CS 173, Spring 2009
Homework 7 Solutions

(Total point value: 30 points.)

1. **Induction with inequalities [10 points]**

   Use induction to show that the following holds for all integers $n \geq 8$.

   $$n^2 > 7n + 1$$

   Solution:

   Base case:

   Note that the base case here occurs when $n = 8$. To see that the given inequality holds for $n = 8$, observe that $8^2 = 64 > 57 = 7 \cdot 8 + 1$.

   Inductive step:

   Assume that $n^2 > 7n + 1$ for some integer n such that $n \geq 8$. Observe,

   $$\begin{aligned}
   (n+1)^2 &= n^2 + 2n + 1 \\
   &> (7n + 1) + 2n + 1 \\
   &= 7n + 2(n + 1) \\
   &> 7n + 16 \\
   &> 7n + 8 \\
   &= 7(n + 1) + 1
   \end{aligned}$$

   where the first inequality above is given by the inductive hypothesis, the second inequality holds because $n + 1 > 8$, and the third inequality is obvious. Thus, we have shown $(n + 1)^2 > 7(n + 1) + 1$, as desired.

2. **Unrolling [10 points]**

   As we have seen (or will soon see) in lecture, it takes $\theta(n \log n)$ time to sort a list of numbers. The usual way to find the median number in a list is to sort the list and then extract the element in the middle of the sorted list. However, there is a somewhat complex alternative algorithm whose running time is (simplifying slightly) given by the recurrence

   $$T(1) = 1$$

   $$T(n) = T(\frac{7}{10}n) + n$$

   (a) Unroll this recurrence (showing your work) and express $T(n)$ as a summation.

   Solution: Let k be the smallest positive integer such that $(7/10)^k \cdot n \leq 1$. Note that k will be the number of steps in the "unrolling" process. It is the number of times we must

apply the recursive definition before getting an argument for T which is small enough to apply the base case to. (See the course website for the correction to the base case for this problem.) We now "unroll" the recurrence as follows:

$$\begin{aligned}
T(n) &= n + T\left(\frac{7}{10}n\right) \\
&= n + \left(\frac{7}{10}\right)\cdot n + T\left(\left(\frac{7}{10}\right)^2 \cdot n\right) \\
&= n + \left(\frac{7}{10}\right)\cdot n + \left(\frac{7}{10}\right)^2 \cdot n + T\left(\left(\frac{7}{10}\right)^3 \cdot n\right) \\
&= \ldots \\
&= n + \left(\frac{7}{10}\right)\cdot n + \left(\frac{7}{10}\right)^2 \cdot n + \ldots + \left(\frac{7}{10}\right)^{k-1} \cdot n + T\left(\left(\frac{7}{10}\right)^k \cdot n\right) \\
&= \left(\sum_{i=0}^{k-1} \left(\frac{7}{10}\right)^i \cdot n\right) + T\left(\left(\frac{7}{10}\right)^k \cdot n\right) \\
&= \left(\sum_{i=0}^{k-1} \left(\frac{7}{10}\right)^i \cdot n\right) + 1
\end{aligned}$$

Note that we obtained the last equality from the base case of the recursion.

We will now pin down more precisely what k needs to be in terms of n. Let $x = \log_{10/7}(n)$, and notice that x satisfies the equation $(7/10)^x \cdot n = 1$. In general, however, x is not an integer but a real number. However, if we choose any integer k larger than x, we will get the inequality we wrote before: $(7/10)^k \cdot n \leq 1$. The least such integer (larger than or equal to x) will be exactly the number of terms of the summation we obtained by "unrolling" the recurrence relation. Thus $k = \lceil x \rceil = \lceil \log_{10/7}(n) \rceil$. Notice that since k depends on n, it may affect our complexity calculation in part c.

(b) In lecture 22, Eric mentioned a couple types of summations whose closed forms you should know. See also the table on p. 157 of the textbook. Find an appropriate formula and convert your answer from part (a) into a closed form.

Solution: We use the first formula from Table 2 on page 157 of Rosen to find a closed form for the summation we obtained above. Using $a = n$ and $r = 7/10$ we see that

$$T(n) = \sum_{i=0}^{k-1} \left(\left(\frac{7}{10}\right)^i \cdot n\right) + 1 = n \cdot \frac{(7/10)^k - 1}{(7/10) - 1} + 1$$

(c) Express $T(n)$ in big-O notation, i.e. using O, Ω and/or θ. (For example, is $T(n) = \theta(n)$? is it $\theta(n \log n)$? is it $\theta(n^2)$?)

Solution: Since k depends on n, we need to do some more simplifications to the closed form we obtained above before we can know the time complexity. Observe

$$\left(\frac{7}{10}\right)^k = \frac{1}{\left(\frac{10}{7}\right)^k} = \frac{1}{\left(\frac{10}{7}\right)^{\lceil \log_{10/7}(n) \rceil}} = \frac{1}{y}$$

where y is a real number greater than or equal to n. (Note that y is still dependent on n.) Now substituting into our expression for $T(n)$ we obtain

$$T(n) = n \cdot \frac{(1/y) - 1}{(7/10) - 1} + 1 = \frac{(n/y) - n}{-3/10} \leq \frac{1 - n}{-3/10} = \frac{n - 1}{3/10} = \frac{10}{3} \cdot n - \frac{10}{3}$$

where the inequality holds since $n/y \leq 1$ (as $y \geq n$). Thus, $T(n) = O(n)$.

(d) How does the big-O running time of this alternate algorithm compare to the $\theta(n \log n)$ time of finding the median by sorting. (Is it larger? smaller? the same?)

Solution: This algorithm is faster, so the running time is smaller.

3. **Algorithm analysis (10 points)**

The following algorithm takes as input an arbitrary list of n real numbers a_1,...,a_n and an integer k. The output is returned in the variable $selected$. The command **swap**(a_i, a_j) is used to interchange the positions of a_i and a_j in the list. You can assume that **swap**(a_i, a_j) takes $\Theta(1)$ time.

```
procedure select( a_1,..., a_n, k)
for i := 1 to k
begin
    min := i
    for j := i + 1 to n
        if a_j < a_min then
        begin
            min := j
            swap(a_i, a_min)
        end
end
selected := a_k
```

An alternative version of the code that correctly performs selection is this:

```
procedure select( a_1,..., a_n, k)
for i := 1 to k
begin
    min := i
    for j := i + 1 to n
        if a_j < a_min then
        begin
            min := j
        end
    swap(a_i, a_min)
end
selected := a_k
```

(a) If the initial list is $10, 5, 2, 8, 3, 0, 15$ with $k = 4$ what value does $selected$ have at the end?

Solution to the alternative version:

When $k = n$, this algorithm is what is known as the **selection sort** algorithm. (See Wikipedia for details on selection sort.) The selection sort algorithm begins by finding the least element in the list and moving it to the front. Next, the least element among the remaining elements is found and put into the second position. This is repeated until the entire list has been sorted.

So when $k < n$, the above algorithm acts like a "partial selection sort". It only sorts the least k elements of the list, and returns $selected$, which has the k^{th} smallest value appearing in the input list. For the given list, $selected$ has the value 5 after the algorithm executes.

Solution to the original version:

For the given list, $selected$ has the value 8 after the algorithm executes.

(b) Suppose each line of the pseudo-code above is a $\Theta(1)$ operation. Suppose also that we are considering "worst-case" running time, so that the conditional code in the 6th line always executes. How many operations are executed by the algorithm for input consisting of a list of n numbers and a specific value k? Express your answer as a function of n and k using big-theta notation and show how you derived that function.

Solution to both versions:

Denote the function giving the number of operations of the algorithm as $T(n, k)$.

Let's first count how many comparisons the algorithm does. During the first iteration of the first **for** loop, the algorithm makes $n - 1$ comparisons. (It compares a_{min} to each of the other $n - 1$ elements of the list. Recall, that we are assuming that the sixth line of code always executes.) During the second iteration, it makes $n - 2$ comparisons. During the third, it makes $n-3$ comparisons, and so on until the outer **for** loop has gone through k iterations. Thus, we have accounted for a total of

$$\begin{aligned}\sum_{i=1}^{k}(n-i) &= (n-1)+(n-2)+\ldots+(n-k)\\ &= n\cdot k-(1+2+\ldots+k)\\ &= n\cdot k-\frac{(k+1)k}{2}\\ &= n\cdot k-\frac{1}{2}k^2-\frac{1}{2}k\end{aligned}$$

total comparisons.

The rest of the operations either happen the same number of times as the comparisons (e.g. incrementing the loop indices) or less often (e.g. the call to swap happens k times) or happen outside both loops (e.g. the return call at the end). We're told that each line of the code is a $\theta(1)$ operation, so it will only contribute some constant amount of running time C to the algorithm. Thus the total number of operations is

$$T(n,k) = C(n\cdot k-\frac{1}{2}k^2-\frac{1}{2}k)+Dk+F$$

where C, D, and F are constants. Since $k \leq n$, the nk-term will always be at least as large as the k^2-term and the k-terms. So we have shown that $T(n, k)$ is $\theta(nk)$.