CSC 505 - Homework 1

Charles Haithcock 1/27/2017

1 (8 points) Goal: Practice analysis of algorithms.

Consider the algorithm represented by the following program fragment; assume that x and n are non-negative.

```
1  BAR(int x, int n)
2  {
3      int sum=0;
4      if x>10 then {
5          for (int i=1; i<=n2; i++) {
6              sum=sum*i;
7          }
8      } else {
9          for (int i=1; i<=n; i++) {
10             sum=sum+i;
11         }
12     }
13     return x + sum;
14 }
```

a) (2 points) What value (as a function of x and n) does BAR return? For full marks, please justify your solution.

Most logic structures have mathematical counterparts; loops are series where the loop body indicates if the series is a sum or product series, if- statements create piecewise functions, etc.

In regards to the code snippet, when taken literally from the code, lines 5- 7 create a series, in which the body of the loop, line 6, indicates the series is a product series with i starting at 1 and going until n^2 and being incremented along the way to itself. As such, lines 5-7 can be written as $0\prod_{i=1}^{n^2} i$.

Similarly, line 10 being a sum indicates lines 9-11 is a summation of i as it increases towards n with sum. As such, lines 9-11 can be written as $0 + \sum_{i=1}^{n} i$.

An if-statement encapsulates lines 5-7 and 9-11. The conditional of the if-statement therein becomes the conditional for the piecewise function. As such, lines 4-12 can be written as

$$\begin{cases} 0\prod_{i=1}^{n^2} i & x > 10 \\ 0 + \sum_{i=1}^{n} i & \text{otherwise.} \end{cases} \tag{1}$$

Finally, sum is added to x before returning, so the piecewise function will need to be summed with

x as the last operation, written as such:

$$x + \begin{cases} 0 \prod_{i=1}^{n^2} i & x > 10 \\ 0 + \sum_{i=1}^{n} i & \text{otherwise} \end{cases} \tag{2}$$

The full mathematical representation of the code snippet is as follows:

$$\text{BAR}(x, n) = x + \begin{cases} 0 \prod_{i=1}^{n^2} i & x > 10 \\ 0 + \sum_{i=1}^{n} i & \text{otherwise} \end{cases} \tag{3}$$

This, however can be simplified to the following since $sum = 0$ before the product series and the sum has a known evaluated form:

$$\text{BAR}(x, n) = x + \begin{cases} \frac{n(n+1)}{2} & x \leq 10 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

b) (4 points) Use only ARITHMETIC operations ($*$ in line 6, and + in lines 10 & 13) as basic operations Compute the exact worst-case running time of BAR as a function of x and n. For full marks, please justify your solution.

Below is the code again except with analysis annotations assuming no compiler optimizations:

```
1     BAR(int x, int n)                      // Cost   Times ran
2     {                                      // ----------------
3         int sum=0;                         //  c1      1
4         if x>10 then {                     //  c2      1
5             for (int i=1; i<=n2; i++) {    //  c3      n
6                 sum=sum*i;                 //  c4      n^2 - 1
7             }                              //
8         } else {                           //
9             for (int i=1; i<=n; i++) {     //  c5      n
10                sum=sum+i;                 //  c6      n - 1
11            }                              //
12        }                                  //
13        return x + sum;                    //  c7      1
14    }
```

However, given the if-statement, lines 4-7 and 8-12 are mutually exclusive and can not both run in the same function call, because of this, each section must be compared for which would produce the largest time: $n(n^2 - 1) > n(n - 1)$ when $n > 1$. As such, lines 5-7 are chosen to provide us a larger running time.

With the above choice, the total worst-running time then becomes:

$$T(n) = c_1 + c_2 + c_3 n + c_4(n^2 - 1) + c_7 \tag{5}$$

To simplify, all constants, $c*$ can be assumed constant value, 1, and eventually just dropped:

$$T(n) = 1 + 1 + 1n + 1(n^2 - 1) + 1 = n^2 + n + 2 = n^2 + n \tag{6}$$

c) (2 points) Assume that `x` remains constant while n goes to infinity. Derive a tight, big-Oh expression (dependent on n) for the running time of BAR. For full marks, please justify your solution.

From the definition of $O(g(n))$:

$$0 \leq f(n) \leq cg(n) \tag{7}$$

Prove:

$$0 \leq n^2 + n \in O(n^2) \tag{8}$$

Proof:

$$0 \leq n^2 + n \leq cn^2 \qquad \text{when } n \geq 1 \tag{9}$$

$$0 \leq 1 + \frac{1}{n} \leq c \qquad \text{when } n \geq 1 \tag{10}$$

Which implies:

$$\text{when } n \geq 1, c = 2 \tag{11}$$

$$\text{so when } n_0 = 1, c = 2, n^2 + n \in O(n^2) \tag{12}$$

2 (12 points) Purpose: Learn about Horner's rule for evaluating polynomials, practice running time analysis, learn how loop invariants are used to prove the correctness of an algorithm. Tip: re-read Section 2.1 in our textbook. Please solve problem 2-3 [a-d] on page 41 of the textbook.

From the text:

> The following code fragment implements Horner's rule for evaluating a polynomial
>
> $$P(x) = \sum_{k=0}^{n} a_k x^k$$
> $$= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n)\cdots)),$$
>
> given the coefficients $a_0, a_1, \ldots . a_n$ and a value for x:
>
> 1 $y = 0$
> 2 **for** $i = n$ **downto** 0
> 3 $\quad y = a_i + x \cdot y$
>
> ***a.*** In terms of Θ-notation, what is the running time of this code fragment for Horner's rule?

The running time of the code is straightforward; line 1, being a value assignment, is constant time ($\Theta(1)$); line 2, being a loop inherently containing a loop termination check and iteration term decrement each performed n times is $\Theta(n)$; line 3 is a sum and product performed $n - 1$ times (as the body of the loop is executed 1 time less than the loop check). As such, the total running time is $T(P(x)) = 1 + n + (n - 1) = 2n$. In terms of Θ- notation, $T(P(x)) = \Theta(n)$ when $n_0 = 1$ and $c = 2$.

> ***b.*** Write pseudocode to implement the naive polynomial-evaluation algorithm that computes each term of the polynomial from scratch. What is the running time of this algorithm? How does it compare to Horner's rule?

```
NAIVE_EVAL(int[] A, int x):       // Cost  Times Ran
    int x_init = x                // c_0     1
    int sum = 0                   // c_1     1
    for i = 0 upto A.length:      // c_2     n
        x = (x_init ** i)         // c_3     n-1
        A[i] += A[i] * x          // c_4     n-1
    for i = 0 upto A.length:      // c_5     n
        sum += A[i]               // c_6     n-1
    return sum                    // c_7     1
```

The pseudocode assumes powers are constant operations and any constant operation has a running time of $\Theta(1)$. Here, the analysis is similar to Horner's rule:

$$\begin{aligned} T(g(n)) &= c_0 + c_1 + c_2 n + c_3(n-1) + c_4(n-1) + c_5 n + c_6(n-1) + c_7 && (13)\\ &= 1 + 1 + n + n - 1 + n - 1 + n + n - 1 + 1 && (14)\\ &= 5n && (15)\\ &= \Theta(n) && (16) \end{aligned}$$

However, when not obfuscating details, the naive approach actually runs 2.5 longer.

> ***c.*** Consider the following loop invariant:
>
> At the start of each iteration of the **for** loop of lines 2-3,
>
> $$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k.$$
>
> Interpret a summation with no terms as equaling 0. Following the structure of the loop invariant proof presented in this chapter, use this loop invariant to show that, at termination,
>
> $$y = \sum_{k=0}^{n} a_k x^k$$

Initialization Prior to the first iteration of the first loop, variables are initialized as $y = 0$ and $i = n$. Since the loop terminates when $i = 0$, for the sake of analysis, n, and subsequently i are assumed $n, i > 0$. In such a case, no terms have yet been evaluated, so $y = 0$ still. The loop maintains this, as, when substituting the appropriate values in, the summation evaluates to

$$\begin{aligned} y &= \sum_{k=0}^{n-(n+1)} a_{k+n+1} x^k && (17)\\ &= \sum_{k=0}^{-1} a_{k+n+1} x^k && (18)\\ &= 0 && (19) \end{aligned}$$

or the zero sum as no such terms are possibly defined.

Maintenance Horner's rule operates from the notion that the i^{th} term is evaluated as $a_i x^1$ wherein the terms $i-1$ down to the 0^{th} term are yet to be evaluated but $i+1$ up to the n^{th} term are evaluated further by iteratively but implicitly multiplying them by the variable.

The summation describes this as

$$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1}x^k \tag{20}$$
$$= a_{i+1} + x(a_{i+2} + x(\cdots + x(a_{n-1} + a_n x)\cdots)) \tag{21}$$
$$= a_{i+1}x^0 + a_{i+2}x^1 + \cdots + a_{n-1}x^{n-(i+2)} + a_n x^{n-(i+1)} \tag{22}$$

The loop maintains this, as by the i^{th} iteration, the intermediate sum, y, will have iteratively multiplied x to itself $n-(i+1)$ times. Within each iteration, the a_i term is also summed in. Decrementing i in line 2 reestablishes the loop invariant so, when evaluating line 3, the next term is partially evaluated while every other term is evaluated further and closer to their original degree.

Termination Upon termination, $i = -1$ and fails the check of $i \geq 0$. When substituted into the prior partial summation

$$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1}x^k \tag{23}$$
$$= \sum_{k=0}^{n} a_k x^k \tag{24}$$
$$= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n)\cdots)) \tag{25}$$

or Horner's Rule.

The code maintains this because at termination, each a_i will have been added to the sum at iteration i. Likewise, by termination, the i^{th} term will have had x multiplied to itself i times via line 3.

d. Conclude by arguing that the given code fragment correctly evaluates a polynomial characterized by the coefficients $a_0, a_1, \ldots, a_n$.

The code fragment correctly evaluates a polynomial; the code first grabs the largest term, and multiplies it by x. At each iteration the next term is next term is partially calculated by summing the product of x with the previous partial summation with a_i. Upon termination, each a_i will have been implicitly multiplied with x i times and summed with the rest of the terms.

3 Purpose: Practice working with asymptotic notation. Please solve (12 points) 3-2 on page 61, (6 points) 3-4 [a-c] on page 62. For full marks justify your solutions.

From the text:

> ***3-2 Relative asymptotic growths***
> Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that $k \geq 1, \epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$lg^k n$	n^ϵ	Yes	Yes	No	No	No
b.	n^k	c^n	Yes	Yes	No	No	No
c.	$\sqrt{n}$	$n^{\sin n}$	No	No	No	No	No
d.	2^n	$2^{n/2}$	No	No	Yes	Yes	No
e.	$n^{\log c}$	$c^{\log n}$	Yes	No	Yes	No	Yes
f.	$\log(n!)$	$\log(n^n)$	Yes	No	Yes	No	Yes

a.

$$\lim_{n\to\infty} \frac{\log(n)^k}{n^\epsilon} = \frac{\infty}{\infty} \quad \text{L'Hopital's Rule!} \tag{26}$$

$$\frac{d\frac{\log(n)^k}{n^\epsilon}}{dn} = \frac{k(\log(n))^{k-1}}{n\ln(10)\epsilon n^{\epsilon-1}} = \frac{k(\log(n))^{k-1}}{\ln(10)\epsilon n^\epsilon} \quad \text{L'Hopital's Rule!} \tag{27}$$

$$\frac{d\frac{k(\log(n))^{k-1}}{\ln(10)\epsilon n^\epsilon}}{dn} = \frac{k(k-1)(\log(n))^{k-2}}{n\ln(10)^2\epsilon^2 n^{\epsilon-1}} = \frac{k(k-1)(\log(n))^{k-2}}{\ln(10)^2\epsilon^2 n^\epsilon} \quad \text{L'Hopital's Rule!} \tag{28}$$

$$\frac{d^k\frac{\log(n)^k}{n^\epsilon}}{d^k n} = \frac{k!}{\ln(10)^k\epsilon^k n^\epsilon}; \lim_{n\to\infty} \frac{k!}{\ln(10)^k\epsilon^k n^\epsilon} = 0 \tag{29}$$

b.

$$\lim_{n\to\infty} \frac{n^k}{c^n} = \frac{\infty}{\infty}; \quad \text{L'Hopital's Rule!} \tag{30}$$

$$\frac{d\frac{n^k}{c^n}}{dn} = \frac{kn^{k-1}}{\ln(c)^k c^n}; \quad \text{L'Hopital's Rule!} \tag{31}$$

$$\frac{d\frac{kn^{k-1}}{\ln(c)c^n}}{dn} = \frac{k(k-1)n^{k-2}}{\ln(c)^2 c^n}; \quad \text{L'Hopital's Rule!} \tag{32}$$

$$\frac{d^k\frac{n^k}{c^n}}{d^k n} = \frac{k!}{\ln(c)^k c^n}; \lim_{n\to\infty} \frac{k!}{\ln(c)^k c^n;} = 0 \tag{33}$$

c. $\sin(n)$ ranges in value $[-1, 1]$, so the limit is evaluated for both:

$$\lim_{n\to\infty} \frac{\sqrt{n}}{n^{-1}} = \infty; \lim_{n\to\infty} \frac{\sqrt{n}}{n} = \frac{\infty}{\infty}; \quad \text{L'Hopital's Rule!} \tag{34}$$

$$\frac{d\frac{\sqrt{n}}{n}}{dn} = \frac{\frac{1}{2\sqrt{n}}}{1} = \frac{1}{2\sqrt{n}} \tag{35}$$

From the above, when $\sin(n) = -1, \sqrt{n} \in \omega(n^{\sin(n)})$, however, when $\sin(n) = 1, \sqrt{n} \in \Theta(\sin(n))$. Due to this oscilating behavior, none apply.

d.

$$\lim_{n\to\infty} \frac{2^n}{2^{\frac{n}{2}}} = \lim_{n\to\infty} 2^{\frac{n}{2}} = \infty \tag{36}$$

e.

$$\log(n^{\log(c)}) = \log(c)\log(n) \tag{37}$$

$$\log(c^{\log(n)}) = \log(n)\log(c) \tag{38}$$

$$\lim_{n\to\infty} \frac{\log(c)\log(n)}{\log(c)\log(n)} = 1 \tag{39}$$

f.

$$\log(n!) = \sum_{i=0}^{n} \log(i) = n\log(n) \tag{40}$$

$$\log(n^n) = n\log(n) \tag{41}$$

$$\log(n!) = \log(n^n) \implies \lim_{n\to\infty} \frac{\log(n!)}{\log(n^n)} = 1 \tag{42}$$

Note above Stirling's approximation: $\log(n!) = \sum_{i=0}^{n} \log(i)$ [1]

Asymptotic notation properties

Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following conjectures:

a. $f(n) = O(g(n)) \implies g(n) = O(f(n))$

If $f(n) = O(g(n))$, then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ must at least hold true. This is to say $g(n)$ grows faster than $f(n)$. However, if $g(n) = O(f(n))$, then $\lim_{x\to\infty} \frac{g(n)}{n(n)} = 0$ which is impossible simply because $g(n)$ grows faster than $f(n)$ for large values of n.

As a counter example, consider $f(n) = n$ and $g(n) = n^2$. Here, while we do not have a tight bound, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ so $g(n)$ is at least an upper bound. That being said, $n^2 \notin O(n)$ since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$ here.

b. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$

As a counter example, consider $f(n) = n^2$ and $g(n) = n \implies n^2 + n \notin \Theta(n)$. As proof by contradiction, assume $n^2 + n \notin \Theta(n)$. If this is true, then

$$f(n) \in \Theta(g(n)) \implies \Omega(g(n)) \cup O(g(n)) \tag{43}$$

$$\implies f(n) \in O(g(n)) = n^2 \in O(n) \tag{44}$$

From the definition of O-notation, $0 \leq n^2 \leq cn$, which is impossible as c is a constant and, as n grow arbitrarily large, a value of c exists where the inequality does not hold true.

c. $f(n) = O(g(n)) \implies \log(f(n)) = O(\log(g(n)))$, where $\log(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n.

$$\begin{aligned} f(n) = O(g(n)) &\implies 0 \leq f(n) \leq cg(n), c > 0, n \geq n_0 && (45) \\ &\implies 0 \leq \log(f(n)) \leq \log(cg(n)) && (46) \\ &\implies 0 \leq \log(f(n)) \leq \log(c) + \log(g(n)) && (47) \\ &\implies \log(f(n)) = O(\log(g(n))) && (48) \end{aligned}$$

4 (5 points) Purpose: Practice working with asymptotic notation. Rank the following functions by order of asymptotic growth; that is, find an arrangement $g_1, g_2, \ldots$ of the below functions with $g_1 \in \Omega(g_2), g_2 \in \Omega(g_3), \ldots$ Mark the functions that are asymptotically equivalent, i.e. $g_k \in \Omega(g_{k+1})$ by a *. Here, lg indicates the binary logarithm. $3\sqrt{n}, \log(n^n), n^{\frac{2}{3}}, 2^{-n}, \frac{n}{2} + \log(n), \sqrt(n)\log(n)$

$$2^{-n} \tag{49}$$
$$3\sqrt{n} \tag{50}$$
$$\sqrt(n)\log(n)* \tag{51}$$
$$n^{\frac{2}{3}}* \tag{52}$$
$$\frac{n}{2} + \log(n) \tag{53}$$
$$\log(n^n) \tag{54}$$
$$\tag{55}$$

The above is based largely on the following notions:

- Since $\lim_{x\to\infty} 2^{-n} = 0$, and the rest of the functions grow monotonically, this function has the lowest growth rate
- $3\sqrt{n}$ grows slower than $\sqrt(n)\log(n)$ when $\log(n) > 9$
- $\sqrt(n)\log(n)$ grows slower than $\frac{n}{2} + \log(n)$ as $\frac{n}{2}$ grows linearly and is added to $\log(n)$ while $3\sqrt{n}$, $\sqrt(n)\log(n)$, and $n^{\frac{2}{3}}$ do not grow linearly
- Despite $\log(n^n)$ being a logarithm, having the n be both the base and the power for the exponent means extremely quick growth
- $n^{\frac{2}{3}}$ and $\sqrt{n}\log(n)$ grow similarly as their growths are dominated by a root operation.

5 (6 points) Purpose: Practice proving asymptotic relationships. In proving big-oh and big-omega bounds there is a relationship between the c that is used and the smallest n_0 that will work (for O, the smaller the c, the larger the n_0; for big-omega, the larger the c, the larger the n_0). In each of the following situations, describe (the smallest integer) n_0 as a function of c. You'll have to use the ceiling function to ensure that n_0 is an integer. Your solution should also give you a lower bound (for big-oh) or an upper bound (for big-omega) on the constant c.

(a) (2 points) Let $f(n) = 2n^3 + 7n^2$ and prove that $f(n) \in O(n^3)$

$$0 \leq 2n^3 + 7n^2 \leq cn^3 \tag{56}$$
$$0 \leq 2 + \frac{7}{n} \leq c \tag{57}$$
$$\tag{58}$$

$$0 \leq 2 + \frac{7}{n} \qquad\qquad 2 + \frac{7}{n} \leq c \tag{59}$$
$$-2 \leq \frac{7}{n} \qquad\qquad \frac{7}{n} \leq c - 2 \tag{60}$$
$$n \geq \frac{-7}{2} \qquad\qquad \frac{7}{c-2} \leq n; c \neq 2 \tag{61}$$

So $n_0(c) = \frac{7}{c-2}$. However, this could be further restrained, as when $0 < c < 2$, $\frac{7}{c-2} < 0$. This is disallowed, as $n > 0$ and the inequality would allow $n = 0$. As such, $c > 2$. So

$$n_0(c) = \lceil \frac{7}{c-2} \rceil, c > 2 \implies f(n) \in O(n^3)$$

(b) (2 points) Let $f(n) = 2n^3 - 7n^2$ and prove that $f(n) \in \Omega(n^3)$

$$0 \leq cn^3 \leq 2n^3 - 7n^2 \tag{62}$$
$$0 \leq c \leq 2 - \frac{7}{n} \tag{63}$$

$$0 < c \qquad\qquad c \leq 2 - \frac{7}{n} \tag{64}$$
$$\frac{7}{n} \leq 2 - c \tag{65}$$
$$\frac{7}{2-c} \leq n, c \neq 2 \tag{66}$$

Similar to above, c can be constrained further since when $c > 2$, $\frac{7}{2-c} < 0 \implies n < 0$ at some point which is impossible. So

$$n_0(c) = \lceil \frac{7}{2-c} \rceil, 0 < c < 2$$

(c) (2 points) Let $f(n) = 3n^3 + n^2$ and prove that $f(n) \in O(n^4)$

$$0 \leq 3n^3 + n^2 \leq cn^4 \quad (67)$$

$$0 \leq 3n^3 + n^2 \qquad 3n^3 + n^2 \leq cn^4 \quad (68)$$

$$0 \leq n^2(3n + 1) \qquad 3n + 1 \leq cn^2 \quad (69)$$

$$0 \leq n^2 \qquad 0 \leq 3n + 1 \qquad \frac{3n+1}{n^2} \leq c \quad (70)$$

$$0 \leq n \qquad \frac{-1}{3} \leq n \quad (71)$$

Unfortunately this is where I am stuck. My inclination would be to attempt to solve for $cn^2 - 3n - 1 \geq 0$ with the quadratic equation, but I have never done it with inequalities, so I can not assume $n \geq \frac{3 \pm \sqrt{9+4c}}{2c}$ is supposedly the correct answer. Wolfram Alpha seems to indicate $n > \frac{\sqrt{\frac{4c+9}{c^2}}c+3}{2c}, c > 0$ which would give me a $n_0(c)$ but I have no idea how this is determined.

6 Sources

- Weisstein, Eric W. "Stirling's Approximation." From Mathworld–A Wolfram Web Resource. `http://mathworld/wolfram.com/StirlingsApproximation.html`