Midterm 2

CSC 236 H1Y (LEC5101)

17 July 2024

Question 1. [10 marks] True/false and short answer

True/false. For true/false, circle one of True or False to select it (you may leave it blank for no marks). Each question in this subsection will be worth 1 mark for a correct answer and -1 *mark* for an incorrect answer. You will get no more than five and no fewer than zero marks. *Do not guess.*

Part (a) [1 mark]

Suppose the time to perform a single digit addition or multiplication is $O(1)$. All algorithms which perform multiplication on two n digit numbers run in time $\Omega(n^2)$.

True / **False**

False. Karatsuba's algorithm is an example of performing the multiplication faster than $O(n^2)$.

Part (b) [1 mark]

In the quicksort algorithm that we discussed in-class, the time it takes to partition a list of length n around a pivot is $\Theta(n)$.

True / **False**

True. Partition is a linear time algorithm on an input of length n.

Part (c) [1 mark]

Suppose the running time of an algorithm on an input of size n is $\Theta\left(n^2\right)$. If I triple the size of the input then the algorithm will take approximately six times as long to run.

True / **False**

False. It will take nine times as long to run.

Part (d) [1 mark]

$2^{n+1} \in O(2^n)$. **True** / **False**

True. $2^{n+1} = 2 \cdot 2^n$.

Part (e) [1 mark]

$2^{2n} \in O(2^n)$. **True** / **False**

False. $2^{2n} = (2^n)^2$. That would be like saying $n^2 \in O(n)$.

Short Answer. Solve the following recurrence relations. Put the answer in closed-form, i.e., $T(n) = \Theta(g(n))$ for $g(n)$ that you determine. No justification required.

For reference, the Master Method states: For constants $a \geq 1$ and $b > 1$, function $f(n)$, and recursive function $T(n)$ defined on the nonnegative integers by

$$T(n) = \begin{cases} c & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

for constant c, then $T(n)$ has the following asymptotic bounds.

- If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
- If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{log_b a} \log n\right)$.
- If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(f(n))$.

Part (f) [1 mark]

$T(n) = T\left(\frac{n}{4}\right) + n$.

$T(n) = \Theta(n)$. The third case applies with $a = 1$, $b = 4$ and $f(n) = n = \Omega\left(n^{\log_4 1}\right)$.

Part (g) [1 mark]

$T(n) = 2T(n/4) + 1$.

$T(n) = \Theta(\sqrt{n})$. The first case applies with $a = 2$, $b = 4$ and $f(n) = 1 = O\left(n^{\log_4 2}\right) = O\left(n^{1/2}\right)$.

Part (h) [1 mark]

$T(n) = 2T(n/4) + \sqrt{n}$

$T(n) = \Theta(\sqrt{n} \log n)$. The second case applies with $a = 2$, $b = 4$ and $f(n) = n^{1/2} = O\left(n^{\log_4 2}\right) = O\left(n^{1/2}\right)$.

Midterm 2

CSC 236 H1Y (LEC5101)

17 July 2024

Part (i) [2 marks]

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$.

Effectively $T(n) = 2T(\lfloor n/2 \rfloor) + n$ which has closed form $T(n) = \Theta(n \log n)$.

Question 2. [6 marks] Iterative Algorithm Analysis

Consider Algorithm 1. Ranges are left-inclusive and right-exclusive, e.g., for $i = 0..5$, i takes values $0, 1, 2, 3, 4$.

Algorithm 1 f(A : list)

```
1: for i = 0..(|A| − 1) do
2:     if A[i] > A[i + 1] then
3:         for j = i..(|A| − 1) do
4:             if A[j] < A[j + 1] then return False
5:             end if
6:         end for
7:     end if
8: end for
9: return True
```

Part (a) [2 marks]

State the post-condition of Algorithm 1.

Use *at most* two sentences (your answer should demonstrate high-level understanding of the algorithm).

Solution. The function f determines whether or not A is weakly uni-modal (i.e., there exists some index i such that the array is increasing or equal from 0 to i and decreasing or equal from $i + 1$ to the end of A).

Part (b) [1 mark]

State an input A of length n for which f(A) runs in $O(1)$ time.

Solution. $A = [a_0, a_1, a_2, ..., a_{n-1}]$ where $a_0 = 1$, $a_1 = 0$, $a_2 = 1$, and $a_k = 0$ for $k \geq 2$.

Part (c) [3 marks]

State and justify the asymptotically tight worst-case running time of Algorithm 1.

To show $T(n) = \Theta(f(n))$, you must show (1) $T(n) = O(f(n))$ by analyzing the algorithm and (2) $T(n) = \Omega(f(n))$ by giving an instance A of length n for which f(A) *runs in time $\Omega(f(n))$.*

Solution. We claim that $T(n) = \Theta\left(n^2\right)$. For the upper bound, we note that there are two nested for-loops on lines 1 and 3 respectively and each for-loop runs for at most n iterations (where n is the length of A). For the lower bound. Consider the input $A = [n, n-1, ..., 1]$. Note that the if-statement on line 2 will be true for all i and so the for-loop on line 3 will run n times. For the first $n/2$ iterations, the for-loop on line 3 will iterate at least $\lfloor n/2 \rfloor$ times so f(A) will take at least $\left\lfloor n^2/4 \right\rfloor = \Omega(n^2)$ time in total.

Midterm 2

CSC 236 H1Y (LEC5101)

17 July 2024

Question 3. [6 marks] Recursion

Compute a tight *upper bound* for the closed-form of the following recurrence relation:

$$T(n) = \begin{cases} c & n \leq 10 \\ T(\lfloor n/2 \rfloor + 1) + n & n > 10 \end{cases}$$

where c is a constant. Formally prove your closed-form expression is correct.

You may use the Master Method (if it is applicable), but you must show your work.

Solution. $T(n) = O(n)$. Note that the Master Method does not apply here so we will prove the closed form by induction. Let c' be a fixed constant that we will define later. Let $P(n)$ be the statement that $T(n) \leq c'n$. We want to show that $P(n)$ is true for all $n \in \mathbb{N}$.

In the base case, for $a \in \{0, \ldots, 10\}$, $T(a)$ is a constant.

In the inductive step, for $k > 10$, suppose that $P(0) \wedge \cdots \wedge P(k-1)$ is true. Show that $P(k)$ is true. Note that

$$\begin{aligned} T(k) &= T(\lfloor k/2 \rfloor + 1) + k \\ &\leq c'(\lfloor k/2 \rfloor + 1) + k && \text{(IH)} \\ &\leq c'k + c' - \frac{c'k}{2} + k \\ &\leq c'k \end{aligned}$$

where $c' - \frac{c'k}{2} + k \leq 0$ since $k \geq 10$ and for all $c' \geq 10$. Choose $c' = \max(c, 10)$.

By the principle of mathematical induction $T(n)$ is true for all n.

Question 4. [12 marks] First Non-positive

You are given as input a *zero-indexed* list A of length n sorted in non-increasing order, i.e., $A[0] \geq A[1] \geq \cdots \geq A[n-1]$. **Write an *iterative* algorithm which outputs the index of the first non-positive entry in A or n if all entries in A are positive.**

For example, on input $A_1 = [1, 0, -1, -2, -3]$, your algorithm should output 1. On input $A_2 = [100, 98, 7, 5, 3, 1]$, your algorithm should output 6.

Your algorithm should run in $O(\log n)$ time. Correct algorithms which run in $O(n)$ time will get part marks.

Part (a) [4 marks]

Write the pseudo-code of your algorithm here.

Solution. This is just binary search for zero. See Algorithm 2.

Algorithm 2 g(A : list)

```
1:  n = |A|
2:  i = 0
3:  j = n
4:  while i < j do
5:      m = (j + i)/2                          ▷ integer division
6:      if A[m] ≤ 0 then
7:          j = m
8:      else
9:          i = m + 1
10:     end if
11: end while
12:
13: if i < n and A[i] ≤ 0 then return i
14: else return n
15: end if
```

Part (b) [2 marks]

State the variables and loop-invariant you will use in your proof-of-correctness.

Solution. Let A_t be the interval specified by the indices i and j after the t^{th} iteration of the while-loop. Let m_t be the value of the variable m after the t^{th} iteration of the while-loop. Note that m_0 is undefined.

Our loop-invariant will be the predicate $P(t)$ which states: if $0 \leq t \leq \lceil \log n \rceil$ and if there exists an index in A which is non-positive, then this index will be in A_t.

We show that $P(t)$ is true for all $t \in \mathbb{N}$.

Part (c) [4 marks]

Prove the correctness of your algorithm.

Solution. Our proof is by induction on t. In the base case $t = 0$ (i.e., before the while-loop on line 4). Initially, $A_0 = A$ so if A contains a non-positive element then, A_0 will certainly contain it as well.

Suppose for the inductive step we fix some $t \in \{0, \ldots, \lceil \log n \rceil - 1\}$ and have that $P(0) \wedge \cdots \wedge P(t)$ is true. We will show that $P(t+1)$ is true. If A contains a non-positive element, then by the inductive hypothesis, A_t will contain this entry. m_{t+1} is the mid-point of the interval A_t and we have that $m_{t+1} \in \{i, \ldots, j-1\}$. If $A[m] \leq 0$, then we know that the *first* index which is non-positive is at some index $i, \ldots, m$ since A is non-increasing. Thus if we update $j = m$, A_{t+1} will still contain the smallest non-positive index. Conversely, if $A[m] > 0$ then the *first* index which is non-positive is at some index $m+1, \ldots, j$ and our update, $i = m+1$ on line 9, will ensure that it falls within A_{t+1}.

By the principle of Mathematical Induction, $P(t)$ is true for all $t \in \mathbb{N}$.

$i \leq n$ and if A does not contain any non-positive elements, then $A[i] > 0$ for all $i \in \{0, \ldots, n-1\}$ and so the algorithm will return n as required.

Part (d) [2 marks]

State and justify the asymptotic running time of your algorithm.

Solution. Binary search runs in $\Theta(\log n)$. At every iteration of the while-loop we half the size of A_t by updating one of i or j to be m_t on line 7 or line 9.

Question 5. [12 marks] Power of Three

A natural number n is a *power of three* if there exists a natural number x such that $n = 3^x$.

Write a *recursive algorithm* which takes as input a natural number n and returns true if n is a power of three and false otherwise. Assume that simple arithmetic operations (addition, subtraction, multiplication, division, and modulus) between n and a constant takes $O(1)$ time. You *do not* have access to complex arithmetic operations such as logarithms, exponentiation, etc.

For example, your algorithm should return *true* on input 27 since $27 = 3^3$ and return *false* on input 0 as there does not exist a natural number x such that $3^x = 0$.

Make your algorithm as fast as possible, but you do not have to justify that it is the fastest.

Part (a) [4 marks]

Write the pseudo-code of your algorithm here.

Solution. See Algorithm 3. The algorithm takes as input a natural number n and outputs true or false depending on whether or not n is a power of three.

Algorithm 3 h($n : \mathbb{N}$)

```
1: if n = 1 then return True
2: else if n = 0 or (n mod 3) ≠ 0 then return False
3: end if
4: return h(n/3)                                   ▷ integer division
```

Part (b) [5 marks]

Prove the correctness of your algorithm.

Solution. It's possible to prove correctness using standard induction on n or structural induction. We will do the latter.

The base case is covered by the first three lines of the algorithm. If $n = 1$ then it is a power of three as $1 = 3^0$. Conversely, if $n = 0$ or n is not a multiple of three, then n is not a power of three.

In the inductive step. Suppose that the recursive call return the correct result on $n/3$ by the inductive hypothesis. We will show that it returns the correct result on n. Since line 2 already took care of the case where n is not a multiple of three, it must be the case that n is a multiple of three. If h($n/3$) returns true, then $n/3 = 3^k$ for some $k \in \mathbb{N}$. It follows that $n = 3^{k+1}$ so the algorithm correctly returns true as well. If h($n/3$) returns false, then $n/3$ is equal to zero or is *not* a multiple of three. It follows that n is not a power of three and the algorithm again returns the correct result.

Part (c) [3 marks]

State and justify the asymptotic running time of your algorithm.

Solution. Let $T(n)$ be the running time of the algorithm. Note that we can define the following recurrence relation for constants c_1 and c_2:

$$T(n) = \begin{cases} c_1 & n = 1 \\ T(n/3) + c_2 & n > 1 \end{cases}.$$

To see this, note that we have one recursive call on an input one third the size on line 4. Further there is some $O(1)$ work done outside the recursive call in the first three lines as well as to compute $n/3$ in line 4. By the Master Method, we know that the asymptotic running time of this recurrence relation is $T(n) = \Theta(\log n)$.

Bonus. [2 marks] More Complex Recurrence Relation

Give the asymptotically tight closed-form expression for the recurrence relation:

$$T(n) = \begin{cases} c & n = 1 \\ 3T(\sqrt{n}) + \log n & n > 1 \end{cases}$$

where c is a constant. Show your work.

You should only try this problem if you have solved the five previous problems and checked that your solution for them is correct. That would be a better use of your time.

$T(n) = \Theta\left((\log n)^{\log_2 3}\right)$ using the substitution $2^m = n$ and applying the master method.