CS350 - Algorithms: Homework #3

Due on January 29, 2015

for Professor Andrew Black

Kristina Frye

Problem 1

Compute the following sums:

(a) $\sum_{i=0}^{n+2} i(i+1)$

$$\begin{aligned}
\sum_{i=0}^{n+2} i(i+1) &= \sum_{i=0}^{n+2} i^2 + i = \sum_{i=0}^{n+2} i^2 + \sum_{i=0}^{n+2} i \\
&= \frac{n(n+1)(2n+1)}{6} + (n+1)^2 + (n+2)^2 + \frac{n(n+1)}{2} + (n+1) + (n+2) \\
&= \frac{2n^3+3n^2+n}{6} + (n^2+2n+1) + (n^2+4n+4) + \frac{n^2+n}{2} + (n+1) + (n+2) \\
&= \frac{2n^3+3n^2+n}{6} + \frac{6(2n^2+6n+5)}{6} + \frac{n^2+n}{2} + \frac{2(2n+3)}{2} \\
&= \frac{2n^3+15n^2+37n+30}{6} + \frac{n^2+5n+6}{2} \\
&= \frac{2n^3+15n^2+37n+30}{6} + \frac{3n^2+15n+18}{6} \\
&= \frac{2n^3+18n^2+52n+48}{6}
\end{aligned}$$

(b) $\sum_{j=1}^{n} 3^{j+2}$

$$\begin{aligned}
\sum_{j=1}^{n} 3^{j+2} &= \sum_{j=1}^{n} 3^2 3^j = 3^2 \sum_{j=1}^{n} 3^j = 3^2 \cdot \frac{3^{n+1}-1}{2} - 1 = 3^2 \cdot \frac{3^{n+1}-1-2}{2} = 3^2 \cdot \frac{3^{n+1}-3}{2} = \frac{3^{n+3}-3^3}{2} \\
&= \frac{3^{n+3}-27}{2}
\end{aligned}$$

(c) $\sum_{i=1}^{n}\sum_{j=1}^{n} ij$

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{n} ij &= \sum_{i=1}^{n} i \sum_{j=1}^{n} j = \sum_{i=1}^{n} i \frac{n(n+1)}{2} = \frac{n(n+1)}{2} \sum_{i=1}^{n} i = \frac{n(n+1)}{2}\frac{n(n+1)}{2} = \frac{1}{4}n^2(n+1)^2 \\
&= \frac{1}{4}n^2(n^2+2n+1) = \frac{1}{4}\left(n^4+2n^3+n^2\right)
\end{aligned}$$

Problem 2

```
procedure SECRET($A[0..n-1]$)            ▷ An array $A[0..n-1]$ of $n$ real numbers
    $minval \leftarrow A[0]; maxval \leftarrow A[0]$
    for $i = 1$ to $n-1$ do
        if $A[i] < minval$ then
            $minval \leftarrow A[i]$
        end if
        if $A[i] > maxval$ then
            $maxval \leftarrow A[i]$
        end if
    end for
    return $maxval - minval$
end procedure
```

(a) What does this algorithm do? **A:** It computes the difference between the maximum and minimum value of the array of numbers (the range).

(b) What is its basic operation **A:** The comparison operator ($<$ and $>$).

(c) How many times is the basic operation executed in the best and worst cases? **A:** The best case is the same as the worse case. For all cases, each element needs to be checked once; there is no way to short circuit this algorithm (assuming there are no absolute min or max values.) The comparison is performed twice for each loop from $i = 1$ to $n - 1$. Therefore, it is performed $2 \cdot (n - 1)$ times.

(d) What is the efficiency class of this algorithm? **A:** $O(n)$ (linear)

Problem 3

Solve the following recurrence relations, using backwards substitution, or by calculating the first few terms and generalizing.

(a) $x(n) = 4x(n-1)$ for $n > 1, x(1) = 2$

$$\begin{aligned}
x(n) &= 4 \cdot x(n-1) \\
\implies x(n-1) &= 4 \cdot x(n-2) \\
\implies x(n) &= 4 \cdot 4 \cdot x(n-2) \\
\implies x(n) &= 4^k \cdot x(n-k) \\
\text{Let } k = n-1. \text{ Then: } x(n) &= 4^{n-1} \cdot x(n-(n-1)) \\
&= 4^{n-1} \cdot x(1) \\
&= 4^{n-1} \cdot 2 \\
\implies x(n) &= 2 \cdot 4^{n-1}
\end{aligned}$$

Problem 3 continued on next page. . .

(b) $x(n) = x(n-1) + n$ for $n > 0, x(0) = 3$

$$\begin{aligned}
x(n) &= x(n-1) + n \\
\implies x(n-1) &= x(n-2) + n - 1 \\
\implies x(n) &= x(n-2) + (n-1) + n \\
\implies x(n) &= x(n-k) + \sum_{k=0}^{n-1} n - k \\
\implies x(n) &= x(n-k) + \sum_{j=1}^{n} j \\
\implies x(n) &= x(n-k) + \frac{n(n+1)}{2} \\
\text{Let } k = n\text{. Then: } x(n) &= x(n-n) + \frac{n(n+1)}{2} \\
\implies x(n) &= 3 + \frac{n(n+1)}{2} \\
\implies x(n) &= \frac{n^2+n+6}{2}
\end{aligned}$$

(c) $x(n) = x(n/3) + 1$ for $n > 1, x(1) = 1$ (solve for $n = 3k$)

$$\begin{aligned}
x(n) &= x\left(\frac{n}{3}\right) + 1 \\
\implies x\left(\frac{n}{3}\right) &= x\left(\frac{n}{9}\right) + 1 \\
\implies x(n) &= x\left(\frac{n}{9}\right) + 1 + 1 \\
\implies x(n) &= x\left(\frac{n}{3^k}\right) + k \\
\text{Let } n = 3k\text{. Then: } x\left(3^k\right) &= x\left(\frac{3^k}{3^k}\right) + \log_3 3^k \\
&= 1 + \log_3 n \\
\implies x(n) &= 1 + \log_3 n \text{ where } n = 3^k
\end{aligned}$$

Problem 4

Consider this algorithm for *min*; it recursively divides the input array A into two halves:

(a) What is this Algorithms basic operation? **A:** The division operation $\frac{l+r}{2}$.

(b) Set up a recurrence relation for $B(n)$, the number of times that the basic operation is executed on an input array of size n, and **A:** $B(n) = 2 \cdot B\left(\frac{n}{2}\right) + 1, B(1) = 0, n > 0$

Problem 4 continued on next page. . .

```
procedure MIN2($A[l..r]$)
    if $l = r$ then
        return $A[l]$
    end if
    $mid \leftarrow \lfloor \frac{l+r}{2} \rfloor$
    $temp1 \leftarrow Min2(A[l..mid])$
    $temp2 \leftarrow Min2(A[mid+1..r])$
    if $temp1 \leq temp2$ then
        return $temp1$
    else
        return $temp2$
    end if
end procedure
```

(c) solve it

$$\begin{aligned}
B(n) &= 2 \cdot B\left(\frac{n}{2}\right) + 1 \\
\implies B\left(\frac{n}{2}\right) &= 2 \cdot B\left(\frac{n}{4}\right) + 1 \\
\implies B(n) &= 2 \cdot \left(2 \cdot B\left(\frac{n}{4}\right) + 1\right) + 1 \\
\implies B(n) &= 2 \cdot 2 \cdot B\left(\frac{n}{4}\right) + 2 + 1 \\
\implies B(n) &= 2^k B\left(\frac{n}{2^k}\right) + \sum_{j=0}^{k-1} 2^j \\
\implies B(n) &= 2^k B\left(\frac{n}{2^k}\right) + 2^k - 1 \\
\text{Let } n = 2^k\text{. Then: } B(n) &= n \cdot B\left(\frac{n}{n}\right) + n - 1 \\
B(n) &= n - 1
\end{aligned}$$

Problem 5

A network topology is a specification of how devices (computers, printers, etc.) are connected in a network. The figures below show two different network topologies: star, and fully-connected. For each of these topologies, design a brute-force algorithm that determines if a given graph has that topology. Assume $n > 3$, and that the input is an adjacency matrix of 1 and 0 values. Do not assume that the input matrix represents an undirected graph. Indicate time the efficiency class of your algorithm show your work.

Solution: Star Topology

```
function IsStarTopology($A[0..n-1][0..n-1]$)
    $key = -1$                                          ▷ 1 op
    for $i := 0$ to $n-1$ do
        $count = 0$                                     ▷ 1 op
        for $j := 0$ to $n-1$ do
            $count = count + A[i][j]$                   ▷ Count how many nodes are connected (1 op)
        end for
        if $count = n-1$ then                           ▷ Central node identified (1 op)
            if $key = -1$ then                          ▷ 1 op
                $key = i$
            else
                return false                            ▷ There can be only one central node
            end if
        else
            if $count \neq 1$ then                      ▷ All non-central nodes must have count of 1 (1 op)
                return false
            end if
        end if
    end for
    if $key = -1$ then                                  ▷ There must be a central node (1 op)
        return false
    end if
    for $i := 0$ to $n-1$ do                            ▷ Check for connection to identified central node
        if $A[i][key] = 0$ AND $i \neq key$ then        ▷ Check each node is connected to the central node (2 op)
            return false
        end if
    end for
    if $A[key][key] \neq 0$ then                        ▷ Check the central node is not connected to itself (1 op)
        return false
    end if
    return true
end function
```

Time efficency calculated assuming that additions and comparisons are counted as operations. Operations involved in for loops (additions and comparisons) are ignored since they will just add a constant to the total sum of operation calculations, are incidental to the algorithm, and won't affect the efficiency class. It is also assumed that $n-1$ is a know precalculated value. Worst-case operations are indicated in the comments shown in the algorithm.

Problem 5 continued on next page...

$$\begin{aligned}
ops(n) &= 3 + \sum_{i=0}^{n-1}\left(4 + \sum_{j=0}^{n-1} 1\right) + \sum_{i=0}^{n-1} 2 \\
&= 3 + \sum_{i=0}^{n-1}(4+n) + 2\cdot n \\
&= 3 + 2n + 4\cdot\left(\sum_{i=0}^{n-1} 1\right) + n\cdot\sum_{i=0}^{n-1} 1 \\
&= 3 + 2n + 4n + n\cdot n \\
&= 3 + 6n + n^2 \\
&\in O(n^2)
\end{aligned}$$

Solution: Fully-Connected Topology

```
function IsFullyConnected($A[0..n-1][0..n-1]$)
    for $i := 0$ to $n-1$ do
        $count = 0$                                   ▷ 1 op
        for $j := 0$ to $n-1$ do
            $count = count + A[i, j]$                 ▷ Count how many nodes are connected (1 op)
        end for    ▷ There should be $n-1$ connected nodes and a node should not be connected to itself
        if $count \neq n-1$ OR $A[i,i] \neq 0$ then   ▷ 2 op
            return false
        end if
    end for
    return true
end function
```

Time efficiency (same assumptions as before):

$$\begin{aligned}
ops(n) &= \left(\sum_{i=0}^{n-1} 1 + \left(\sum_{j=0}^{n-1} 1\right) + 2\right) \\
&= \sum_{i=0}^{n-1} 1 + n + 2 \\
&= \sum_{i=0}^{n-1} 1 + n\cdot\sum_{i=0}^{n-1} 1 + 2\cdot\sum_{i=0}^{n-1} 1 \\
&= n + n\cdot n + 2\cdot n \\
&= 3n + n^2 \\
&\in O(n^2)
\end{aligned}$$

Problem 6

A stack of fake coins There are n stacks of n identical-looking coins. All of the coins in one of these stacks are counterfeit, while all the coins in the other stacks are genuine. Every genuine coin weighs 10 grams; every fake weighs 11 grams. You have an analytical scale that can determine the exact weight of any number of coins.

(a) Devise a brute-force algorithm to identify the stack with the fake coins, and

```
function IndexOfFakeCoins($A[0..n-1]$)                ▷ Array of $n$ stacks of $n$ coins
    for $i := 0$ to $n-1$ do
        if $weight = 11n$ then
            return $i$
        end if
    end for
end function
```

(b) determine its worst-case efficiency class:
Basic operation = weight function. Worst case: last stack

$$ops(n) = \sum_{i=0}^{n-1} 1 = n \in O(n)$$

Problem 7

Alternating disks You have a row of 2n disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring disks. Design an algorithm for solving this puzzle and determine the number of moves it makes.

Solution:

```
function SortDisks($A[0..2n-1]$)
    for $i := 0$ to $n-1$ do                          ▷ Array should be sorted after halfway through
        for $j := 0$ to $n-1-i$ do                    ▷ Only need to check every other index
            if $A[2j+i] = D$ AND $A[2j+i+1] = L$ then
                $flip(A[2j+i], A[2j+i+1])$
            end if
        end for
    end for
end function
```

Problem 7 continued on next page. . .

Basic operation: flip

$$\begin{aligned}
ops(n) &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-i-1} 1 \\
&= \sum_{i=0}^{n-1} n - 1 - i + 1 = \sum_{i=0}^{n-1} n - i \\
&= \sum_{j=1}^{n} j = \frac{n(n+1)}{2} \\
&\in O(n^2)
\end{aligned}$$