CS271: Data Structures

Name: Linh Mai, Bao Luu, Jeremiah Shelton +17 proof points

Instructor: Dr. Stacey Truex

Project #1

1. Prove Theorem 3.1 on page 48: +3/3

   For any two functions $f(n)$ and $g(n)$, $f(n) = \Theta(g(n))$
   if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

   Want to show that:

   - $f(n) = \Theta(g(n)) \implies f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. **(1)**
   - $f(n) = O(g(n))$ and $f(n) = \Omega(g(n)) \implies f(n) = \Theta(g(n))$. **(2)**

   (1). $f(n) = \Theta(g(n)) \implies f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

   Since $f(n) = \Theta(g(n))$, by definition, there exists positive constants c_1, c_2, n_0 so that:

   $$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \ \forall n \geq n_0$$

   Since $0 \leq$ $c_1 g(n) \leq f(n) \ \forall n \geq n_0$ with c_1, n_0 being positive constants, that satisfies the definition of the Ω notation. Hence, $f(n) = \Omega(g(n))$. Similarly, for the inequality $0 \leq$ $f(n) \leq c_2 g(n) \ \forall n \geq n_0$, with c_2, n_0 being positive constants, we can also conclude that $f(n) = O(g(n))$.

   (2). $f(n) = O(g(n))$ and $f(n) = \Omega(g(n)) \implies f(n) = \Theta(g(n))$.

   Since $f(n) = O(g(n))$, there exists positive constants c_3, n_1 so that:

   $$0 \leq f(n) \leq c_3 g(n) \ \forall n \geq n_1$$

   Similarly, since $f(n) = \Omega(g(n))$, there exists positive constants c_4, n_2 so that:

   $$0 \leq c_4 g(n) \leq f(n) \ \forall n \geq n_2$$

   Without loss of generality, let $n_1 > n_2$. Then c_3, c_4, n_1 are positive constants so that:

   Fair enough use of WOLOG, easier might have been $n_0 = \max(n_1, n_2)$

$$0 \le c_4 g(n) \le f(n) \le c_3 g(n)\ \forall n \ge n_1$$

Hence, $f(n) = \Theta(g(n))$.

2. Prove the following using the definitions of O, Ω, and Θ. +5.5/9

+1 (a) $17n^2 + 8n - 25 = O(n^2)$
Want to show that $\exists$ positive constants c, n_0 such that:

$$0 \le 17n^2 + 8n - 25 \le cn^2 \quad \forall n \ge n_0$$

REFORMAT:
$17n^2 + 8n - 25 < 17n^2 + 8n$
$\le 17n^2 + 8n^2$ when $n \ge 1$
$= 25n^2$
$\le cn^2$ when $c \ge 25$

1. $17n^2 + 8n - 25 \ge 0$

$(17n + 25)(n - 1) \ge 0$

when $n \ge -\frac{25}{17}$ **or** $n \ge 1$ X

n=0 satisfies first condition but violates inequality, stick w/ second constraint only

2. $17n^2 + 8n - 25 \le cn^2$

$\le 17n^2 + 8n \le 17n^2 + 8n^2$ when?

$\Leftrightarrow 17n^2 + 8n - 25 \le 25n^2 \le cn^2$ when?

alignment is confusing

when $n^2 \ge n$ and $c \ge 25$ put constraint on corresponding line

For constants $c = 26$, $n = 1$, $0 \le 17n^2 + 8n - 25 \le cn^2$ Therefore, ~~$n^2 + 3n - 20$~~ $17n^2 + 8n - 25$ $=$ $O(n^2)$.

my bad :)

+0.5 (b) $\frac{1}{2}n - 15 = \Omega(n)$
Want to show that $\exists$ positive constants c, n_0 such that:

$$0 \le cn \le \frac{1}{2}n - 15\ \ \forall n \ge n_0$$

1. $cn \ge 0$ when $c \ge 0$ and $n \ge 0$ ✓

2. Suppose $\frac{1}{2}n - 15 \ge cn$

$$cn - \frac{1}{2}n \le -15$$

$$n(c - \frac{1}{2}) \le -15$$

since n is positive

$$\Rightarrow c - \frac{1}{2} < 0 \Leftrightarrow 0 < c < \frac{1}{2}$$

? how do these steps relate?

$$\Rightarrow n(c - \frac{1}{2}) \le -15 \text{ when } n > 30$$

consider n=31, c=0.49 violates inequality

For constants $c = \frac{1}{3}$, $n = 32$, $0 \le cn \le \frac{1}{2}n - 15$. Therefore, $\frac{1}{2}n - 15 = \Omega(n)$.

+0.5

(c) $\log_{16} n + 1 = \Theta(\log_2 n)$

Want to show that $\exists$ positive constants c_1, c_2, n_0, such that:

$$0 \leq c_1 \log_2 n \leq \log_{16} n + 1 \leq c_2 \log_2 n \;\; \forall n \geq n_0$$

1. $0 \leq c_1 \log_2 n$ when $c_1 \geq 0, n \geq \cancel{0}$ 1

2. $c_1 \log_2 n \leq \log_{16} n + 1$

$\Leftrightarrow c_1 \log_2 n \leq \frac{1}{4} \log_2 n + log_2 2$

$\Leftrightarrow c_1 \log_2 n \leq$?

3. $\log_{16} n + 1 \leq \frac{1}{4}.\log_2 n + 1$

$\leq c_2 \log_2 n$ when $c_2 \geq 2$ too big a jump in logic for a proof

For constants $c_1 = 13/4$, $c_2 = 4$. Therefore, $\log_{10} n + 4 = \Theta(\log_2 n)$.

+0.5

(d) $3^{n+4} = O(3^n)$

Want to show that $\exists$ positive constants c_1, n_0 such that:
$0 \leq 3^{n+4} \leq c_1 3^n \quad \forall n \geq n_0$
1.) $0 \leq 3^{n+4}$ for all n ✓
2.) $3^{n+4} \leq c_1 3^n$
$3^n \leq c_1.3^n$ when $c_1 \geq 0$ WTS $3^{n+4} \leq c_1 3^n$ not that $3^n \leq c_1 3^n$
For constants $c_1 = 5$, $n_0 = 2$, $0 \leq 3^{n+4} \leq c_1 3^n$. Therefore, $2^{n+1} = O(2^n)$.

+1.5

(e) $\ln n = \Theta(\log_2 n)$

Want to show that there exists positive constants c_1, c_2, n_0 so that:

$$0 \leq c_1 \log_2 n \leq \ln n \leq c_2 \log_2 n \;\forall n \geq n_0$$

REFORMAT

$\ln n = \frac{\log_2 n}{\log_2 e}$

$\geq c_1 \log_2 n$ when $c_1 \leq \frac{1}{\log_2 e}$

$$\ln n = \frac{\log_2 n}{\log_2 e}$$

$$\Rightarrow \ln n \geq c_1 \log_2 n$$

$$\Leftrightarrow \frac{1}{log_2 e} log_2 n \geq c_1 \log_2 n$$

$$\Leftrightarrow \frac{1}{log_2 e} \geq c_1$$

Therefore for constants $c_1 = 0.5, n_0 = 1$, $0 \leq c_1 \log_2 n \leq \ln n \; \forall n \geq n_0$. We also ~~show~~: note

$$\ln n \leq \log_2 n \; \forall n \geq 1$$

For constants $c_1 = 0.5, c_2 = 1, n_0 = 1$, $0 \leq c_1 \log_2 n \leq \ln n \leq c_2 \log_2 n \; \forall n \geq n_0$. Therefore, $\ln n = \Theta(\log_2 n)$.

+1.5

(f) $n^\epsilon + 3 = \Omega(\log_2 n)$ for any $\epsilon > 0$

Want to show there exists positive constants c, n_0 so that:

$$0 \leq c \log_2 n \leq n^\epsilon + 3 \; \forall n \geq n_0$$

$$0 \leq c log_2 n \text{ when } n \geq 1$$ and $c \geq 0$

Lemma 1 ($2^x > x \; \forall x \geq 0$).

Proof. We will prove this lemma using induction.

- Base case:

$$x = 0 : 2^0 = 1 > 0$$
$$x = 1 : 2^1 = 2 > 1$$
$$x = 2 : 2^2 = 4 > 2$$

- Inductive hypothesis: Let $k > 2$ be a number where $2^k > k$.
- Inductive step: We will show that $2^{k+1} > k + 1$.

$$\begin{aligned} 2^{k+1} &= 2(2^k) \\ &> 2k \\ &> k + 2 \\ &> k + 1 \end{aligned}$$

you have no such candidate k from your base cases on which to build your induction. To see this try to write the wrap up.

□

Furthermore, if we take the logarithm of both sides of the inequality:

$$\begin{aligned} 2^x &> x \\ \log_2(2^x) &> \log_2 x \\ x &> \log_2 x \; \forall x \geq 0 \end{aligned}$$

$$\begin{aligned} n^\epsilon + 3 &> n^\epsilon \\ &> \log_2 n^\epsilon \\ &= \epsilon \log_2 n \\ &\geq c \log_2 n \Leftrightarrow c \leq \epsilon \end{aligned}$$

✓

For constants $\epsilon > 0, c = \epsilon, n_0 = 1,$ 0^{ϵ} $c \log_2 n \leq n^\epsilon + 3$. Therefore, $n^\epsilon = \Omega(\log_2 n)$ for any $\epsilon > 0$.

+5.5/9 3. For each of the following recurrences, find a tight upper bound for $T(n)$. It will be useful to remember

$$\sum_{j=0}^{\infty} a^j = \frac{1}{1-a}, \text{ if } |a| < 1.$$

Prove that each is correct using induction. In each case, assume that $T(n)$ is constant for $n \leq 2$ and that floor division applies to all recurrences.

+1 (a) $T(n) = 2T(n/2) + n^3$.

Guess: $T(n) = O(n^3)$

Inductive Hypothesis: For all positive integers $m < n$, in particular for $m = \frac{n}{2}$, it holds that $T(\frac{n}{2}) \leq c.(\frac{n}{2})^3$ for some positive constant $c > 0$
Inductive Step: $T(m) \leq cm^3$
Substituting into the recurrence yields:

$$T(n) \leq 2c(\frac{n}{2})^3 + n^3$$
$$= 2c\frac{n^3}{8} + n^3$$
$$= c\frac{n^3}{4} + n^3$$
$$= \cancel{\frac{5}{4}n^3}$$
$$T(n) \leq cn^3 \text{ when } c \ldots$$

(c is a positive constant)

Let $a > 0$ be the constant runtime when $n \leq 2$. Base Case(s):

- $T(1) = a$ $T(2) = a$

  Prove the base case:
- ~~$T(2) = 2.T(1) + 2^3 = 2a + 8 \leq c.2^3$~~
  ~~$\Rightarrow T(2) \leq 8c$ when $c \geq \frac{a}{4} + 1$~~
- $T(3) = 2.T(1) + 3^3 = 2a + 9 \leq c.3^3$
  $\Rightarrow T(3) \leq 9c$ when $c \geq \frac{2a}{9} + 1$

T(4)?
T(5)?

For constants $c = \frac{a}{4} + 2$ $n = 2$, $T(n) \leq cn^3$. Therefore, $T(n) = O(n^3)$.

+1 (b) $T(n) = T(9n/10) + n$

$$\text{Guess: } T(n) = O(n)$$

Inductive Hypothesis: For all positive integers $m < n$, in particular for $m = \frac{9n}{10}$, it holds that $T(\frac{9n}{10}) \leq c.\frac{9n}{10}$ for some positive constant $c > 0$
Inductive Step: $T(m) \leq cm$
Substituting into the recurrence yields:

$$\begin{aligned} T(n) &\leq c.\frac{9n}{10} + n \\ &\leq n.(c.\frac{9}{10} + 1) \\ &\leq cn \quad \text{when } c \ldots \end{aligned}$$

(c is some positive constant)

Let $a > 0$ be the constant runtime when $n \leq 2$. Base Case(s):

- ~~$T(1) = a$~~
  Prove the base case:
- ~~$T(2) = T$~~$(1) + 2(\text{floor division}) = a + 2 \leq c$
  $\Rightarrow T(2) \leq c$ when $c \geq a + 2$

For constants $c = a + 3$ $n = 2$, $T(n) \leq cn$. Therefore, $T(n) = O(n)$.

+1 (c) $T(n) = 3T(n/3) + bn$ for some positive constant b

$$\text{Guess: } T(n) = O(n \log n)$$

if you don't specify log base I'll assume 2

Inductive Hypothesis: what is $g(k)$?
Assume $T(k) \leq cg(k)$ for a const. c and all k ....$n_0 \leq k < n$
Inductive Step: prove $T(n) \leq cg_n$

$$\begin{aligned} T(n) &= 3T(n/3) + bn \\ &\leq 3c\frac{n}{3}\log(n/3) + bn \\ &= cn\log n - cn\log_3 3 + bn \\ &cn \log n + bn \\ &c < b \end{aligned}$$

have not completed proof w/ this additional term

Let $a > 0$ be the constant runtime when $n \leq 2$. Base Case(s):

- ~~$T(1)$~~ $= a$
  $T(3) = 3T(1) + bn = 3T(1) + 3b = 3a + 3b = 3(a + b)$
  $\leq cn \log n$
  $3c \log n$

T(4)? T(5)? T(6)? ...

$3c$ when $a + b < c$

For constants $c = a + b + 1$ $n_0 = 3$. Therefore, $T(n) = O(n \log n)$.

+0.5 (d) $T(n) = 7T(n/3) + n^2$

$$\text{Guess: } T(n) = O(n^2 \log n)$$

Inductive Hypothesis: $\forall k \leq n; T(k) \leq ck^2 \log k$
Inductive Step: prove $T(n) \leq cg(n)$

Incorrect inductive step

$$T(n) = 7T(n/3) + n^2$$
$$\boxed{T(n)} \leq cn^2 \log n \quad c(n/3)^2 \log n/3 + n^2$$
$$< 7cn \log(n/3) + n^2$$
$$< 7c \log_3 + n^2$$
$$cn^2 \log_3 n$$

Let $a > 0$ be the constant runtime when $n \leq 2$. Base Case(s):

- $T(1) = a$
  $T(3) = 3T(1) + bn = 3(a + b)$
  ? $\boxed{cn^2 \log n}$
  $\boxed{7c}$ when $a + b < 7$
  $T(3) = 7T(1) + 4b = 7(a + b)$
  $cn^2 \log n$
  when $c > a + b + 1/log_7$

additional base cases?

For constants $c = a + b + 1$ $n_0 = 7$. Therefore, $T(n) = O(n^2 \log n)$.

+1.5 (e) $T(n) = T(\sqrt{n}) + 1$ (Assume the floor is taken for $\sqrt{\cdot}$)

$$\text{Guess: } T(n) = O(\log_2(\log_2 n))$$

Inductive Hypothesis: For all positive integers $k < n$, particularly for $k = \sqrt{n}$, it holds that $T(k) \leq c \log_2(\log_2 k)$.
Inductive Step: Show that $T(n) \leq c \log_2(\log_2 n))$.

$$
\begin{aligned}
T(n) &= T(\sqrt{n}) + 1 \\
&\leq c\log_2(log_2\sqrt{n}) + 1 \\
&= c\log_2(log_2 n^{\frac{1}{2}}) + 1 \\
&= c\log_2(\frac{1}{2}log_2 n) + 1 \\
&= c\log_2(\log_2 n) - c\log_2 2 + 1 \\
&= c\log_2(\log_2 n) - c + 1 \quad \checkmark \\
&\leq c\,log_2(\log_2 n) \text{ when } c \geq 1
\end{aligned}
$$

Let $a > 0$ be the constant runtime when $n \leq 2$. Base Case(s):

- $T(3) = T(1) + 1 = a + 1 \leq c\log_2(\log_2 3)$ when $c \geq \frac{a+1}{\log_2(\log_2 3)}$
- $T(4) = T(2) + 1 = a + 1 \leq c\log_2(\log_2 4) = c$ when $c \geq a + 1$
- $T(5) = T(2) + 1 = a + 1 \leq c\log_2(\log_2 5)$ when $c \geq \frac{a+1}{\log_2(\log_2 5)}$
- $T(6) = T(2) + 1 = a + 1 \leq c\log_2(\log_2 6)$ when $c \geq \frac{a+1}{\log_2(\log_2 6)}$
- $T(7) = T(2) + 1 = a + 1 \leq c\log_2(\log_2 7)$ when $c \geq \frac{a+1}{\log_2(\log_2 7)}$
- $T(8) = T(2) + 1 = a + 1 \leq c\log_2(\log_2 8)$ when $c \geq \frac{a+1}{\log_2(\log_2 8)}$ ✓

For constants $c = 2a + 2, n_0 = 3$, $0 \leq T(n) \leq c\log_2(\log_2 n)$. Therefore, $T(n) = O(\log_2(\log_2 n))$.

+0.5 (f) $T(n) = T(n-1) + \log_2 n$

Guess: $T(n)$ $\cancel{\theta}$ $O(n \log n)$

Inductive Hypothesis: For all positive integers $m < n$, in particular for $m = n - 1$, it holds that $T(n-1) \leq c.(n-1)\log(n-1)$

Inductive Step: $T(m) \leq cm \log m$

Substituting into the recurrence yields:

$$
\begin{aligned}
T(n) &\leq c(n-1)\log(n-1) + \log_2 n \quad \text{what changed here?} \\
&\leq c(n-1)\log(n-1) + \log n \\
&\leq c(n-1)\log n + \log n \\
&\leq (c(n-1) + 1)\log n
\end{aligned}
$$

$\leq cn\log n$ since c is some positive constants ✗

Let $a > 0$ be the constant runtime when $n \leq 2$. Base Case(s):

- T(1) = a
  Proof for base case:

- $T(2) = T(1) + 1 = a + 1 \leq c.2\log 2$ when $c \geq \frac{a+1}{2\log 2}$

For constants $c = \frac{a+2}{2\log 2}$, $T(n) \leq cn\log n$ Therefore, $T(n) = n\log n$.

YES! +3/3!!

4. Using a loop invariant, prove that the following algorithm correctly sorts the array $A[1:n]$ in ascending order. You may assume that SWAP is correct.

Algorithm 1 Sort the array $A[1:n]$ in ascending order

```
procedure NESTED-SORT(A, n)
    for front = 1 to n − 1 do
        for pos = n downto front + 1 do
            if A[pos] < A[pos − 1] then
                SWAP(A, pos, pos − 1)
            end if
        end for
    end for
end procedure
```

Lemma 2 (Inner Loop Invariant). *Before each iteration pos of the inner for loop, $A[pos:n]$ contains the original elements in $A[pos:n]$ with the smallest value at index pos.*

Proof. **Initialization:** Before the first iteration, $pos = n$. So $A[pos:n]$ contains only 1 element, which is the original element in $A[n:n]$. Since there is only 1 element, it is trivially proved that $A[pos:n]$ is sorted. ✓

Maintenance: Assume that the loop invariant is true before some iteration pos, that is, $A[pos:n]$ contains the original elements in $A[pos:n]$ with the smallest value at index pos. During iteration pos, there are two cases. In the first case, suppose that $A[pos] \geq A[pos-1]$. Then the body of the **if** statement is not executed, and $A[pos-1]$ is the smallest value in $A[pos-1:n]$. In the second case, suppose that $A[pos] < A[pos-1]$. Then the body of the **if** statement is executed, swapping the values at index pos and $pos-1$. Then $A[pos-1]$ is now the smallest value in $A[pos-1:n]$ after the swap. In both cases, $A[pos-1]$ is the smallest value in $A[pos-1:n]$. Furthermore, since $A[pos:n]$ contains the original elements in $A[pos:n]$ and $A[pos-1]$ is the original element at index $pos-1$, $A[pos-1:n]$ contains the original elements in $A[pos-1:n]$. Before the next iteration of the loop, *pos* is decremented. Once this happens, our conclusion is rewritten as "$A[pos : n]$ contains the original elements in $A[pos : n]$ with the smallest value at index *pos*". Therefore, the maintenance step holds. YES!! □

Lemma 3 (Inner Loop Termination Condition). *After the inner for loop terminates, $A[front : n]$ contains the original elements in $A[front : n]$, with the smallest value at index $front$.*

Proof. In the final iteration of the inner for loop, *pos* is equal to $front$. Therefore, the termination condition directly follows **Lemma 1**. □

Lemma 4 (Outer Loop Invariant). *Before each iteration $front$ of the outer for loop, $A[1 : front - 1]$ is in sorted order* <u>*and*</u> *each element in $A[1 : front - 1]$ is less than all elements in $A[front : n]$.*

Proof. **Initialization:** Before the first iteration, $front = 1$. In this case, the loop invariant says that $A[1 : 0]$ is in sorted order and each element in $A[1 : 0]$ is less than all elements in $A[1 : n]$. This is true because there are no elements in $A[1 : 0]$, so our loop invariant holds.

Maintenance: Assume the loop invariant is true before some iteration $front$, that is, $A[1 : front - 1]$ is in sorted order and each element in $A[1 : front - 1]$ is less than all elements in $A[front : n]$. Lemma 3 (the termination condition of the inner loop) tells us that, at the end of the inner loop, $A[front : n]$ contains the original elements in $A[front : n]$, with the smallest value at index $front$. Since we have assumed that the inner loop invariant is true , we know that $A[front]$ is the smallest value in $A[front : n]$. And since each element in $A[1 : front - 1]$ is smaller than $A[front]$ and $A[1 : front - 1]$ is in sorted order, $A[1 : front]$ will also be in sorted order. And since $A[front]$ is the smallest value in $A[front : n]$, each element in $A[1 : front]$ will be less than all elements in $A[front + 1 : n]$. write as complete conclusion here for clarity Before the next iteration of the loop, $front$ is incremented After this happens, our conclusion is rewritten as "$A[1 : front - 1]$ is in sorted order and each element in $A[1 : front - 1]$ is less than all elements in $A[front : n]$". Therefore, the maintenance step holds. □

Lemma 5 (Outer Loop Termination Condition). *After the outer for loop terminates, $A[1 : n-1]$ is in sorted order and each element in $A[1 : n-1]$ is less than all elements in $A[n : n]$.*

Proof. In the final iteration of the outer for loop, $front = n$. Therefore, the termination condition directly follows **Lemma 3**. □

Theorem 1. NESTED-SORT *correctly sorts the elements in $A[1 : n]$ in ascending order.*

Proof. According to Lemma 5, after the outer for loop terminates, $A[1 : n-1]$ is in sorted order and each element in $A[1 : n-1]$ is less than all elements in $A[n : n]$. But since $A[n : n]$ only contains 1 element $A[n]$, $A[n]$ is greater than every element in $A[1 : n-1]$. And since $A[1 : n-1]$ is in sorted order, $A[1 : n]$ is also in sorted order. Therefore, we can conclude that NESTED-SORT correctly sorts the elements in $A[1 : n]$ in ascending order. $\square$