ) algorithm based on the fact from (1):

BINOMIAL( $n, k$ ):

```

1 // base cases:  $k = 0$  or  $n = k$  imply  $\binom{n}{k} = 1$ 
2 if  $k == 0$  or  $n == k$ 
3   return 1
4 // recursive step
5 else return BINOMIAL( $n-1, k$ ) + BINOMIAL( $n-1, k-1$ )
```

(3) For  $n \geq 0$ , we define the property

2.1 5 / 5

✓ - 0 pts Correct

The running time of this algorithm is therefore  $\Theta(n)$ .

(3) Yes, this algorithm is asymptotically optimal, since any algorithm searching for the maximal subarray of the form  $A[1..j]$  must examine each element of the array (of length  $n$ ) at least once and will therefore have running time  $\Omega(n)$ .

**Question 2.** Recall that for  $0 \leq k \leq n$ , the **binomial coefficient**  $\binom{n}{k}$  is defined by

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}$$

In particular, we have  $\binom{n}{0} = \binom{n}{n} = 1$  for every  $n$ .

(1) Prove for every  $0 < k < n$ :

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

You can use any valid method you know to prove this (from the definition, combinatorial, generating function, etc.) (5pts)

(2) Write pseudocode for a recursive algorithm BINOMIAL( $n, k$ ) which returns  $\binom{n}{k}$ . Your algorithm should use the above fact you proved in (1). (5pts)

(3) Give a proof of correctness of your algorithm in (2). You should prove the statement: “For every  $n \geq 0$  and for every  $0 \leq k \leq n$ , BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$ .” (5pts)

*Solution.*

(1) Let  $k$  and  $n$  be such that  $0 < k < n$ . Therefore, since  $k \leq n-1$  and  $k-1 \leq n-1$ , the binomial coefficients  $\binom{n-1}{k}$  and  $\binom{n-1}{k-1}$  are well-defined. We therefore have

$$\begin{aligned} \binom{n-1}{k} + \binom{n-1}{k-1} &= \frac{(n-1)!}{k!(n-k-1)!} + \frac{(n-1)!}{(k-1)!(n-k)!} \\ &= \frac{(n-1)!}{k!(n-k-1)!} \cdot \frac{n-k}{n-k} + \frac{(n-1)!}{(k-1)!(n-k)!} \cdot \frac{k}{k} \\ &= \frac{(n-k) \cdot (n-1)!}{k!(n-k)!} + \frac{k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n-k) \cdot (n-1)! + k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n \cdot (n-1)!)}{k!(n-k)!} = \frac{n!}{k!(n-k)!} = \binom{n}{k} \\ \therefore \binom{n}{k} &= \binom{n-1}{k} + \binom{n-1}{k-1} \quad \blacksquare \end{aligned}$$

(2) Following is pseudocode for a recursive BINOMIAL( $n, k$ ) algorithm based on the fact from (1):

BINOMIAL( $n, k$ ):

```

1  // base cases:  $k = 0$  or  $n = k$  imply  $\binom{n}{k} = 1$ 
2  if  $k == 0$  or  $n == k$ 
3      return 1
4  // recursive step
5  else return BINOMIAL( $n-1, k$ ) + BINOMIAL( $n-1, k-1$ )
```

(3) For  $n \geq 0$ , we define the property

2.2 5 / 5

✓ - 0 pts Correct

- 2 pts Incorrect

The running time of this algorithm is therefore  $\Theta(n)$ .

(3) Yes, this algorithm is asymptotically optimal, since any algorithm searching for the maximal subarray of the form  $A[1..j]$  must examine each element of the array (of length  $n$ ) at least once and will therefore have running time  $\Omega(n)$ .

**Question 2.** Recall that for  $0 \leq k \leq n$ , the **binomial coefficient**  $\binom{n}{k}$  is defined by

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}$$

In particular, we have  $\binom{n}{0} = \binom{n}{n} = 1$  for every  $n$ .

(1) Prove for every  $0 < k < n$ :

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

You can use any valid method you know to prove this (from the definition, combinatorial, generating function, etc.) (5pts)

(2) Write pseudocode for a recursive algorithm BINOMIAL( $n, k$ ) which returns  $\binom{n}{k}$ . Your algorithm should use the above fact you proved in (1). (5pts)

(3) Give a proof of correctness of your algorithm in (2). You should prove the statement: “For every  $n \geq 0$  and for every  $0 \leq k \leq n$ , BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$ .” (5pts)

*Solution.*

(1) Let  $k$  and  $n$  be such that  $0 < k < n$ . Therefore, since  $k \leq n-1$  and  $k-1 \leq n-1$ , the binomial coefficients  $\binom{n-1}{k}$  and  $\binom{n-1}{k-1}$  are well-defined. We therefore have

$$\begin{aligned} \binom{n-1}{k} + \binom{n-1}{k-1} &= \frac{(n-1)!}{k!(n-k-1)!} + \frac{(n-1)!}{(k-1)!(n-k)!} \\ &= \frac{(n-1)!}{k!(n-k-1)!} \cdot \frac{n-k}{n-k} + \frac{(n-1)!}{(k-1)!(n-k)!} \cdot \frac{k}{k} \\ &= \frac{(n-k) \cdot (n-1)!}{k!(n-k)!} + \frac{k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n-k) \cdot (n-1)! + k \cdot (n-1)!}{k!(n-k)!} \\ &= \frac{(n \cdot (n-1)!)}{k!(n-k)!} = \frac{n!}{k!(n-k)!} = \binom{n}{k} \\ \therefore \binom{n}{k} &= \binom{n-1}{k} + \binom{n-1}{k-1} \quad \blacksquare \end{aligned}$$

(2) Following is pseudocode for a recursive BINOMIAL( $n, k$ ) algorithm based on the fact from (1):

BINOMIAL( $n, k$ ):

```

1 // base cases:  $k = 0$  or  $n = k$  imply  $\binom{n}{k} = 1$ 
2 if  $k == 0$  or  $n == k$ 
3   return 1
4 // recursive step
5 else return BINOMIAL( $n-1, k$ ) + BINOMIAL( $n-1, k-1$ )
```

(3) For  $n \geq 0$ , we define the property

C(n): “BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$  for every  $k \in \mathbb{Z}$  such that  $0 \leq k \leq n$ .”

We show this is true for all  $n \geq 0$  by strong induction.

*Base case:* ( $n = 0$ ) Since  $0 \leq k \leq n$ , we only have one possible value of  $k$ , i.e.,  $k = 0$ . We therefore have

$$\binom{n}{k} = \binom{0}{0} = \frac{0!}{0! \cdot 0!} = 1$$

which, since  $n = k = 0$ , is what BINOMIAL( $0, 0$ ) returns in line 3.  $P(0)$  is therefore true.

*Inductive step:* Let  $n \geq 0$  such that  $P(0), \dots, P(n)$  holds. We show  $P(n+1)$  holds.

Let  $k$  be arbitrary such that  $0 \leq k \leq n+1$ . We have the following two cases for  $k$ :

*Case 1:* ( $k = 0$ ) In this case, we have

$$\binom{n+1}{k} = \binom{n+1}{0} = \frac{(n+1)!}{0! \cdot (n+1)!} = 1$$

which, since  $k = 0$ , is what BINOMIAL( $n+1, k$ ) returns in line 3. BINOMIAL( $n+1, k$ ) therefore returns  $\binom{n+1}{k}$  for  $k = 0$ .

*Case 2:* ( $k = n+1$ ) In this case, we have

$$\binom{n+1}{k} = \binom{n+1}{n+1} = \frac{(n+1)!}{(n+1)! \cdot 0!} = 1$$

which, since  $n+1 = k$ , is what BINOMIAL( $n+1, k$ ) returns in line 3. BINOMIAL( $n+1, k$ ) therefore returns  $\binom{n+1}{k}$  for  $k = n+1$ .

*Case 3:* ( $0 < k < n+1$ )<sup>1</sup> In this case, the condition in line 1 fails and we go to line 5. In line 5, we call BINOMIAL( $n, k$ ) and BINOMIAL( $n, k-1$ ). Since  $k < n+1$ , we have  $0 \leq k \leq n$  and  $0 \leq k-1 \leq n$ . By the inductive hypothesis, we know BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$  and BINOMIAL( $n, k-1$ ) returns  $\binom{n}{k-1}$ . Furthermore, from (1), we have

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

Therefore, in line 5, we return

$$\text{BINOMIAL}(n, k-1) + \text{BINOMIAL}(n, k) = \binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}$$

which is the desired value. BINOMIAL( $n+1, k$ ) therefore returns  $\binom{n+1}{k}$  for  $0 < k < n+1$ .

From cases 1, 2, and 3, we therefore know BINOMIAL( $n+1, k$ ) returns  $\binom{n+1}{k}$  for  $0 \leq k \leq n+1$ .

$\therefore P(n+1)$  is true. By the Principle of Induction we conclude that  $P(n)$  is true for  $n \in \mathbb{N}$ . ■

**Question 3.** Use a recursion tree and the substitution method to guess and verify an asymptotically tight bound for the following recurrence (5pts):

$$T(n) = 2T(n-1) + 1$$

Following is the recursion tree for the given recurrence relation, with the cost per level in the right column:

---

<sup>1</sup>Note that this case is only possible for  $n+1 \geq 2$ , and, thus far, we have only shown  $P(0)$  is true. However, cases 1 and 2 are sufficient to show  $P(1)$  is true, since the only values  $k$  can take for  $n+1 = 1$  are 0 (case 1) and 1 (case 2). Case 3 is therefore only to be considered for  $n+1 \geq 2$ .



2.3 5 / 5

✓ - 0 pts Correct

- 1 pts Error

- 2 pts Incorrect

C(n): “BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$  for every  $k \in \mathbb{Z}$  such that  $0 \leq k \leq n$ .”

We show this is true for all  $n \geq 0$  by strong induction.

*Base case:* ( $n = 0$ ) Since  $0 \leq k \leq n$ , we only have one possible value of  $k$ , i.e.,  $k = 0$ . We therefore have

$$\binom{n}{k} = \binom{0}{0} = \frac{0!}{0! \cdot 0!} = 1$$

which, since  $n = k = 0$ , is what BINOMIAL( $0, 0$ ) returns in line 3.  $P(0)$  is therefore true.

*Inductive step:* Let  $n \geq 0$  such that  $P(0), \dots, P(n)$  holds. We show  $P(n+1)$  holds.

Let  $k$  be arbitrary such that  $0 \leq k \leq n+1$ . We have the following two cases for  $k$ :

*Case 1:* ( $k = 0$ ) In this case, we have

$$\binom{n+1}{k} = \binom{n+1}{0} = \frac{(n+1)!}{0! \cdot (n+1)!} = 1$$

which, since  $k = 0$ , is what BINOMIAL( $n+1, k$ ) returns in line 3. BINOMIAL( $n+1, k$ ) therefore returns  $\binom{n+1}{k}$  for  $k = 0$ .

*Case 2:* ( $k = n+1$ ) In this case, we have

$$\binom{n+1}{k} = \binom{n+1}{n+1} = \frac{(n+1)!}{(n+1)! \cdot 0!} = 1$$

which, since  $n+1 = k$ , is what BINOMIAL( $n+1, k$ ) returns in line 3. BINOMIAL( $n+1, k$ ) therefore returns  $\binom{n+1}{k}$  for  $k = n+1$ .

*Case 3:* ( $0 < k < n+1$ )<sup>1</sup> In this case, the condition in line 1 fails and we go to line 5. In line 5, we call BINOMIAL( $n, k$ ) and BINOMIAL( $n, k-1$ ). Since  $k < n+1$ , we have  $0 \leq k \leq n$  and  $0 \leq k-1 \leq n$ . By the inductive hypothesis, we know BINOMIAL( $n, k$ ) returns  $\binom{n}{k}$  and BINOMIAL( $n, k-1$ ) returns  $\binom{n}{k-1}$ . Furthermore, from (1), we have

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

Therefore, in line 5, we return

$$\text{BINOMIAL}(n, k-1) + \text{BINOMIAL}(n, k) = \binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}$$

which is the desired value. BINOMIAL( $n+1, k$ ) therefore returns  $\binom{n+1}{k}$  for  $0 < k < n+1$ .

From cases 1, 2, and 3, we therefore know BINOMIAL( $n+1, k$ ) returns  $\binom{n+1}{k}$  for  $0 \leq k \leq n+1$ .

$\therefore P(n+1)$  is true. By the Principle of Induction we conclude that  $P(n)$  is true for  $n \in \mathbb{N}$ . ■

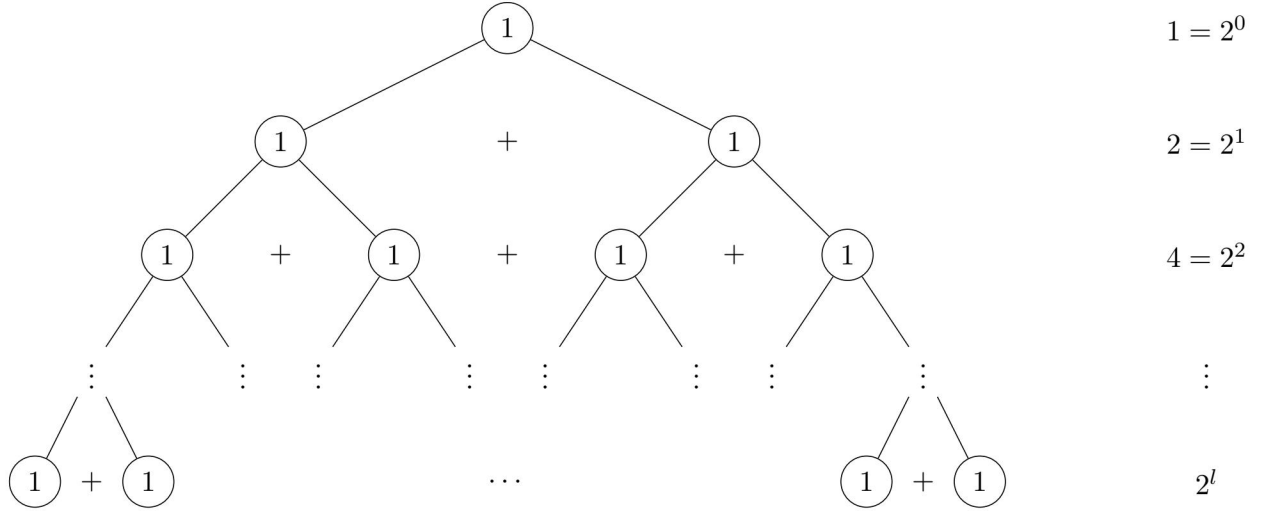
**Question 3.** Use a recursion tree and the substitution method to guess and verify an asymptotically tight bound for the following recurrence (5pts):

$$T(n) = 2T(n-1) + 1$$

Following is the recursion tree for the given recurrence relation, with the cost per level in the right column:

---

<sup>1</sup>Note that this case is only possible for  $n+1 \geq 2$ , and, thus far, we have only shown  $P(0)$  is true. However, cases 1 and 2 are sufficient to show  $P(1)$  is true, since the only values  $k$  can take for  $n+1 = 1$  are 0 (case 1) and 1 (case 2). Case 3 is therefore only to be considered for  $n+1 \geq 2$ .



On each level  $k$ , we call  $T(n-k)$   $2^k$  times. Assuming running  $T(0)$  costs 1, the leaves of the recursion tree each represent a  $T(0)$  call. In the last layer  $l$ , we therefore have  $T(n-l) = T(0) \implies l = n$ . We guess the total cost of  $T(n)$  is therefore given by

$$\sum_{k=0}^l 2^k = \sum_{k=0}^n 2^k = \frac{2^{n+1} - 1}{2 - 1} = 2^{n+1} = \Theta(2^n)$$

We now want to verify our guess by showing  $T(n) = \Theta(2^n)$ , i.e., we need to find some  $c_1, c_2 > 0$ ,  $n_0 \geq 0$  such that for all  $n \geq n_0$ ,  $c_1 2^n \leq T(n) \leq c_2 2^n$ .

We assume that there is some  $n_0 \geq 0$  such that for some  $c_1, c_2 > 0$ ,  $c_1 2^m \leq T(m) \leq c_2 2^m - k$  for all  $m < n$ .<sup>2</sup> We therefore have

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \implies 2c_1 \cdot 2^{n-1} + 1 \leq T(n) \leq 2 \cdot (c_2 2^{n-1} - k) + 1 \quad (\text{by assumption}) \\ &\implies c_1 2^n \leq T(n) \leq c_2 2^n - 2k + 1 \\ &\implies c_1 2^n \leq T(n) \leq c_2 2^n - k \end{aligned}$$

where the last inequality holds for  $k \geq 1$ . Let  $k := 1$ . We now consider the base case of  $n = 0$  with  $T(n) = 1$ :

$$c_1 2^n \leq T(n) \leq c_2 2^n - k \implies c_1 \leq 1 \leq c_2 - 1$$

which holds true for  $c_1 \leq 1$  and  $c_2 \geq 2$ . We also consider the base case  $n = 1$  with  $T(n) = 2T(n-1) + 1 = 2T(0) + 1 = 3$ :

$$c_1 2^n \leq T(n) \leq c_2 2^n - k \implies 2c_1 \leq 3 \leq 2c_2 - 1$$

which holds true for  $c_1 \leq 1.5$  and  $c_2 \geq 2$ . Choosing  $c_1 = 1$  and  $c_2 = 2$  should therefore satisfy our inductive hypothesis. We now show directly by induction (tracing our steps backwards) that  $c_1 2^n \leq T(n) \leq c_2 2^n$  for all  $n \geq 0$ .

We have already shown for chosen  $c_1, c_2$  that for  $n = 0$  and  $n = 1$ ,  $c_1 2^n \leq T(n) \leq c_2 2^n - 1$ .

*Inductive step:* Let  $n_0 \geq 0$  be such that for  $n = n_0$ ,  $2^n \leq T(n) \leq 2 \cdot 2^n - 1$ . We show the inequality also holds for  $n = n_0 + 1$ :

$$\begin{aligned} T(n_0 + 1) &= 2T(n_0) + 1 \implies 2 \cdot 2^{n_0} + 1 \leq T(n_0 + 1) \leq 2 \cdot (2^{n_0+1} - 1) + 1 \quad (\text{by assumption on } T(n_0)) \\ &\implies 2^{n_0+1} \leq T(n_0 + 1) \leq 2 \cdot 2^{n_0+1} - 1 \end{aligned}$$

<sup>2</sup>We subtract  $k$  to account for  $+1$  in the recurrence relation.

By the Principle of Induction, we therefore have for all  $n \geq 0$

$$2^n \leq T(n) \leq 2 \cdot 2^n - 1$$

We therefore have  $2^n \leq T(n) \leq 2 \cdot 2^n$  for all  $n \geq 0$ . By definition of  $\Theta$ -notation,  $T(n) = \Theta(2^n)$ . ■

**Question 4.** For the following recurrence determine an asymptotically tight bound using any method (recursion tree and substitution, master method, etc.). (5pts)

$$T(n) = 25T(n/5) + \frac{n^2}{\lg n}$$

*Solution.* We find an asymptotically tight bound for  $T(n)$  using the Master Theorem. From the recursion relation, we have  $a = 25$ ,  $b = 5$ , and  $f(n) = n^2 \lg^{-1} n$ . We therefore have

$$f(n) = n^2 \lg^{-1} n = n^{\log_5 25} \lg^{-1} n = n^{\log_b a} \lg^{-1} n$$

The recursion relation is therefore characterised by the middle case of the Master Theorem for  $k = -1$ . We therefore have, from the Master Theorem,

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a} \lg \lg n) = \Theta(n^{\log_5 25} \lg \lg n) = \Theta(n^2 \lg \lg n) \\ \therefore T(n) &= \Theta(n^2 \lg \lg n) \end{aligned}$$

**Question 5.** For the following functions  $f(n)$  and  $g(n)$ , determine whether they satisfy:

- (1)  $f(n) = o(g(n))$ ,
- (2)  $f(n) = \Theta(g(n))$ , or
- (3)  $f(n) = \omega(g(n))$ .

The functions are:

$$f(n) = (\lg n)^{\sqrt{\lg n}} \quad \text{and} \quad g(n) = \sqrt{n}$$

Justify your answer. (5pts)

*Solution.* We show  $f(n) = o(g(n))$ .

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \lim_{n \rightarrow \infty} \frac{(\lg n)^{\sqrt{\lg n}}}{\sqrt{n}} = \lim_{n \rightarrow \infty} \text{two} \left( \lg \left( \frac{(\lg n)^{\sqrt{\lg n}}}{\sqrt{n}} \right) \right) \quad (\text{for } \text{two}(n) := 2^n) \\ &= \text{two} \left( \lim_{n \rightarrow \infty} \left( \sqrt{\lg n} \lg \lg n - \frac{1}{2} \lg n \right) \right) \\ &= \text{two} \left( \lim_{n \rightarrow \infty} \frac{1}{2} \lg n \left( \frac{2 \lg \lg n}{\sqrt{\lg n}} - 1 \right) \right) \\ &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \quad (\text{for } k := \lg n) \end{aligned}$$

Since  $\lim_{k \rightarrow \infty} 2 \lg k = \infty$  and  $\lim_{k \rightarrow \infty} \sqrt{k} = \infty$ , we can apply L'Hôpital's Rule to show  $\lim_{k \rightarrow \infty} 2 \lg k / \sqrt{k}$  exists and equals 0:

$$\lim_{k \rightarrow \infty} \frac{2 \lg k}{\sqrt{k}} = \lim_{k \rightarrow \infty} 2 \cdot \frac{1}{k} \cdot \frac{2\sqrt{k}}{1} = 4 \lim_{n \rightarrow \infty} \frac{1}{\sqrt{k}} = 0 \implies \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) = -1$$

Applying this limit, we get

3 5 / 5

- 0 pts Correct

✓ - 0 pts Assume a particular value for  $T(0)$  or  $T(1)$

- 0.5 pts Upper bound only

- 1 pts Derive consequence different than inductive hypothesis

- 0.5 pts Neglect constant term

- 1.5 pts Incorrect induction

- 1 pts Only prove lower bound by substitution

- 1 pts Recursion tree drawn but seemingly not used

- 4 pts Incorrect recursion tree and bound, no proof

- 1 pts Repeatedly substitute in recurrence instead of using substitution method

- 2 pts Repeated substitution; don't use recursion tree

1  $T(0) \neq 1$ .

By the Principle of Induction, we therefore have for all  $n \geq 0$

$$2^n \leq T(n) \leq 2 \cdot 2^n - 1$$

We therefore have  $2^n \leq T(n) \leq 2 \cdot 2^n$  for all  $n \geq 0$ . By definition of  $\Theta$ -notation,  $T(n) = \Theta(2^n)$ . ■

**Question 4.** For the following recurrence determine an asymptotically tight bound using any method (recursion tree and substitution, master method, etc.). (5pts)

$$T(n) = 25T(n/5) + \frac{n^2}{\lg n}$$

*Solution.* We find an asymptotically tight bound for  $T(n)$  using the Master Theorem. From the recursion relation, we have  $a = 25$ ,  $b = 5$ , and  $f(n) = n^2 \lg^{-1} n$ . We therefore have

$$f(n) = n^2 \lg^{-1} n = n^{\log_5 25} \lg^{-1} n = n^{\log_b a} \lg^{-1} n$$

The recursion relation is therefore characterised by the middle case of the Master Theorem for  $k = -1$ . We therefore have, from the Master Theorem,

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a} \lg \lg n) = \Theta(n^{\log_5 25} \lg \lg n) = \Theta(n^2 \lg \lg n) \\ \therefore T(n) &= \Theta(n^2 \lg \lg n) \end{aligned}$$

**Question 5.** For the following functions  $f(n)$  and  $g(n)$ , determine whether they satisfy:

- (1)  $f(n) = o(g(n))$ ,
- (2)  $f(n) = \Theta(g(n))$ , or
- (3)  $f(n) = \omega(g(n))$ .

The functions are:

$$f(n) = (\lg n)^{\sqrt{\lg n}} \quad \text{and} \quad g(n) = \sqrt{n}$$

Justify your answer. (5pts)

*Solution.* We show  $f(n) = o(g(n))$ .

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \lim_{n \rightarrow \infty} \frac{(\lg n)^{\sqrt{\lg n}}}{\sqrt{n}} = \lim_{n \rightarrow \infty} \text{two} \left( \lg \left( \frac{(\lg n)^{\sqrt{\lg n}}}{\sqrt{n}} \right) \right) \quad (\text{for } \text{two}(n) := 2^n) \\ &= \text{two} \left( \lim_{n \rightarrow \infty} \left( \sqrt{\lg n} \lg \lg n - \frac{1}{2} \lg n \right) \right) \\ &= \text{two} \left( \lim_{n \rightarrow \infty} \frac{1}{2} \lg n \left( \frac{2 \lg \lg n}{\sqrt{\lg n}} - 1 \right) \right) \\ &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \quad (\text{for } k := \lg n) \end{aligned}$$

Since  $\lim_{k \rightarrow \infty} 2 \lg k = \infty$  and  $\lim_{k \rightarrow \infty} \sqrt{k} = \infty$ , we can apply L'Hôpital's Rule to show  $\lim_{k \rightarrow \infty} 2 \lg k / \sqrt{k}$  exists and equals 0:

$$\lim_{k \rightarrow \infty} \frac{2 \lg k}{\sqrt{k}} = \lim_{k \rightarrow \infty} 2 \cdot \frac{1}{k} \cdot \frac{2\sqrt{k}}{1} = 4 \lim_{n \rightarrow \infty} \frac{1}{\sqrt{k}} = 0 \implies \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) = -1$$

Applying this limit, we get

4 5 / 5

✓ - 0 pts Correct

- 2 pts Incorrect

- 1 pts Bounds not tight

By the Principle of Induction, we therefore have for all  $n \geq 0$

$$2^n \leq T(n) \leq 2 \cdot 2^n - 1$$

We therefore have  $2^n \leq T(n) \leq 2 \cdot 2^n$  for all  $n \geq 0$ . By definition of  $\Theta$ -notation,  $T(n) = \Theta(2^n)$ . ■

**Question 4.** For the following recurrence determine an asymptotically tight bound using any method (recursion tree and substitution, master method, etc.). (5pts)

$$T(n) = 25T(n/5) + \frac{n^2}{\lg n}$$

*Solution.* We find an asymptotically tight bound for  $T(n)$  using the Master Theorem. From the recursion relation, we have  $a = 25$ ,  $b = 5$ , and  $f(n) = n^2 \lg^{-1} n$ . We therefore have

$$f(n) = n^2 \lg^{-1} n = n^{\log_5 25} \lg^{-1} n = n^{\log_b a} \lg^{-1} n$$

The recursion relation is therefore characterised by the middle case of the Master Theorem for  $k = -1$ . We therefore have, from the Master Theorem,

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a} \lg \lg n) = \Theta(n^{\log_5 25} \lg \lg n) = \Theta(n^2 \lg \lg n) \\ \therefore T(n) &= \Theta(n^2 \lg \lg n) \end{aligned}$$

**Question 5.** For the following functions  $f(n)$  and  $g(n)$ , determine whether they satisfy:

- (1)  $f(n) = o(g(n))$ ,
- (2)  $f(n) = \Theta(g(n))$ , or
- (3)  $f(n) = \omega(g(n))$ .

The functions are:

$$f(n) = (\lg n)^{\sqrt{\lg n}} \quad \text{and} \quad g(n) = \sqrt{n}$$

Justify your answer. (5pts)

*Solution.* We show  $f(n) = o(g(n))$ .

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \lim_{n \rightarrow \infty} \frac{(\lg n)^{\sqrt{\lg n}}}{\sqrt{n}} = \lim_{n \rightarrow \infty} \text{two} \left( \lg \left( \frac{(\lg n)^{\sqrt{\lg n}}}{\sqrt{n}} \right) \right) \quad (\text{for } \text{two}(n) := 2^n) \\ &= \text{two} \left( \lim_{n \rightarrow \infty} \left( \sqrt{\lg n} \lg \lg n - \frac{1}{2} \lg n \right) \right) \\ &= \text{two} \left( \lim_{n \rightarrow \infty} \frac{1}{2} \lg n \left( \frac{2 \lg \lg n}{\sqrt{\lg n}} - 1 \right) \right) \\ &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \quad (\text{for } k := \lg n) \end{aligned}$$

Since  $\lim_{k \rightarrow \infty} 2 \lg k = \infty$  and  $\lim_{k \rightarrow \infty} \sqrt{k} = \infty$ , we can apply L'Hôpital's Rule to show  $\lim_{k \rightarrow \infty} 2 \lg k / \sqrt{k}$  exists and equals 0:

$$\lim_{k \rightarrow \infty} \frac{2 \lg k}{\sqrt{k}} = \lim_{k \rightarrow \infty} 2 \cdot \frac{1}{k} \cdot \frac{2\sqrt{k}}{1} = 4 \lim_{n \rightarrow \infty} \frac{1}{\sqrt{k}} = 0 \implies \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) = -1$$

Applying this limit, we get



$$\begin{aligned}
\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \\
&= \text{two} \left( \lim_{n \rightarrow \infty} k \cdot \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \quad (\text{the limit exists}) \\
&= \text{two} \left( \lim_{k \rightarrow \infty} k \cdot (-1) \right) = \text{two} \left( \lim_{k \rightarrow \infty} -k \right) \\
&= \text{two}(-\infty) = 0 \\
\implies f(n) &= o(g(n)) \quad \blacksquare \quad (\text{by definition of } o\text{-notation})
\end{aligned}$$

**Question 6.** (True/False) For each of the following statements indicate whether they are **true** or **false**. Each question is worth 2pts, a blank answer will receive 1pt. Recall that “true” means “always true” and “false” means “there exists a counterexample”.

- (1) For every  $n \geq 1$  and  $a, b \in \mathbb{Z}$ , if  $ab \bmod n = 0$ , then either  $a \bmod n = 0$  or  $b \bmod n = 0$ .
- (2) Let  $(F_n)_{n \geq 0}$  be the sequence of Fibonacci numbers, so  $F_0 = 0, F_1 = 1$  and for every  $n \geq 2$ ,  $F_n = F_{n-1} + F_{n-2}$ . Then for every  $n \geq 2$ ,  $F_{2n} = F_{2(n-1)} + F_{2(n-2)}$ .
- (3) Suppose  $f(n)$  and  $g(n)$  are asymptotically positive, polynomially bounded functions. If  $f(n) = \Theta(g(n))$ , then  $2^{2^{f(n)}} = \Theta(2^{2^{g(n)}})$ .
- (4)  $\Omega(n) = O(n^2)$ .
- (5) The best-case running time of INSERTION-SORT is  $O(n \lg n)$ .
- (6) MERGE-SORT is an asymptotically optimal comparison-based sorting algorithm.

*Solution.*

- (1) False. We present a counter-example: let  $a := 2 \in \mathbb{Z}$ ,  $b := 2 \in \mathbb{Z}$  and  $n := 4 \geq 1$ . We have  $ab \bmod n = 4 \bmod 4 = 0$ , but  $a \bmod n = 2 \bmod 4 = 2 \neq 0$  and  $b \bmod n = 2 \bmod 4 = 2 \neq 0$ .
- (2) False. We present a counter-example: let  $n := 2 \geq 2$ . We have  $F_{2n} = F_4 = 3$ , but  $F_{2(n-1)} + F_{2(n-2)} = F_2 + F_0 = 1 + 0 = 1 \neq 3$ .
- (3) False. We present a counter-example: let  $g(n) := 2n$ ,  $f(n) := n = \Theta(g(n))$ . Both  $f(n)$  and  $g(n)$  are asymptotically positive and polynomially bounded functions. We show  $f(n) = o(g(n))$ :

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^{2n}}} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^n \cdot 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n \cdot 2^n - 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n(2^n - 1)}} = 0$$

We therefore have  $f(n) = o(g(n))$ , and therefore,  $f(n) \neq \Theta(g(n))$ .

- (4) False. We present a counter-example: let  $f(n) = n^3$ . We therefore clearly have  $f(n) = \Omega(n)$ , but  $f(n) \neq O(n^2)$ .
- (5) True. The best-case running time of INSERTION-SORT is  $\Theta(n)$ , which in turn is  $O(n \lg n)$ .
- (6) True. From Theorem 3.3.1 (proven in class), we know any comparison-based sorting algorithm performs at  $\Omega(n \lg n)$ . Since the running time of MERGE-SORT is  $\Theta(n \lg n)$ , there does not exist any comparison-based sorting algorithm that is asymptotically better than MERGE-SORT.

5 5 / 5

✓ - 0 pts Correct

- 1 pts Correct answer, gap in proof
- 2 pts Correct answer, error in proof
- 3 pts Correct answer, proof just reformulates the desired result and asserts truth
- 3 pts Correct answer, incorrect proof
- 3 pts Correct answer, no proof
- 3 pts Partially correct answer, reformulated and asserted
- 1 pts Incorrect answer from careless error
- 2 pts Incorrect answer, hole in proof
- 4 pts Incorrect answer, reformulated and asserted
- 4 pts Incorrect answer, incorrect proof.
- 5 pts Incorrect answer, no justification

$$\begin{aligned}
\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \\
&= \text{two} \left( \lim_{n \rightarrow \infty} k \cdot \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) && \text{(the limit exists)} \\
&= \text{two} \left( \lim_{k \rightarrow \infty} k \cdot (-1) \right) = \text{two} \left( \lim_{k \rightarrow \infty} -k \right) \\
&= \text{two}(-\infty) = 0 \\
\implies f(n) &= o(g(n)) \quad \blacksquare && \text{(by definition of } o\text{-notation)}
\end{aligned}$$

**Question 6.** (True/False) For each of the following statements indicate whether they are **true** or **false**. Each question is worth 2pts, a blank answer will receive 1pt. Recall that “true” means “always true” and “false” means “there exists a counterexample”.

- (1) For every  $n \geq 1$  and  $a, b \in \mathbb{Z}$ , if  $ab \bmod n = 0$ , then either  $a \bmod n = 0$  or  $b \bmod n = 0$ .
- (2) Let  $(F_n)_{n \geq 0}$  be the sequence of Fibonacci numbers, so  $F_0 = 0, F_1 = 1$  and for every  $n \geq 2$ ,  $F_n = F_{n-1} + F_{n-2}$ . Then for every  $n \geq 2$ ,  $F_{2n} = F_{2(n-1)} + F_{2(n-2)}$ .
- (3) Suppose  $f(n)$  and  $g(n)$  are asymptotically positive, polynomially bounded functions. If  $f(n) = \Theta(g(n))$ , then  $2^{2^{f(n)}} = \Theta(2^{2^{g(n)}})$ .
- (4)  $\Omega(n) = O(n^2)$ .
- (5) The best-case running time of INSERTION-SORT is  $O(n \lg n)$ .
- (6) MERGE-SORT is an asymptotically optimal comparison-based sorting algorithm.

*Solution.*

- (1) False. We present a counter-example: let  $a := 2 \in \mathbb{Z}$ ,  $b := 2 \in \mathbb{Z}$  and  $n := 4 \geq 1$ . We have  $ab \bmod n = 4 \bmod 4 = 0$ , but  $a \bmod n = 2 \bmod 4 = 2 \neq 0$  and  $b \bmod n = 2 \bmod 4 = 2 \neq 0$ .
- (2) False. We present a counter-example: let  $n := 2 \geq 2$ . We have  $F_{2n} = F_4 = 3$ , but  $F_{2(n-1)} + F_{2(n-2)} = F_2 + F_0 = 1 + 0 = 1 \neq 3$ .
- (3) False. We present a counter-example: let  $g(n) := 2n$ ,  $f(n) := n = \Theta(g(n))$ . Both  $f(n)$  and  $g(n)$  are asymptotically positive and polynomially bounded functions. We show  $f(n) = o(g(n))$ :

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^{2n}}} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^n \cdot 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n \cdot 2^n - 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n(2^n - 1)}} = 0$$

We therefore have  $f(n) = o(g(n))$ , and therefore,  $f(n) \neq \Theta(g(n))$ .

- (4) False. We present a counter-example: let  $f(n) = n^3$ . We therefore clearly have  $f(n) = \Omega(n)$ , but  $f(n) \neq O(n^2)$ .
- (5) True. The best-case running time of INSERTION-SORT is  $\Theta(n)$ , which in turn is  $O(n \lg n)$ .
- (6) True. From Theorem 3.3.1 (proven in class), we know any comparison-based sorting algorithm performs at  $\Omega(n \lg n)$ . Since the running time of MERGE-SORT is  $\Theta(n \lg n)$ , there does not exist any comparison-based sorting algorithm that is asymptotically better than MERGE-SORT.

6.1 2 / 2

- ✓ - 0 pts Correct
- 2 pts Incorrect
- 1 pts No answer

$$\begin{aligned}
\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \\
&= \text{two} \left( \lim_{n \rightarrow \infty} k \cdot \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) && \text{(the limit exists)} \\
&= \text{two} \left( \lim_{k \rightarrow \infty} k \cdot (-1) \right) = \text{two} \left( \lim_{k \rightarrow \infty} -k \right) \\
&= \text{two}(-\infty) = 0 \\
\implies f(n) &= o(g(n)) \quad \blacksquare && \text{(by definition of } o\text{-notation)}
\end{aligned}$$

**Question 6.** (True/False) For each of the following statements indicate whether they are **true** or **false**. Each question is worth 2pts, a blank answer will receive 1pt. Recall that “true” means “always true” and “false” means “there exists a counterexample”.

- (1) For every  $n \geq 1$  and  $a, b \in \mathbb{Z}$ , if  $ab \bmod n = 0$ , then either  $a \bmod n = 0$  or  $b \bmod n = 0$ .
- (2) Let  $(F_n)_{n \geq 0}$  be the sequence of Fibonacci numbers, so  $F_0 = 0, F_1 = 1$  and for every  $n \geq 2$ ,  $F_n = F_{n-1} + F_{n-2}$ . Then for every  $n \geq 2$ ,  $F_{2n} = F_{2(n-1)} + F_{2(n-2)}$ .
- (3) Suppose  $f(n)$  and  $g(n)$  are asymptotically positive, polynomially bounded functions. If  $f(n) = \Theta(g(n))$ , then  $2^{2^{f(n)}} = \Theta(2^{2^{g(n)}})$ .
- (4)  $\Omega(n) = O(n^2)$ .
- (5) The best-case running time of INSERTION-SORT is  $O(n \lg n)$ .
- (6) MERGE-SORT is an asymptotically optimal comparison-based sorting algorithm.

*Solution.*

- (1) False. We present a counter-example: let  $a := 2 \in \mathbb{Z}$ ,  $b := 2 \in \mathbb{Z}$  and  $n := 4 \geq 1$ . We have  $ab \bmod n = 4 \bmod 4 = 0$ , but  $a \bmod n = 2 \bmod 4 = 2 \neq 0$  and  $b \bmod n = 2 \bmod 4 = 2 \neq 0$ .
- (2) False. We present a counter-example: let  $n := 2 \geq 2$ . We have  $F_{2n} = F_4 = 3$ , but  $F_{2(n-1)} + F_{2(n-2)} = F_2 + F_0 = 1 + 0 = 1 \neq 3$ .
- (3) False. We present a counter-example: let  $g(n) := 2n$ ,  $f(n) := n = \Theta(g(n))$ . Both  $f(n)$  and  $g(n)$  are asymptotically positive and polynomially bounded functions. We show  $f(n) = o(g(n))$ :

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^{2n}}} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^n \cdot 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n \cdot 2^n - 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n(2^n - 1)}} = 0$$

We therefore have  $f(n) = o(g(n))$ , and therefore,  $f(n) \neq \Theta(g(n))$ .

- (4) False. We present a counter-example: let  $f(n) = n^3$ . We therefore clearly have  $f(n) = \Omega(n)$ , but  $f(n) \neq O(n^2)$ .
- (5) True. The best-case running time of INSERTION-SORT is  $\Theta(n)$ , which in turn is  $O(n \lg n)$ .
- (6) True. From Theorem 3.3.1 (proven in class), we know any comparison-based sorting algorithm performs at  $\Omega(n \lg n)$ . Since the running time of MERGE-SORT is  $\Theta(n \lg n)$ , there does not exist any comparison-based sorting algorithm that is asymptotically better than MERGE-SORT.

6.2 2 / 2

- ✓ - 0 pts Correct
- 2 pts Incorrect
- 1 pts No answer

$$\begin{aligned}
\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \\
&= \text{two} \left( \lim_{n \rightarrow \infty} k \cdot \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \quad (\text{the limit exists}) \\
&= \text{two} \left( \lim_{k \rightarrow \infty} k \cdot (-1) \right) = \text{two} \left( \lim_{k \rightarrow \infty} -k \right) \\
&= \text{two}(-\infty) = 0 \\
\implies f(n) &= o(g(n)) \quad \blacksquare \quad (\text{by definition of } o\text{-notation})
\end{aligned}$$

**Question 6.** (True/False) For each of the following statements indicate whether they are **true** or **false**. Each question is worth 2pts, a blank answer will receive 1pt. Recall that “true” means “always true” and “false” means “there exists a counterexample”.

- (1) For every  $n \geq 1$  and  $a, b \in \mathbb{Z}$ , if  $ab \bmod n = 0$ , then either  $a \bmod n = 0$  or  $b \bmod n = 0$ .
- (2) Let  $(F_n)_{n \geq 0}$  be the sequence of Fibonacci numbers, so  $F_0 = 0, F_1 = 1$  and for every  $n \geq 2$ ,  $F_n = F_{n-1} + F_{n-2}$ . Then for every  $n \geq 2$ ,  $F_{2n} = F_{2(n-1)} + F_{2(n-2)}$ .
- (3) Suppose  $f(n)$  and  $g(n)$  are asymptotically positive, polynomially bounded functions. If  $f(n) = \Theta(g(n))$ , then  $2^{2^{f(n)}} = \Theta(2^{2^{g(n)}})$ .
- (4)  $\Omega(n) = O(n^2)$ .
- (5) The best-case running time of INSERTION-SORT is  $O(n \lg n)$ .
- (6) MERGE-SORT is an asymptotically optimal comparison-based sorting algorithm.

*Solution.*

- (1) False. We present a counter-example: let  $a := 2 \in \mathbb{Z}$ ,  $b := 2 \in \mathbb{Z}$  and  $n := 4 \geq 1$ . We have  $ab \bmod n = 4 \bmod 4 = 0$ , but  $a \bmod n = 2 \bmod 4 = 2 \neq 0$  and  $b \bmod n = 2 \bmod 4 = 2 \neq 0$ .
- (2) False. We present a counter-example: let  $n := 2 \geq 2$ . We have  $F_{2n} = F_4 = 3$ , but  $F_{2(n-1)} + F_{2(n-2)} = F_2 + F_0 = 1 + 0 = 1 \neq 3$ .
- (3) False. We present a counter-example: let  $g(n) := 2n$ ,  $f(n) := n = \Theta(g(n))$ . Both  $f(n)$  and  $g(n)$  are asymptotically positive and polynomially bounded functions. We show  $f(n) = o(g(n))$ :

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^{2n}}} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^n \cdot 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n \cdot 2^n - 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n(2^n - 1)}} = 0$$

We therefore have  $f(n) = o(g(n))$ , and therefore,  $f(n) \neq \Theta(g(n))$ .

- (4) False. We present a counter-example: let  $f(n) = n^3$ . We therefore clearly have  $f(n) = \Omega(n)$ , but  $f(n) \neq O(n^2)$ .
- (5) True. The best-case running time of INSERTION-SORT is  $\Theta(n)$ , which in turn is  $O(n \lg n)$ .
- (6) True. From Theorem 3.3.1 (proven in class), we know any comparison-based sorting algorithm performs at  $\Omega(n \lg n)$ . Since the running time of MERGE-SORT is  $\Theta(n \lg n)$ , there does not exist any comparison-based sorting algorithm that is asymptotically better than MERGE-SORT.

6.3 2 / 2

- ✓ - 0 pts Correct
- 2 pts Incorrect
- 1 pts No answer



$$\begin{aligned}
\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \\
&= \text{two} \left( \lim_{n \rightarrow \infty} k \cdot \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) && \text{(the limit exists)} \\
&= \text{two} \left( \lim_{k \rightarrow \infty} k \cdot (-1) \right) = \text{two} \left( \lim_{k \rightarrow \infty} -k \right) \\
&= \text{two}(-\infty) = 0 \\
\implies f(n) &= o(g(n)) \quad \blacksquare && \text{(by definition of } o\text{-notation)}
\end{aligned}$$

**Question 6.** (True/False) For each of the following statements indicate whether they are **true** or **false**. Each question is worth 2pts, a blank answer will receive 1pt. Recall that “true” means “always true” and “false” means “there exists a counterexample”.

- (1) For every  $n \geq 1$  and  $a, b \in \mathbb{Z}$ , if  $ab \bmod n = 0$ , then either  $a \bmod n = 0$  or  $b \bmod n = 0$ .
- (2) Let  $(F_n)_{n \geq 0}$  be the sequence of Fibonacci numbers, so  $F_0 = 0, F_1 = 1$  and for every  $n \geq 2$ ,  $F_n = F_{n-1} + F_{n-2}$ . Then for every  $n \geq 2$ ,  $F_{2n} = F_{2(n-1)} + F_{2(n-2)}$ .
- (3) Suppose  $f(n)$  and  $g(n)$  are asymptotically positive, polynomially bounded functions. If  $f(n) = \Theta(g(n))$ , then  $2^{2^{f(n)}} = \Theta(2^{2^{g(n)}})$ .
- (4)  $\Omega(n) = O(n^2)$ .
- (5) The best-case running time of INSERTION-SORT is  $O(n \lg n)$ .
- (6) MERGE-SORT is an asymptotically optimal comparison-based sorting algorithm.

*Solution.*

- (1) False. We present a counter-example: let  $a := 2 \in \mathbb{Z}$ ,  $b := 2 \in \mathbb{Z}$  and  $n := 4 \geq 1$ . We have  $ab \bmod n = 4 \bmod 4 = 0$ , but  $a \bmod n = 2 \bmod 4 = 2 \neq 0$  and  $b \bmod n = 2 \bmod 4 = 2 \neq 0$ .
- (2) False. We present a counter-example: let  $n := 2 \geq 2$ . We have  $F_{2n} = F_4 = 3$ , but  $F_{2(n-1)} + F_{2(n-2)} = F_2 + F_0 = 1 + 0 = 1 \neq 3$ .
- (3) False. We present a counter-example: let  $g(n) := 2n$ ,  $f(n) := n = \Theta(g(n))$ . Both  $f(n)$  and  $g(n)$  are asymptotically positive and polynomially bounded functions. We show  $f(n) = o(g(n))$ :

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^{2n}}} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^n \cdot 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n \cdot 2^n - 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n(2^n - 1)}} = 0$$

We therefore have  $f(n) = o(g(n))$ , and therefore,  $f(n) \neq \Theta(g(n))$ .

- (4) False. We present a counter-example: let  $f(n) = n^3$ . We therefore clearly have  $f(n) = \Omega(n)$ , but  $f(n) \neq O(n^2)$ .
- (5) True. The best-case running time of INSERTION-SORT is  $\Theta(n)$ , which in turn is  $O(n \lg n)$ .
- (6) True. From Theorem 3.3.1 (proven in class), we know any comparison-based sorting algorithm performs at  $\Omega(n \lg n)$ . Since the running time of MERGE-SORT is  $\Theta(n \lg n)$ , there does not exist any comparison-based sorting algorithm that is asymptotically better than MERGE-SORT.

6.4 2 / 2

- ✓ - 0 pts Correct
- 2 pts Incorrect
- 1 pts No answer

$$\begin{aligned}
\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \\
&= \text{two} \left( \lim_{n \rightarrow \infty} k \cdot \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \quad (\text{the limit exists}) \\
&= \text{two} \left( \lim_{k \rightarrow \infty} k \cdot (-1) \right) = \text{two} \left( \lim_{k \rightarrow \infty} -k \right) \\
&= \text{two}(-\infty) = 0 \\
\implies f(n) &= o(g(n)) \quad \blacksquare \quad (\text{by definition of } o\text{-notation})
\end{aligned}$$

**Question 6.** (True/False) For each of the following statements indicate whether they are **true** or **false**. Each question is worth 2pts, a blank answer will receive 1pt. Recall that “true” means “always true” and “false” means “there exists a counterexample”.

- (1) For every  $n \geq 1$  and  $a, b \in \mathbb{Z}$ , if  $ab \bmod n = 0$ , then either  $a \bmod n = 0$  or  $b \bmod n = 0$ .
- (2) Let  $(F_n)_{n \geq 0}$  be the sequence of Fibonacci numbers, so  $F_0 = 0, F_1 = 1$  and for every  $n \geq 2$ ,  $F_n = F_{n-1} + F_{n-2}$ . Then for every  $n \geq 2$ ,  $F_{2n} = F_{2(n-1)} + F_{2(n-2)}$ .
- (3) Suppose  $f(n)$  and  $g(n)$  are asymptotically positive, polynomially bounded functions. If  $f(n) = \Theta(g(n))$ , then  $2^{2^{f(n)}} = \Theta(2^{2^{g(n)}})$ .
- (4)  $\Omega(n) = O(n^2)$ .
- (5) The best-case running time of INSERTION-SORT is  $O(n \lg n)$ .
- (6) MERGE-SORT is an asymptotically optimal comparison-based sorting algorithm.

*Solution.*

- (1) False. We present a counter-example: let  $a := 2 \in \mathbb{Z}$ ,  $b := 2 \in \mathbb{Z}$  and  $n := 4 \geq 1$ . We have  $ab \bmod n = 4 \bmod 4 = 0$ , but  $a \bmod n = 2 \bmod 4 = 2 \neq 0$  and  $b \bmod n = 2 \bmod 4 = 2 \neq 0$ .
- (2) False. We present a counter-example: let  $n := 2 \geq 2$ . We have  $F_{2n} = F_4 = 3$ , but  $F_{2(n-1)} + F_{2(n-2)} = F_2 + F_0 = 1 + 0 = 1 \neq 3$ .
- (3) False. We present a counter-example: let  $g(n) := 2n$ ,  $f(n) := n = \Theta(g(n))$ . Both  $f(n)$  and  $g(n)$  are asymptotically positive and polynomially bounded functions. We show  $f(n) = o(g(n))$ :

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^{2n}}} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^n \cdot 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n \cdot 2^n - 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n(2^n - 1)}} = 0$$

We therefore have  $f(n) = o(g(n))$ , and therefore,  $f(n) \neq \Theta(g(n))$ .

- (4) False. We present a counter-example: let  $f(n) = n^3$ . We therefore clearly have  $f(n) = \Omega(n)$ , but  $f(n) \neq O(n^2)$ .
- (5) True. The best-case running time of INSERTION-SORT is  $\Theta(n)$ , which in turn is  $O(n \lg n)$ .
- (6) True. From Theorem 3.3.1 (proven in class), we know any comparison-based sorting algorithm performs at  $\Omega(n \lg n)$ . Since the running time of MERGE-SORT is  $\Theta(n \lg n)$ , there does not exist any comparison-based sorting algorithm that is asymptotically better than MERGE-SORT.

6.5 2 / 2

- ✓ - 0 pts Correct
- 2 pts Incorrect
- 1 pts No answer

$$\begin{aligned}
\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &= \text{two} \left( \lim_{k \rightarrow \infty} \frac{k}{2} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) \\
&= \text{two} \left( \lim_{n \rightarrow \infty} k \cdot \lim_{n \rightarrow \infty} \left( \frac{2 \lg k}{\sqrt{k}} - 1 \right) \right) && \text{(the limit exists)} \\
&= \text{two} \left( \lim_{k \rightarrow \infty} k \cdot (-1) \right) = \text{two} \left( \lim_{k \rightarrow \infty} -k \right) \\
&= \text{two}(-\infty) = 0 \\
\implies f(n) &= o(g(n)) \quad \blacksquare && \text{(by definition of } o\text{-notation)}
\end{aligned}$$

**Question 6.** (True/False) For each of the following statements indicate whether they are **true** or **false**. Each question is worth 2pts, a blank answer will receive 1pt. Recall that “true” means “always true” and “false” means “there exists a counterexample”.

- (1) For every  $n \geq 1$  and  $a, b \in \mathbb{Z}$ , if  $ab \bmod n = 0$ , then either  $a \bmod n = 0$  or  $b \bmod n = 0$ .
- (2) Let  $(F_n)_{n \geq 0}$  be the sequence of Fibonacci numbers, so  $F_0 = 0, F_1 = 1$  and for every  $n \geq 2$ ,  $F_n = F_{n-1} + F_{n-2}$ . Then for every  $n \geq 2$ ,  $F_{2n} = F_{2(n-1)} + F_{2(n-2)}$ .
- (3) Suppose  $f(n)$  and  $g(n)$  are asymptotically positive, polynomially bounded functions. If  $f(n) = \Theta(g(n))$ , then  $2^{2^{f(n)}} = \Theta(2^{2^{g(n)}})$ .
- (4)  $\Omega(n) = O(n^2)$ .
- (5) The best-case running time of INSERTION-SORT is  $O(n \lg n)$ .
- (6) MERGE-SORT is an asymptotically optimal comparison-based sorting algorithm.

*Solution.*

- (1) False. We present a counter-example: let  $a := 2 \in \mathbb{Z}$ ,  $b := 2 \in \mathbb{Z}$  and  $n := 4 \geq 1$ . We have  $ab \bmod n = 4 \bmod 4 = 0$ , but  $a \bmod n = 2 \bmod 4 = 2 \neq 0$  and  $b \bmod n = 2 \bmod 4 = 2 \neq 0$ .
- (2) False. We present a counter-example: let  $n := 2 \geq 2$ . We have  $F_{2n} = F_4 = 3$ , but  $F_{2(n-1)} + F_{2(n-2)} = F_2 + F_0 = 1 + 0 = 1 \neq 3$ .
- (3) False. We present a counter-example: let  $g(n) := 2n$ ,  $f(n) := n = \Theta(g(n))$ . Both  $f(n)$  and  $g(n)$  are asymptotically positive and polynomially bounded functions. We show  $f(n) = o(g(n))$ :

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^{2n}}} = \lim_{n \rightarrow \infty} \frac{2^{2^n}}{2^{2^n \cdot 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n \cdot 2^n - 2^n}} = \lim_{n \rightarrow \infty} \frac{1}{2^{2^n(2^n - 1)}} = 0$$

We therefore have  $f(n) = o(g(n))$ , and therefore,  $f(n) \neq \Theta(g(n))$ .

- (4) False. We present a counter-example: let  $f(n) = n^3$ . We therefore clearly have  $f(n) = \Omega(n)$ , but  $f(n) \neq O(n^2)$ .
- (5) True. The best-case running time of INSERTION-SORT is  $\Theta(n)$ , which in turn is  $O(n \lg n)$ .
- (6) True. From Theorem 3.3.1 (proven in class), we know any comparison-based sorting algorithm performs at  $\Omega(n \lg n)$ . Since the running time of MERGE-SORT is  $\Theta(n \lg n)$ , there does not exist any comparison-based sorting algorithm that is asymptotically better than MERGE-SORT.

6.6 2 / 2

- ✓ - 0 pts Correct
- 2 pts Incorrect
- 1 pts No answer