CS271 Homework 2 Solution

3-1-50

a) This is essentially the same as Algorithm 5 in the book, but working from the other end. However, we can do the moving while we do the searching for the correct insertion spot, so the pseudocode has only one section.

backward insertion sort
for $j := 2$ **to** n
    $m := a_j$
    $i := j - 1$
    while ($m < a_i$ and $i > 0$)
        $a_{i+1} := a_i$
        $i := i - 1$
    end
    $a_{i+1} := m$
end

b) On the first pass the 2 is compared to the 3 and found to be less, so 3 moves to the right. We have reached the beginning of the list and the loop terminates, and the 2 is inserted, yielding 2,3,4,5,1,6. On the second pass the 4 is compared to the 3, and since $4 > 3$, the *while* loop terminates and nothing changes. Similarly, no changes are made as 5 is inserted. On the fourth pass, the 1 is compared to all the way to the front of the list, with each element moving toward the back of the list, and finally the 1 is inserted in its correction position, yielding 1,2,3,4,5,6. The final pass produces no change.
c) Only one comparison is used during each pass, since the condition $m < a_i$ is immediately false, a total of $n - 1$ comparisons are used.
d) The $j - th$ pass requires $j - 1$ comparisons of elements, so the total number of comparison is $1 + 2 + 3 + ... + (n - 1) = n(n - 1)/2$.

3-1-54

a) The algorithm uses the maximum number of quarters, three. Then it used the maximum number of dimes(one) and then two pennies. The greedy algorithm worked.
b) One quarter, two dimes and four pennies. The greedy algorithm worked.
c) The greedy algorithm give use one quarter and eight pennies, a total of nine coins. However, we could have used three dines and three pennies, a total of six coins. Thus the greedy algorithm is not correct in this case.

3-1-58

Here is one counterexample with 11 talks. Suppose the start and end times are as follows: $A : 1-3$, $B : 3-5$,

$C : 5-7$, $D : 7-9$, $E : 2-4$, $F : 2-4$, $G : 2-4$, $H : 4-6$, $J : 6-8$, $K : 6-8$, $L : 6-8$. The optimal schedule is A, B, C and D. However, the talk with fewest overlaps is H, which only overlaps with B and C, However, once we decide to take H, we can schedule at most three talks. So this algorithm will not produce an optimal solution.

3-2-14

a)No. Given a C value, when $x > C$, it is always the case that $x^3 > Cx^2$. Therefore we cannot find the corresponding k value.
b) Yes, since $x^3 \leq x^3$ for all x(witnesses $C = 1$, $k = 0$).
c) Yes, since $x^3 \leq x^2 + x^3$ for all x(witnesses $C = 1$, $k = 0$).
d) Yes, since $x^3 \leq x^2 + x^4$ for all x(witnesses $C = 1$, $k = 0$).
e) Yes, since $x^3 \leq 2^x \leq 3^x$ for all $x > 10$(witnesses $C = 1$, $k = 10$).
f) Yes, since $x^3 \leq 2 \cdot (x^3/2)$ for all x(witnesses $C = 2$, $k = 0$).

3-2-30

a) This follows from the fact that for all $x > 7$, $x \leq 3x + 7 \leq 4x$.
b) For $x > 1000$, clearly $x^2 \leq 2x^2 + x - 7$. On the other hand, for $x \geq 1$, we have $2x^2 + x - 7 \leq 3x^2$.
c) For $x > 2$, we have $\lfloor x + \frac{1}{2} \rfloor \leq 2x$ and also $x \leq \lfloor x + \frac{1}{2} \rfloor$.
d) For $x > 2$, $\log(x^2 + 1) \leq \log(2x^2) = 1 + 2\log x \leq 3\log x$. On the other hand, $\log x \leq \log(x^2 + 1)$.
e) This follows from the fact that $\log_{10} x = \frac{1}{\log_2 10} \cdot \log_2 x$.

3-2-42

This does not follow. Let $f(x) = 2x$ and $g(x) = x$. Then $f(x)$ is $O(g(x))$. Now $2^{f(x)} = 2^{2x} = 4^x$ and $2^{g(x)} = 2^x$. Obviously, 4^x is not $O(2^x)$.

3-3-8

If we successively square k times we get x^{2^k}. Thus we get the result through only k operations.

3-3-12a

There 3 loops, each nested inside the next. The outer loop is executed n times, the middle and inner loops are executed at most n times. Therefore the statement inside the inner loop is executed at most n^3 times. So the total number of comparison is $O(n^3)$.

3-2-38

First we sort the n talks by end time; this takes $O(n \log n)$ time. We initialize a variable *opentime* to be 0; it will be updated whenever we schedule a talk and set to the time at which that talk ends. Then we go through the talks in order. For each talk we compare its start time to *opentime*. If it is later than *opentime* we schedule that talk and update *opentime*. This takes $O(1)$ per talk. Since each talk is visited only once. The total time needed for scheduling is $O(n)$. Combining with the initial sort, we get an overall time complexity of $O(n \log n)$.