Lecture 8 Scratchwork

COT 4400, Fall 2015

September 17, 2015

Input: *data*: an array of integers to sort
Input: n: the number of values in data
Output: permutation of data such that $data[1] \leq \ldots \leq data[n]$

```
Algorithm: BubbleSort
repeat
    for i = 1 to n − 1 do
        if data[i] > data[i + 1] then
            Swap data[i] and data[i + 1]
    end
until the for loop makes no swaps
return data
```

1. Prove that Bubble Sort is correct.

2. Prove a tight upper bound on the worst-case complexity of Bubble Sort.

   Hint: After iteration k of the outer loop in Bubble Sort (line 1), the last k values will be the largest k values in the array, in sorted order.

   Solution:

   The inner loop iterates $n-1 = O(n)$ times, and each iteration costs $O(1)$, for a total time of $O(n)O(1) = O(n)$.

   Claim: the outer loop iterates $O(n)$ times in the worst case.

   Consider an array sorted in decreasing order. By the end of the first iteration, the largest value (data[1]) will be swapped to data[n], and every other value will be shifted to the left. In the second iteration, data[1] will get swapped to data[n-1] and all values in between will be shifted one to the left. This will continue until the $(n-1)^{\text{st}}$ iteration, when data[1] will be swapped into data[2]. On the next iteration, all of the values data[1], . . . , data[n] will be in sorted order, so no swaps will be made, so the outer loop will terminate. Total iterations: n iterations.

   Therefore, the outer loop may iterate $O(n)$ times in the worst case, for a total time of $O(n^2)$. This time dominates the $O(1)$ cost for the return statement, so the entire algorithm will be $O(n^2)$.

3. Prove the multiplicative envelopment property of Big-Omega:

   $j(n)k(n) = \Omega(f(n)g(n))$, where $j(n) = \Omega(f(n))$ and $k(n) = \Omega(g(n))$

   Assumptions: $j(n) = \Omega(f(n))$ and $k(n) = \Omega(g(n))$

   Show: $j(n)k(n) = \Omega(f(n)g(n))$

   Proof. Since $j(n) = \Omega(f(n))$, there exists c_1, n_1 such that $j(n) \geq c_1 f(n)$ for all $n \geq n_1$. Similarly, $k(n) \geq c_2 g(n)$ for all $n \geq n_2$, for some positive constants c_2 and n_2. Since $j(n), k(n), f(n), g(n), c_1$, and c_2 are positive, we can multiply both sides of the inequalities, and we get $j(n)k(n) \geq c_1 c_2 f(n)g(n)$ for all $n \geq n_3$. where $n_3 = \max\{n_1, n_2\}$. Let $c_3 = c_1 c_2$. Therefore, there exist positive constants c_3 and n_3 such that $j(n)k(n) \geq c_3 f(n)g(n)$ for all $n \geq n_3$, so $j(n)k(n) = \Omega(f(n)g(n))$ by the definition of Big-Omega. $\square$

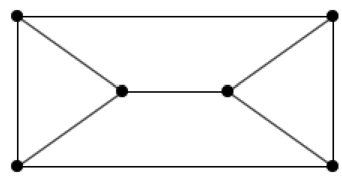
4. Prove that if $g(n) \neq O(1)$, $f(n)g(n) \neq O(f(n))$.

5. Design a greedy algorithm that finds a graph coloring.

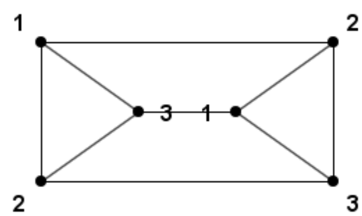
   Problem: Graph coloring

   - **Input:** a graph network G and a coloring number n
   - **Output:** an assignment of colors $(1..n)$ to the nodes of G such that no nodes connected by an edge are the same color, or "no such coloring" if none exists
   - **Example:** $n = 3$

G:



Possible solution:



   Hint: $N(v)$ is the set of neighbors of v; i.e., the vertices joined to v by an edge

6. Identify the worst-case complexity of the PolyEval algorithm (below):

```
Input: d: the degree of the polynomial to evaluate
Input: coeff: the coefficients of the polynomial (largest to smallest)
Output: the value of f(x)
Algorithm: PolyEval
if d = 0 then
    return coeff[1]
else
    Let temp =PolyEval(d − 1,coeff[1..d − 1], x)
    return x ∗ temp+coeff[d]
end
```

Let $T(d)$ represent the number of instructions required to evaluate a polynomial of degree d.

$T(0) = O(1)$

If $d > 0$ and we make a copy of the coeff[1..d-1] array, the copy will take $O(n)$ time, the recursive call will take $T(d-1)$ time, and everything else will take constant time, for a total of $T(d) = T(d-1) + O(n)$.

Characteristic equation: $c(x) = x - 1$. The zero of this equation is $r = 1$. The general form of our solution will be $T(n) = d_1(1^n) + O(n^m f(n)) = O(1^n) + O(n^1)O(n) = O(1) + O(n^2) = O(n^2)$.

If we are using pointers, this is reduced to $T(d) = T(d-1) + O(1)$. Characteristic equation: $c(x) = x - 1$. The zero of this equation is $r = 1$. The general form of our solution will be $T(n) = d_1(1^n) + O(n^m f(n)) = O(1^n) + O(n^1)O(1) = O(1) + O(n) = O(n)$.