**MATH182 MIDTERM SOLUTIONS**
**DUE July 7, 2020**

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

```
MAX-LEFT-ARRAY(A)
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*

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

(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 will prove the following claim:

**Claim.** *After line 7 finishes, the value of max is*

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

*Proof of claim.* After lines 1 and 2 run, we have $\mathit{max} = \mathit{sum} = A[1]$. We will now prove the following is true about line 3:

> (Loop Invariant) After each time line 3 runs, the value of $\mathit{sum}$ is $\mathit{sum} = \sum_{i=1}^{j-1} A[i]$. Furthermore, the value of $\mathit{max}$ is
>
> $$\mathit{max} \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j-1\right\}$$

(Initialization) Suppose line 3 has just run for the first time. Then $j = 2$, $\mathit{sum} = A[1] = \sum_{i=1}^{j-1} A[i]$, and

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

(Maintenance) Suppose line 3 has just finished running and the current value of $j$ is $j = j_0$, where $2 \leq j_0 \leq A.length$. Furthermore, suppose we know that the loop invariant is true at this point. This means that $sum = \sum_{i=1}^{j_0-1} A[i]$ and

$$max \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0 - 1\right\}$$

Next we perform line 4, so the new value of $sum$ is $sum = \sum_{i=1}^{j_0-1} + A[j_0] = \sum_{i=1}^{j_0} A[i]$. Then in line 5 we check if $sum > max$. This gives us two cases:

(Case 1) Suppose $sum > max$. Thus

$$\sum_{i=1}^{j_0} A[i] \;>\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0 - 1\right\}.$$

In particular,

$$\sum_{i=1}^{j_0} A[i] \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0\right\}.$$

Then we go into line 6 and set $max = sum = \sum_{i=1}^{j_0} A[i]$. Thus

$$max \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0\right\}$$

Finally, we go back to line 3 and increase $j$, so the current value of $j$ is $j = j_0 + 1$. In particular, $sum = \sum_{i=1}^{j_0} A[i] = \sum_{i=1}^{j-1} A[i]$. Also,

$$max \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0\right\} \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j - 1\right\}$$

Thus the loop invariant remains true in this case.

(Case 2) Suppose $sum \leq max$. Thus

$$\sum_{i=1}^{j_0} A[i] \;\leq\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0 - 1\right\}.$$

In particular,

$$max \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0 - 1\right\} \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0\right\}$$

Next we go directly back to line 3 and increase $j$. The new value of $j$ is $j = j_0 + 1$. In particular, $sum = \sum_{i=1}^{j_0} A[i] = \sum_{i=1}^{j-1} A[i]$. Also,

$$max \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j_0\right\} \;=\; \max\left\{\sum_{i=1}^{k} A[i] : 1 \leq k \leq j - 1\right\}$$

Thus the loop invariant remains true in this case.

(Termination) Now that we know that the loop invariant is true, let's see what it says the final time line 3 is run. In this case, $j = A.length + 1$, and the loop invariant implies

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

Then in line 7 we return this value of $max$. Since this is what we wanted to show the algorithm does, we conclude that the algorithm is correct. $\square$

(2) We will analyze the running time of this algorithm, where $n := A.length$:

MAX-LEFT-ARRAY(A)

| | | |
|---|---|---|
| 1 | $sum = A[1]$ | $cost : c_1 \;\; times : 1$ |
| 2 | $max = sum$ | $cost : c_2 \;\; times : 1$ |
| 3 | **for** $j = 2$ **to** $A.length$ | $cost : c_3 \;\; times : n$ |
| 4 | $\quad sum = sum + A[j]$ | $cost : c_4 \;\; times : n - 1$ |
| 5 | $\quad$ **if** $sum > max$ | $cost : c_5 \;\; times : n - 1$ |
| 6 | $\qquad max = sum$ | $cost : c_6 \;\; times : \sum_{j=2}^{n} \delta_j$ |
| 7 | **return** $max$ | $cost : c_7 \;\; times : 1$ |

where for each $j = 2, \ldots, n$,

$$\delta_j \;:=\; \begin{cases} 1 & \text{if } sum > max \text{ for this value of } j \\ 0 & \text{otherwise} \end{cases}$$

Summing up all the costs, we find that the running time is:

$$\begin{aligned} T(n) \;&=\; c_1 + c_2 + c_3 n + c_4(n-1) + c_5(n-1) + c_6 \sum_{j=2}^{n} \delta_j + c_7 \\ &=\; (c_3 + c_4 + c_5)n + c_6 \sum_{j=2}^{n} \delta_j + (c_1 + c_2 - c_4 - c_5 + c_7) \end{aligned}$$

In the best case, we have that all $\delta_j = 0$, in which case the running time is

$$T(n) \;=\; (c_3 + c_4 + c_5)n + (c_1 + c_2 - c_4 - c_5 + c_7) \;=\; an + b$$

for appropriate constants $a$ and $b$. In the worst case, we have all $\delta_j = 1$, in which case the running time is

$$\begin{aligned} T(n) \;&=\; (c_3 + c_4 + c_5)n + c_6(n-1) + (c_1 + c_2 - c_4 - c_5 + c_7) \\ &=\; (c_3 + c_4 + c_5 + c_6)n + (c_1 + c_2 - c_4 - c_5 - c_6 + c_7) \;=\; an + b \end{aligned}$$

for appropriate constants $a$ and $b$. Thus our overall running time is $O(n)$ and $\Omega(n)$, hence $\Theta(n)$.

(3) This algorithm is asymptotically optimal. Every algorithm which compute this max-subarray-from-the-left must read all $n$ entries of the array $A$, so it must be at least $\Omega(n)$. $\square$

**Question 2.** *Recall that for $0 \le k \le n$, the* ***binomial coefficent*** $\binom{n}{k}$ *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* <u>*recursive*</u> *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) See `https://en.wikipedia.org/wiki/Pascal%27s_rule` for various proofs.

(2) The following pseudocode works:

```
BINOMIAL(n, k)
1  if n == k or k == 0
2      return 1
3  else
4      return BINOMIAL(n − 1, k) + BINOMIAL(n − 1, k − 1)
```

(3) We will prove the following claim:

**Claim.** *For every $n \geq 0$ and every $0 \leq k \leq n$,* BINOMIAL$(n,k)$ *returns $\binom{n}{k}$.*

*Proof of claim.* Since BINOMIAL is a recursive algorithm, we will prove this by induction on $n \geq 0$:

$$P(n): \quad \text{"For every } 0 \leq k \leq n, \text{ BINOMIAL}(n,k) \text{ returns } \binom{n}{k}\text{"}$$

(Base Case) Suppose $n = 0$. Then $k = 0$. In line 1 the condition "$n == k$ or $k == 0$" is true, so in line 2 we return 1. However, $\binom{n}{k} = \binom{0}{0} = 1$, so the base case is proved.

(Inductive step) Suppose for some $n \geq 1$ we know that $P(n-1)$ is true. We will prove $P(n)$. Let $0 \leq k \leq n$. We have two cases:

(Case 1) Suppose $k = 0$ or $k = n$. In this case, in line 1 the condition "$n == k$ or $k == 0$" is true, so in line 2 we return 1. However, $\binom{n}{0} = \binom{n}{n} = 1$, so we have returned the correct number.

(Case 2) Suppose $0 < k < n$. Then the condition in line 1 is false, so we proceed to lines 3 and 4. In line 4 we compute BINOMIAL$(n-1,k)$ + BINOMIAL$(n-1,k-1)$. Since we know $P(n-1)$ is true, this means we compute $\binom{n-1}{k} + \binom{n-1}{k-1}$. By part (1), we have computed $\binom{n}{k}$, so we return $\binom{n}{k}$.

This concludes the proof of the inductive step. By the Principle of Induction, we conclude that $P(n)$ is true for all $n \geq 0$. Thus our claim is proved. □

□

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

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

*Solution.* The recursion tree in Figure 1 shows the following:

(1) There are $n-1$ levels of the tree.
(2) For each level $k$, $0 \leq k < n-1$, the total work done is $2^k$.
(3) For level $n-1$, the total work done is $T(1)2^{n-1}$.
(4) Thus the total work done is

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

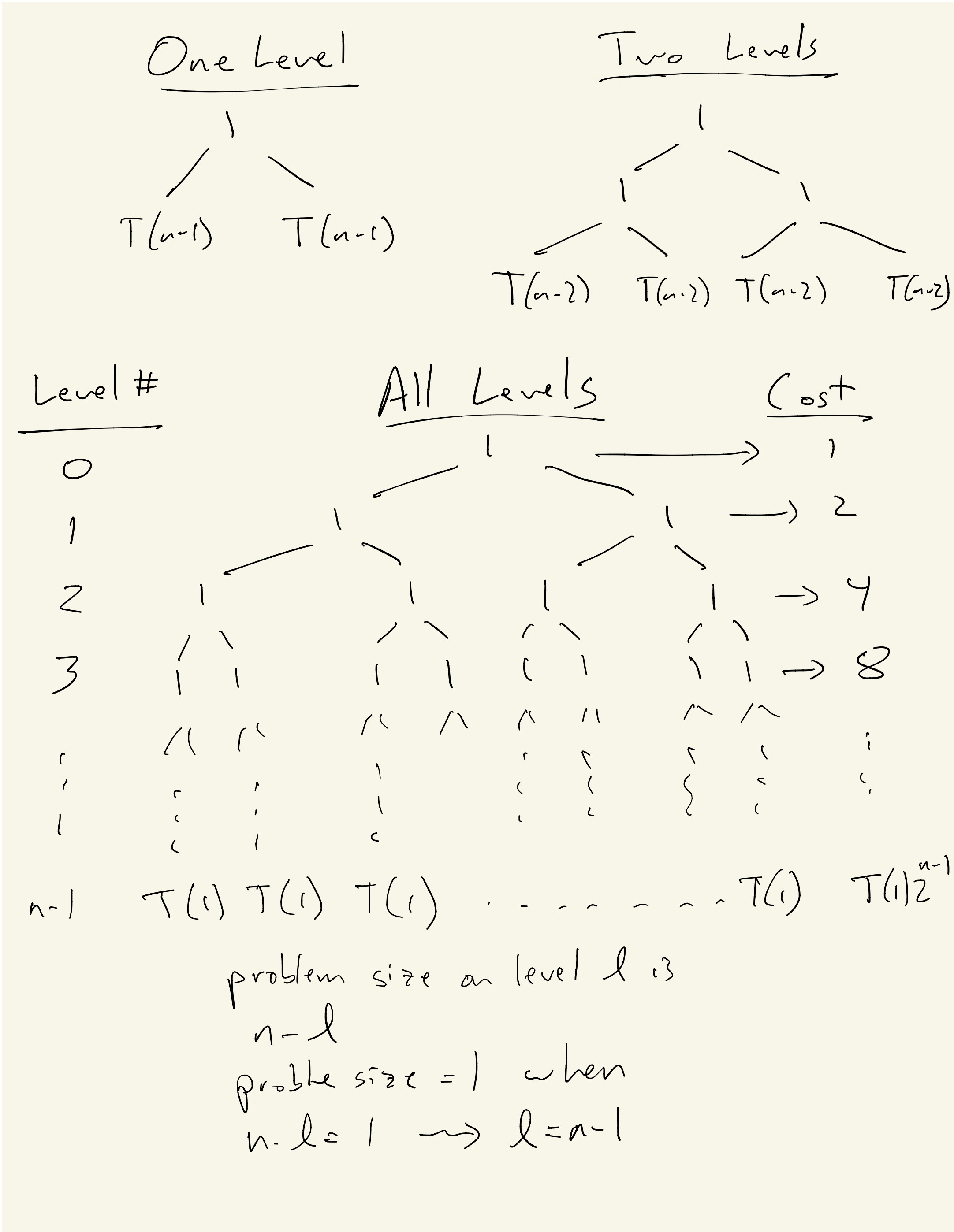


FIGURE 1. The recursion tree for $T(n) = 2T(n-1) + 1$

Using the substitution method, we will first show that $T(n)$ is $O(2^n)$. Assuming inductively there is some $c > 0$ such that $T(m) \leq c2^m$ for $m < n$. Note that

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &\leq 2c2^{n-1}+1 \\ &= c2^n+1 \end{aligned}$$

It seems there is no value of $c > 0$ which will make this inductive proof work. As our next attempt, assume inductively there are values $c, d > 0$ such that $T(m) \leq c2^m - d$ for $m < n$. Note that

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &\leq 2c2^{n-1}-2d+1 \\ &= c2^n-(2d-1) \end{aligned}$$

Now we want this last quantity to be $\leq 2c^n - d$, so we require $-(2d-1) \leq -d$, i.e., $1 \leq d$. Thus for any value of $d \geq 1$ this inductive proof will work. Thus $T(n) = O(2^n)$.

Next we will show that $T(n)$ is $\Omega(2^n)$. Assume inductively there is some $c > 0$ such that $T(m) \geq c2^m$ for $m < n$. Note that

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &\geq 2c2^{n-1}+1 \\ &= c2^n+1 \\ &\geq c2^n. \end{aligned}$$

Thus we see that this inductive proof works for any value of $c > 0$, hence $T(n) = \Omega(2^n)$. $\square$

**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.* In this case we have $a = 25$, $b = 5$, $n^{\log_b a} = n^{\log_5 25} = n^2$. Furthermore, $f(n) = \Theta(n^2 \lg^{-1} n)$, so $f(n) = \Theta(n^{\log_b a} \lg^k n)$, where $k = -1$. Thus we are in Case (2)(b) of the Master Theorem, so we conclude

$$T(n) = \Theta(n^{\log_b a} \lg\lg n) = \Theta(n^2 \lg\lg n).$$ $\square$

**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:*

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

*Justify your answer. (5pts)*

*Solution.* We claim that $f(n) = o(g(n))$. To show this, it suffices to show that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

To compute the limit of $f(n)/g(n)$, we can instead compute the limit of $f(h(n))/g(h(n))$ where $h(n)$ is any function such that $\lim_{n\to\infty} h(n) = +\infty$. Consider $h(n) := 2^n$. Then the limit we have to compute is

$$\lim_{n\to\infty} \frac{n^{\sqrt{n}}}{\sqrt{2^n}} \;=\; \lim_{n\to\infty} \frac{2^{\sqrt{n}\lg n}}{2^{n/2}} \;=\; \lim_{n\to\infty} 2^{\sqrt{n}\lg n - n/2}$$

Now we will look at the limit of the exponent:

$$\lim_{n\to\infty} \sqrt{n}\lg n - n/2 \;=\; \lim_{n\to\infty} \sqrt{n}(\lg n)(1 - \sqrt{n}/2\lg n)$$

Now notice that

$$\lim_{n\to\infty} \frac{\sqrt{n}}{2\lg n} \;=\; \lim_{n\to\infty} \frac{1/2n^{-1/2}}{2/n} \;=\; \lim_{n\to\infty} \frac{\sqrt{n}}{4} \;=\; +\infty$$

(by L'Hopital's rule), and thus

$$\lim_{n\to\infty} (1 - \sqrt{n}/2\lg n) \;=\; -\infty.$$

Since $\lim_{n\to\infty} \sqrt{n}\lg n = +\infty$, we have

$$\lim_{n\to\infty} \sqrt{n}\lg n - n/2 \;=\; \lim_{n\to\infty} \sqrt{n}\lg n(1 - \sqrt{n}/2\lg n) = -\infty.$$

Thus

$$\lim_{n\to\infty} \frac{n^{\sqrt{n}}}{\sqrt{2^n}} \;=\; \lim_{n\to\infty} 2^{\sqrt{n}\lg n - n/2} \;=\; 0,$$

from which it follows

$$\lim_{n\to\infty} \frac{(\lg n)^{\sqrt{\lg n}}}{\sqrt{n}} \;=\; 0,$$

which is what we wanted to show. □

**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. Let $a = 2, b = 3, n = 6$. Then $ab \bmod n = 6 \bmod 6 = 0$, whereas $a \bmod b = 2 \bmod 6 = 2$ and $b \bmod n = 3 \bmod 6 = 3$.

(2) False. Note that $F_0 = 0, F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3$, so for $n = 2$ we have $F_4 = 3$, but $F_2 + F_0 = 1 + 0 = 1$.

(3) False. For instance, $f(n) = \lg n$ and $g(n) = \lg(n/2)$ are both asymptotically positive, polynomially bounded, with $f(n) = \Theta(g(n))$. However,

$$2^{2^{f(n)}} \;=\; 2^n$$

and

$$2^{2^{g(n)}} \;=\; 2^{n/2}$$

and

$$\lim_{n\to\infty} \frac{2^{n/2}}{2^n} \;=\; \lim_{n\to\infty} \frac{1}{2^{n/2}} \;=\; 0,$$

which shows $2^{2^{f(n)}} = \omega(2^{2^{g(n)}})$.

(4) False. For example, the function $f(n) = n^3$ is $\Omega(n)$, but it is not $O(n^2)$.

(5) True. The best-case running time of INSERTION-SORT is $O(n)$, and $O(n) = O(n \lg n)$.

(6) True. MERGE-SORT has running time $\Theta(n \lg n)$, and we showed that the worst-case running time of any comparison-based sorting algorithm is always $\Omega(n \lg n)$. $\square$