Analysis of Algorithms: Assignment 1

Hal DiMarchi

September 14, 2018

Contents

1 Section 1.3

1.1 Problem 1

1.1.1 b.

Is this algorithm stable?

Answer: No. As you can see in the innermost block of code, if $A[i] >= A[j]$ (aka not less than) Count[i] is incremented, meaning A[i] will be moved ahead of A[j] when these elements are equal. This reverses their relative order, making the algorithm unstable.

1.2 c.

Is it in place?

Answer: No. A new array (S) is returned from the function, thus the algorithm violates the definition of in-place.

2 Section 1.4

2.1 Problem 3

Create a simple algorithm for the string-matching problem

Answer: Create a list of all the substrings in your string that are of the same length as your pattern. Simultaneously create a list of integers that denote the index at which each substring begins in the parent string. Compare your pattern to each of these substrings. If the substring and the pattern match, return, or push, the corresponding starting index of the substring.

3 Section 2.1

3.1 Problem 7

Gaussian elimination, the classic algorithm for solving systems of n linear equations in n unknowns, requires about $1/3(n^3)$ multiplications, which is the algorithm's basic operation.

3.1.1 a.

How much longer should you expect Gaussian elimination to work on a system of 1000 equations versus a system of 500 equations

Answer:

$$G(500) = 1/3(500^3) = 41666666.6667 \quad (1)$$
$$G(1000) = 1/3(1000^3) = 333333333.333 \quad (2)$$
$$G(1000) \div G(500) = 8 \quad (3)$$

I should expect it to take eight times longer on a system of 1000 equations versus a system of 500 equations.

3.1.2 b.

You are considering buying a computer that is 1000 times faster than the one you currently have. By what factor will the faster computer increase the sizes of systems solvable in the same amount of time as on the old computer?

Answer: If the computer is 1000 times faster than our equation becomes $10^{-3} * 1/3n^3$. Suppose an input of size n had a time t before the upgrade. The new time will be $t/1000$. If we plug $10n$ into our new equation we produce $1/1000 * 1/3(10n)^3$ or $1/1000 * 10^3 * t$ or t. From this we see that this new computer allows us to handle inputs increased by a factor of 10 at the same speed of our older computer.

3.2 Problem 8

For each of the following functions, indicate how much the function's value will change if its argument is increased fourfold.

3.2.1 a. $\log_2(n)$

Answer: $\log_2(4n) = log_2(4) + log_2(n) = 2 + log_2(n)$
The value will increase by 2.

3.2.2 b. $\sqrt{n}$

Answer: $\sqrt{4n} = 2\sqrt{n}$
The value will double.

3.2.3 c. n

Answer: $4 * n = 4n$
The value will increase by a factor of 4.

3.2.4 d. n^2

Answer: $(4n)^2 = 4n * 4n = 16n^2$
The value will increase by a factor of 16.

3.2.5 **e.** n^3

Answer: $(4n)^3 = 4n * 4n * 4n = 64n^3$
The value will increase by a factor of 64.

3.2.6 **f.** 2^n

Answer: $2^(4n) = 2^n * 2^n * 2^n * 2^n = (2^4)^n = 8^n * 2^n$
The value will increase by a factor of 8^n.

3.3 Problem 9

For each of the following pairs of functions, indicate whether the first function of each of the following pairs has a lower, same, or higher order of growth (to within a constant multiple) than the second function.

3.3.1 **a.** $n(n+1)$ *and* $2000n^2$

Answer: Same, as the highest order element in each is an n^2

3.3.2 **b.** $100n^2$ *and* $0.01n^3$

Answer: Lower, as the highest order element in the second is n^3 and only n^2 in the first. Constants do not matter.

3.3.3 **c.** $\log_2(n)$ *and* $\ln(n)$

Answer: Same, as both of these functions are on a logarithmic scale. Differing bases does not affect order of growth.

3.3.4 **d.** $\log_2^2(n)$ *and* $\log_2(n^2)$

Answer: Higher as $\log_2^2(n) = (\log_2(n))^2$ *and* $\log_2(n^2) = 2\log_2(n)$ meaning the first is quadratic relative to the second when placed on a logarithmic scale.

3.3.5 **e.** 2^{n-1} *and* 2^n

Answer: Same, as 2^{n-1} can be expressed as $2^n/2$, meaning it differs from 2^n by the constant of $1/2$. We can ignore constants when considering the order of growth.

3.3.6 **f.** $(n-1)!$ *and* $n!$

Answer: Same, as $(n-1)!$ can be expressed as $n!/n$ meaning in both functions the highest order term is $n!$. Thus they have the same order of growth.

4 Section 2.2

4.1 Problem 2

Use the informal definitions of O, Θ, and Ω to determine whether the following assertions are true or false

4.1.1 a. $n(n+1)/2 \in O(n^3)$

Answer:
True as $n(n+1)/2$ is on the order of n^2, thus it is certainly bounded above by Cn^3 where C is a constant.

4.1.2 b. $n(n+1)/2 \in O(n^2)$

Answer:
True as $n(n+1)/2 = n^2/2 + n/2 <= n^2/2 + n^2/2 = n^2 < 2n^2 for n > 0$, meaning it is bounded above by some Cn^2 where C is a constant. This satisfies the definition of $\in O(n^2)$.

4.1.3 c. $n(n+1)/2 \in \Theta(n^3)$

Answer:
False, as $n(n+1)/2$ is of order 2 and n^3 is of order 3. Thus there is no n_1 such that for all $n > n_1 n(n+1)/2 > Cn^3$ where C is a constant. Thus our lower bound is missing, failing to satisfy the definition of $\in \Theta(n^3)$.

4.1.4 d. $n(n+1)/2 \in \Omega(n)$

Answer:
True. $n(n+1)/2 = (n^2+n)/2 > n/2 for all n > 1$. Thus $n(n+1)/2$ is bounded below by some Cn where C is a constant. This satisfies the definition of $\in \Omega(n)$.

5 Section 2.3

5.1 Problem 1

Compute the following sums.

5.1.1 a. $1 + 3 + 5 + 7 + ... + 999$

Answer:
$\sum_{i=1}^{999} i = (999(1000))/2 = 499500$

5.1.2 **b.** $2 + 4 + 8 + 16 + ... + 1024$

Answer: $\sum_{i=1}^{10} 2^i = ((2^{11} - 1)/(2 - 1)) - 1$(for sum starting at 1 rather than 0)$= 2^{11} - 2 = 2046$

5.1.3 **c.** $\sum_{i=3}^{n+1} 1$

Answer: $\sum_{i=3}^{n+1} 1 = n + 1 - 3 + 1 = n - 3$

5.1.4 **d.** $\sum_{i=3}^{n+1} i$

Answer:
$\sum_{i=3}^{n+1} i = \sum_{i=1}^{n+1} i - \sum_{i=1}^{3} = (n + 1(n + 2))/2 - 6$

5.1.5 **e.** $\sum_{i=0}^{n-1} i(i + 1)$

Answer:
$\sum_{i=0}^{n-1} i(i+1) = \sum_{i=0}^{n-1} i * \sum_{i=0}^{n-1} i+1 = ((n-1)(n))/2 * \sum_{i=0}^{n} i = (n^2-n)/2*(n^2+n)/2 = n^4 - n^2$

5.1.6 **f.** $\sum_{j=1}^{n} 3^{j+1}$

Answer:
$\sum_{j=1}^{n} 3^{j+1} = \sum_{j=1}^{n} 3 * 3^j = 3 \sum_{j=1}^{n} 3^j = 3(3^{n+1} - 1)/2$

5.1.7 **g.** $\sum_{i=1}^{n} \sum_{j=1}^{n} ij$

Answer:
$\sum_{i=1}^{n} \sum_{j=1}^{n} ij = \sum_{i=1}^{n} (i * \sum_{j=1}^{n} j) = \sum_{i=1}^{n} (i * (n(n + 1))/2) = (n(n + 1))/2 * \sum_{i=1}^{n} i = (n(n + 1))/2 * (n(n + 1))/2 = ((n(n + 1))/2)^2$

5.1.8 **h.** $\sum_{i=1}^{n} 1/(i(i + 1))$

Answer:
$\sum_{i=1}^{n} 1/(i(i+1)) = \sum_{i=1}^{n} (1/i-1/(i+1)) = \sum_{i=1}^{n} 1/i - \sum_{i=1}^{n} 1/(i+1) = \ln(n) - \sum_{i=2}^{n} 1/i =$

$\ln(n) - (\sum_{i=1}^{n} 1/i - \sum_{i=1}^{1} \ln(n) - (ln(n) - 1) = 1$

5.2 Problem 5

Answer questions (a)–(e) of Problem 4 about this algorithm. (Not pictured)

5.2.1 a.

What does this algorithm compute?

Answer: This algorithm computes the range of the values in a list.

5.2.2 b.

What is its basic operation?

Answer: Its basic operation is comparison.

5.2.3 c.

How many times is the basic operation executed?

Answer: This operation is executed $2(n-1)$ times, where n is the size of the array.

5.2.4 d.

What is the efficiency class of this algorithm?

Answer: The efficiency class of this algorithm is $O(n)$.

5.2.5 e.

Suggest an improvement, or a better algorithm altogether, and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

Answer: This algorithm does some unnecessary checking. If a value is less than the minimum value we do not need to check if it is also more than the maximum value, as the minimum and maximum values started out as equal. From this we can conclude that all values that are less than the minimum value are also not greater than the maximum value. If we change these two if statements into an if else statement we can go from $2(n-1)$ executions of the basic operations to $(n-1)$ basic operations in our best case scenario.

6 Section 2.4

6.1 Problem 1

Solve the following recurrence relations.

6.1.1 a. $x(n) = x(n-1) + 5\ \forall n,\ n > 1, x(1) = 0$

Answer: Clearly $x(n) = 5(n-1)$.

6.1.2 b. $x(n) = 3x(n-1)\ \forall n,\ n > 1, x(1) = 4$

Answer: Forward substitution:

n	1	2	3	4
$x(n)$	4	12	36	108

Clearly $x(n) = 3^{n-1} * 4$

6.1.3 c. $x(n) = x(n-1) + n\ \forall n,\ n > 0, x(0) = 0$

Answer: $x(n)$ is clearly $\sum_{0}^{n} i$. Thus $x(n) = (n(n+1))/2$

6.1.4 d. $x(n) = x(n/2) + n\ \forall n,\ n > 1, x(1) = 1$ (solve for $n = 2^k$)

Answer:

$$n = 2^k;\ x(2^k) = x(2^{k-1}) + 2^k.$$
$$x(2^{k-1}) = x(2^{k-2}) + 2^{k-1}$$

Clearly, $x(2^k) = \sum_{i=1}^{k} 2^i$

6.1.5 e. $x(n) = x(n/3) + 1\ \forall n,\ n > 1, x(1) = 1$ (solve for $n = 3^k$)

Answer:

$$n = 3^k;\ x(3^k) = x(3^{k-1}) + 1.$$
$$x(3^{k-1}) = x(3^{k-2}) + 1$$

Clearly, $x(3^k) = \sum_{i=0}^{k} 1 = k$

6.2 Problem 4

```
Q(n):
  if (n=1)
    return 1
  return Q(n - 1) + 2 * (n - 1)
```

6.2.1 a.

Set up a recurrence relation for this function's values and solve it to determine what this algorithm computes.

Answer:

Recurrence Relation: $Q(n) = Q(n-1) + 2(n-1)\ \forall n,\ n > 1, Q(1) = 1$

Backward Substitution:

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

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

Clearly, $Q(n) = (\sum\limits_{i=2}^{n} 2(i-1)) + 1$

6.2.2 b.

Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.

Answer:

Recurrence Relation: $M(n) = M(n-1) + 1\ \forall n,\ n > 1, M(1) = 0$

Clearly, $M(n) = \sum\limits_{i=2}^{n} 1$

6.2.3 c.

Set up a recurrence relation for the number of additions/subtractions made by this algorithm and solve it

Answer:

Recurrence Relation: $A(n) = A(n-1) + 2\ \forall n,\ n > 1, A(1) = 0$

Clearly, $A(n) = \sum\limits_{i=2}^{n} 2$