CMPUT204
Department of Computing Science
University of Alberta

Solutions for Homework Assignment #1

Question 1. (10 marks) Prove or disprove each of the following statements:

a. (2 marks) $1^k + 2^k + \ldots + n^k \in O(n^{k+1})$ for every $k \in \mathbb{N}$.

Solution: This is true because for every $1 \leq i \leq n$ and $k \geq 0$: $i^k \leq n^k$. Thus the summation is at most $n \times n^k \in O(n^{k+1})$.

b. (2 marks) For any functions $f, g : \mathbb{N} \longrightarrow R^+$, either $f \in O(g)$ or $g \in O(f)$.

Solution: This is false. Suppose that

$$f(n) = \begin{cases} 0 & \text{if } n \text{ is odd} \\ n & \text{if } n \text{ is even} \end{cases} \qquad g(n) = \begin{cases} n & \text{if } n \text{ is odd} \\ 0 & \text{if } n \text{ is even} \end{cases}$$

It is easy to see that for any constants $c, d > 0$, even if $n > d$, always $f(n) > c.g(n)$ if n is even and $g(n) > c.f(n)$ if n is odd.

c. (2 marks) For any functions $f, g : \mathbb{N} \longrightarrow R^+$, $f \in O(g)$ if and only if $g \in \Omega(f)$.

Solution: This follows directly from the definition of O and Ω.

d. (2 marks) For any functions $f, f', g, g' : \mathbb{N} \longrightarrow R^+$, if $f \in O(f')$ and $g \in O(g')$ then $(f + g) \in O(\max(f', g'))$ where $(f + g)(n)$ is defined to be $f(n) + g(n)$ for any value of $n \in \mathbb{N}$.

Solution: This is true. By hypothesis about f and g, there are constants $c_1, c_2 \in \mathbb{R}^+$ and $n_1, n_2 \in \mathbb{N}$ such that:

$$\text{for all } n \geq n_1 : f(n) \leq c_1.f'(n), \text{ and for all } n \geq n_2 : g(n) \leq c_2.g'(n).$$

Therefore, for all $n \geq \max(n_1, n_2)$ we have:

$$\begin{aligned} (f+g)(n) &= f(n) + g(n) \\ &\leq c_1.f'(n) + c_2.g'(n) \\ &\leq \max(c_1, c_2).\max(f'(n), g'(n) + \max(c_1, c_2).\max(f'(n), g'(n) \\ &= 2.\max(c_1, c_2).\max(f'(n), g'(n)). \end{aligned}$$

So let $n_3 = \max(n_1, n_2)$ and $c_3 = 2.\max(c_1, c_2)$. Then for all $n \geq n_3$: $(f + g)(n) \leq c_3.\max(f', g')(n)$, as wanted.

e. (2 marks) For any functions $f, g : \mathbb{N} \longrightarrow R^{\geq 1}$, $\log(f(n)) \in \Theta(\log(g(n))$ if and only if $f(n) \in \Theta(g(n))$.

Solution: This is not true. For instance, let $f(n) = n$ and $g(n) = n^2$. Then $\log(f(n)) = \log n$ and $\log(g(n)) = 2 \log n$. Thus $\log(f(n)) \in \Theta(\log n)$ and $\log(g(n)) \in \Theta(\log n)$ but $f(n) \notin \Theta(g(n))$.

Question 2. (10 marks) Prove by induction that using an unlimited supply of 4-cent and 7-cent postage stamps we can make (exactly) any amount of postage that is 18 cents or more.

Solution: We prove by induction on n that any amount $n \geq 18$ of postage can be made using 4-cent and 7-cent stamps. The base case is $n = 18$ and this is easily seen to be true since we can use two 7-cent stamps plus one 4-cent one. So let's assume that $n \geq 18$ and that we can make n using 4- and

7-cent stamps. Assume that $n = 4 \times a + 7 \times b$, i.e. a and b are the numbers of 4-cent and 7-cent stamps (respectively) that we use to make n. We want to show how to make $n+1$ using 4- and 7-cent stamps.

Case 1: if $b \geq 1$ then we can replace one 7-cent stamp with two 4-cent stamps. So we get $4 \times (a+2) + 7 \times (b-1) = n+1$ and we are done.

Case 2: if $b = 0$ then, because $n \geq 18$, there must be at least five 4-cent stamps (otherwise the total will be at most 16). That is $a \geq 5$ and so $a - 5 \geq 0$. So we can replace five 4-cent stamps with three 7-cent stamps and get $4 \times (a-5) + 7(b+3) = n+1$. This completes the proof.

Question 3. (10 marks) Consider the following recursive function:

$$f(n) = \begin{cases} 3 & \text{if } n = 0 \\ 6 & \text{if } n = 1 \\ 5f(n-1) - 6f(n-2) + 2 & \text{if } n > 1 \end{cases}$$

Prove that $f(n) = 1 + 2^n + 3^n$ for all values of $n \geq 0$.

Solution: We prove this by induction on n. If $n = 0$ then $f(0) = 3 = 1 + 2^0 + 3^0$ and if $n = 1$ then $f(1) = 6 = 1 + 2^1 + 3^1$. Thus for both $n = 0$ and $n = 1$ the statement is true. Assume that the statement is true for all integers between 0 and $i \geq 1$ inclusive. We want to prove the statement for $i+1$. Since $i + 1 > 1$: $f(i+1) = 5f(i) - 6f(i-1) + 2$, and because $i \geq 1$: $i - 1 \geq 0$. So we can apply the induction hypothesis to i and $i-1$ and get $f(i) = 1 + 2^i + 3^i$ and $f(i-1) = 1 + 2^{i-1} + 3^{i-1}$. Thus:

$$\begin{aligned} f(i+1) &= 5(1 + 2^i + 3^i) - 6(1 + 2^{i-1} + 3^{i-1}) + 2 \\ &= 5 + 5 \times 2^i + 5 \times 3^i - 6 - 3 \times 2^i - 2 \times 3^i + 2 \\ &= 1 + (5-3) \times 2^i + (5-2) \times 3^i \\ &= 1 + 2^{i+1} + 3^{i+1}, \end{aligned}$$

as wanted.

Question 4. (10 marks) Consider the following program:

```
Smallest (A, f, l)
**Precondition: A[f ... l] is an array of integers, f ≤ l and f, l ∈ ℕ.
**Postcondition: Returns the smallest element of A[f ... l].
if f = l then
    return A[f]
else
    m = ⌊(f+l)/2⌋
    return min(Smallest(A, f, m), Smallest(A, m+1, l))
end if
```

a. (3 marks) Write a recurrence relation which describes the time complexity of this program in terms of the size of array A.

Solution: Let's define $n = l - f + 1$. Then $T(n)$ is constant for sufficiently small n and $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + c$ where c is a constant (for the time spent for computing m and taking the min and doing the recursive calls).

b. (2 marks) Using the master theorem, give a Θ bound on the complexity of this program.

Solution: We can use the master theorem with $a = 2$, $b = 2$ and $f(n) = n^0$, i.e. $c = 0$. This implies that $T(n) \in \Theta(n)$.

c. (3 marks) Write a non-recursive version of Smallest(A, f, l) (in pseudo-code).

Solution:

```
Smallest (A, f, l)
**Precondition: A[f ... l] is an array of integers, f ≤ l and f, l ∈ ℕ.
**Postcondition: Returns the smallest element of A[f ... l].
smallest ⟵ A[f]
for i ⟵ f + 1 to l do
    if A[i] < A[smallest] then
        smallest ⟵ i
    end if
end do
return A[smallest]
```

d. (2 marks) Find the worst case running time of this new program and express it in Θ notation.

Solution: The for loop iterates for $O(n)$ times where $n = l - f + 1$ is the size of array A and at each iteration it takes constant time. Therefore the running time is $\Theta(n)$.

Question 5. (12 marks) True or False? for each case give a justification of no more than two lines.

a. (2 marks) $n \log^5 n \in O(n^{\frac{5}{4}})$

Solution: This is true because we know that $\log^5 n \in O(n^{\frac{1}{4}})$. Alternatively by applying L'Hôpital's rule 5 times we get that $\lim_{n\to\infty} \frac{n \log^5 n}{n^{5/4}} = 0$.

b. (2 marks) $2^{3n} \in \Theta(6^n)$.

Solution: This is false since $2^{3n} = (2^3)^n = 8^n$ and $\lim_{n\to\infty} \frac{8^n}{6^n} \longrightarrow \infty$. So $2^{3n} \notin O(6^n)$ and thus $2^{3n} \notin \Theta(6^n)$.

c. (2 marks) $(\log\log n)^{\log n} \in \Omega(1.1^n)$

Solution: This is false because $(\log\log n)^{\log n} \in O(1.1^{\log n \log\log\log n})$ and $\lim_{n\to\infty} \frac{1.1^n}{1.1^{\log n \log\log\log n}} = \lim_{n\to\infty} 1.1^{\frac{n}{\log n \log\log\log n}} \longrightarrow \infty$.

d. (2 marks) $5^{\sqrt{\log n}} \in \Omega(n)$

Solution: This is false: $\lim_{n\to\infty} \frac{n}{5^{\sqrt{\log n}}} = \lim_{n\to\infty} \frac{2^{\lg n}}{2^{\lg 5\sqrt{\log n}}} = \lim_{n\to\infty} 2^{\lg n - \lg 5\sqrt{\log n}} \longrightarrow \infty$, because $\lg n \in \Theta((\sqrt{\log n})^2)$ and so the exponent goes to infinity.

e. (2 marks) $n^{\frac{\log\log n}{\log n}} \in O(n^{0.01})$.

Solution: This is true because $\frac{\log\log n}{\log n}$ is a decreasing function tending to zero as $n \longrightarrow \infty$. So for large enough n it will be smaller than 0.01.

f. (2 marks) $n! \in O(2^{n \log n})$

Solution: If $\log n$ is in base 2 (i.e. $\lg n$ which we intended to write) then the statement is true because $n! < n^n = (2^{\lg n})^n = 2^{n \lg n} \in O(2^{n \lg n})$. If $\log n$ is in base larger than 2 then the statement is false. If you assumed any of these two cases and got the correct answer you will get the mark.

Question 6. (12 marks) P2-2 on page 38.

Solution:

a) To prove that BubbleSort actually sorts, we should also prove that A' is a permutation of the elements of A. That is, each element of A appears exactly once in A'. This can be easily shown since in the algorithm the only operation we do on A is to exchange the value of two (consecutive) elements of A. So no new value is generated and no value is lost. At the end A' will be a permutation of A.

b) Let n be the length of array A. The loop invariant we propose for the "for loop" in lines 2-4 is: "$A[j]$ is the smallest among $A[j], \ldots, A[n]$".

Initialization: Clearly the invariant is true before the first iteration of the loop (i.e. when $j = n$) since $A[n]$ is the smallest among $A[n]$.

Maintenance: Suppose the loop invariant holds before some iteration of the loop, say when $j = k$. We have to show that it holds too after that iteration, i.e. when j becomes equal to $k - 1$. If $A[k-1] \leq A[k]$ then the statement on line 4 is not executed. Since $A[k]$ was the smallest among $A[k], \ldots, A[n]$ and because $A[k-1] \leq A[k]$, therefore after this iteration $A[k-1]$ is the smallest among $A[k-1], \ldots, A[n]$. On the other hand if $A[k-1] > A[k]$ then the algorithm exchanges it with $A[k]$. Now the element in $A[k-1]$ (which is the one that used to be in $A[k]$) is smaller than $A[k]$ and smaller than all in $A[k+1], \ldots, A[n]$. So $A[k-1]$ is the smallest among $A[k-1], \ldots, A[n]$. In both cases, after this iteration and at the beginning of the next iteration (when $j = k - 1$) the invariant holds as well.

Termination: After the last iteration, when $j = i$ the invariant holds, which implies that $A[i]$ is the smallest among $A[i], \ldots, A[n]$.

c) The loop invariant for the main loop is: "$A[1], \ldots, A[i-1]$ is sorted and all the elements in $A[1], \ldots, A[i-1]$ are smaller than or equal to all elements in $A[i], \ldots, A[n]$".

Initialization: At the beginning of the first iteration, $i = 1$ and there is nothing in $A[1], \ldots, A[i-1]$. So the invariant holds by definition.

Maintenance: Suppose that the invariant was true before an iteration of the loop, say when $i = k$. We have to show that it holds too after that iteration and before the next iteration begins, i.e. when $i = k + 1$. By the termination condition of the invariant in part b), when this iteration finishes, the inner loop has placed the smallest element among $A[k], \ldots, A[n]$ at location $A[k]$. Also, the inner loop never touches the elements at locations $A[1], \ldots, A[k-1]$. So they are unchanged. Since $A[1], \ldots, A[k-1]$ was sorted and each of $A[1], \ldots, A[k-1]$ is smaller than (or equal to) each of $A[k], \ldots, A[n]$ and because $A[k]$ contains the smallest of $A[k], \ldots, A[n]$, therefore $A[1], \ldots, A[k]$ is sorted and they are all smaller than (or equal to) $A[k+1], \ldots, A[n]$. So the invariant holds after this iteration too.

Termination: After the termination of the outer loop, $i = n + 1$, which implies that $A[1], \ldots, A[n]$ are sorted.

d) The number of iterations of the inner loop is $n - i$, each of which takes constant time. So the total running time of the algorithm is proportional to $\sum_{i=1}^{n}(n-i) = \frac{n(n-1)}{2} \in \Theta(n)$. So this algorithm has the same worst case running time as insertion sort but the best case running of insertion sort (which was $O(n)$) is better that of BubbleSort (which is $\Omega(n)$).