HW1

Ronan Wallace

February 21, 2019

Section 2.2 p.59-61

2a) $n(n+1)/2 \in O(n^3) \rightarrow$ **True** because $n^2 < n^3$

2b) $n(n+1)/2 \in O(n^2) \rightarrow$ **True** because $n^2 = n^2$

2c) $n(n+1)/2 \in \theta(n^3) \rightarrow$ **False** because $n^2 \neq n^3$

2d) $n(n+1)/2 \in \Omega(n) \rightarrow$ **True** because $n^2 > n$

7a) If $t(n) \in O(g(n))$, then $g(n) \in \Omega(t(n))$

True. By definition, big-oh is the set of all functions with lower or same order of growth of a given function (to within a constant multiple, as n goes to infinity). Big-omega is the set of all functions with a higher or same order of growth as a given function (to within a constant multiple, as n goes to infinity). Both expressions fulfill both definitions. For example, assume $t(n) = n$ and $g(n) = n^2$. This example cooperates with both given definitions. The statement if $n \in O(n^2)$, then $n^2 \in \Omega(n)$ is true.

7b) $\Theta(\alpha g(n)) = \Theta(g(n))$, where $\alpha > 0$

True. By definition, big-theta is the same order of growth. This statement is saying that both expressions are equal to each other when it comes to order of growth. Because $g(n)$ is equivalent to itself, both expressions will have the same highest degree of n. And, since alpha is a constant multiple, it would have no affect on the growth rate of the first expression. Therefore, the order of growth between both expressions will be equivalent.

7c) $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

True. This statement says that big-theta is equivalent to the intersection of big-oh and big-omega. The commonality between big-oh and big-omega is the set of functions that have the same order of growth as a given function. The definition of big-theta is that it is the set of all functions that have the same order of growth as a given function. Therefore, the two expressions are equivalent.

7d) For any two non-negative functions $t(n)$ and $g(n)$ defined on the set of non-negative integers, either $t(n) \in O(g(n))$, or $t(n) \in \Omega(g(n))$, or both.

False. Counterexample: $t(n) =$ cosine ; $g(n) =$ cosine

11) ALGORITHM *WeightCheck*(coins)

//Determines if a fake coin is lighter or heavier than a genuine coin
//*Input*: A pile of n coins, one fake coin and all others are genuine coins
//*Output*: True - Fake coin is heavier than genuine coin; False - Fake coin is lighter than genuine coin

if n % 3 is equal to 0 **do**
  divide coins into 3 piles
  pile1, *pile2*, *pile3* ← three divided piles
  if *pile1* weight == *pile2* weight **do**
    if *pile1* weight < *pile3* weight **do**
      return *true*
    else do
      return *false*
  else if *pile1* weight == *pile3* weight **do**
    if *pile1* weight < *pile2* weight **do**
      return *true*
    else do
      return *false*
  else do
    if *pile2* weight < *pile1* weight **do**
      return *true*
    else do
      return *false*
else if n % 3 is equal to 1 **do**
  divide coins into 3 equal piles with one remaining coin
  pile1, *pile2*, *pile3* ← three divided piles
  remainingCoin ← remaining coin
  if *pile1* weight == *pile2* weight && *pile2* weight == *pile3* weight **do**
    replace a coin in *pile3* with *remainingCoin*
    if *pile1* weight < *pile3* weight **do**

```
        return true
    else do
        return false
else if pile1 weight == pile2 weight do
    if pile1 weight < pile3 weight do
        return true
    else do
        return false
else if pile1 weight == pile3 weight do
    if pile1 weight < pile2 weight do
        return true
    else do
        return false
else do
    if pile2 weight < pile1 weight do
        return true
    else do
        return false
else if n % 3 is equal to 2 do
    divide coins into 3 equal piles with one remaining coin
    pile1, pile2, pile3 ← three divided piles
    remainingCoin1 ← first remaining coin
    remainingCoin2 ← second remaining coin
    if pile1 weight == pile2 weight && pile2 weight == pile3 weight do
        replace a coin in pile2 with remainingCoin1
        replace a coin in pile3 with remainingCoin2
        if pile1 weight == pile2 weight do
            if pile1 weight < pile3 weight do
                return true
            else do
                return false
        else do
            if pile1 weight < pile2  weight do
                return true
            else do
                return false
    else if pile1 weight == pile2 weight do
        if pile1 weight < pile3 weight do
            return true
        else do
            return false
    else if pile1 weight == pile3 weight do
        if pile1 weight < pile2 weight do
            return true
        else do
            return false
```

else do
    if *pile2* weight < *pile1* weight **do**
        return *true*
    else do
        return *false*
else do
    return *Error Message*

Section 2.3 p.67-68

1c)

$$\sum_{i=3}^{n+1} 1 \to u - l + 1 \to (n+1) - 3 + 1 \to n - 1$$

1d)

$$\sum_{i=3}^{n+1} i \to \frac{n(n+1)}{2} \to \frac{(n+1)((n+1)+1)}{2} \to \frac{(n+1)(n+2)}{2} \to \frac{n^2+3n+2}{2}$$

2d)

$$\sum_{i=0}^{n-1}\sum_{j=0}^{i-1}(i+j) \to \sum_{i=0}^{n-1}(\sum_{j=0}^{i-1} i + \sum_{j=0}^{i-1} j) \to \sum_{i=0}^{n-1}(i\sum_{j=0}^{i-1} 1 + \sum_{j=0}^{i-1} j) \to \sum_{i=0}^{n-1}[i((i-1)-0+1) + \frac{(i-1)((i-1)+1)}{2}] \to$$

$$\to \sum_{i=0}^{n-1}(\frac{2i^2}{2} + \frac{i^2-i}{2}) \to \sum_{i=0}^{n-1}(\frac{3i^2-i}{2}) \to \sum_{i=0}^{n-1}\frac{3}{2}i^2 - \sum_{i=0}^{n-1}\frac{1}{2}i \to \frac{3}{2}\sum_{i=0}^{n-1} i^2 - \frac{1}{2}\sum_{i=0}^{n-1} i \to$$

$$\to \frac{3}{2}[\frac{(n-1)((n-1)+1)(2(n-1)+1)}{2}] - \frac{1}{2}[\frac{(n-1)((n-1)+1)}{2}] \to \frac{3}{2}(\frac{2n^3-3n^2+n}{2}) - \frac{1}{2}(\frac{n^2-n}{2}) \to$$

$$\to \frac{(6n^3-9n^2+3n)-(n^2-n)}{4} \to \frac{6n^3-10n^2+4n}{4} \to \frac{3n^3-5n^2+2n}{2} \to \Theta(n^3)$$

5a) What does this algorithm compute?

This algorithm computes the range of a set of n real numbers

5b) What is its basic operation?

Either of the two *if* statements in the *for* loop (assign max min values if conditions are met).

5c) How many times is the basic operation executed?

The basic operation is executed n-1 times

5d) What is the efficiency class of this algorithm?

This algorithm has a *linear* $\Theta(n)$ efficiency class.

1.4 Program pp.38

10) Included in .zip folder