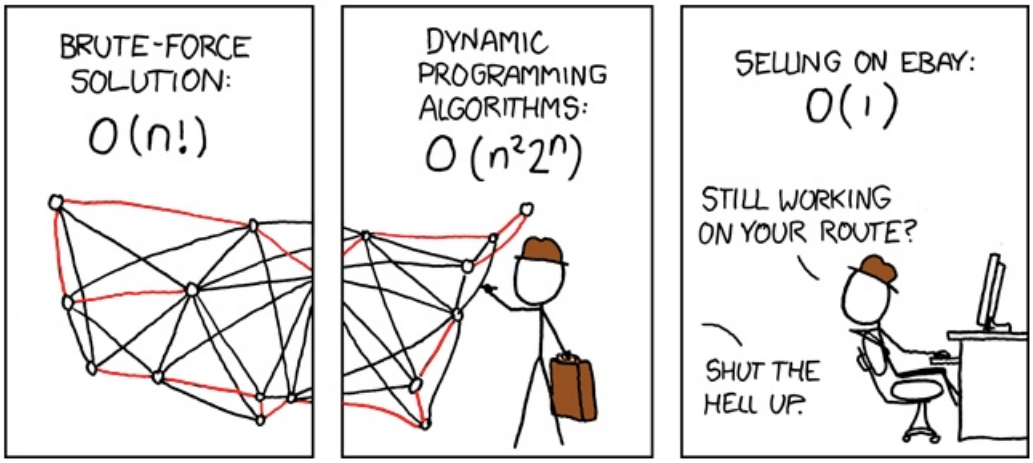
CS140 - Assignment 2

Due: Sunday, Feb. 5 at 8pm



http://xkcd.com/399/

This problem set should be done in the pairs within your learning group. If your group has five members, you may do a group of three.

You must use LaTeX to format your solution; one person should upload the pdf to gradescope.

1. **[10 points]** The Geometric Series

   (a) Use induction on n to show that for all integers $n \geq 0$

   $$1 + a + a^2 + a^3 + \ldots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

   where a is some arbitrary real number other than 1. Please make sure to write your proof carefully, with a base case, induction hypothesis, induction step, and conclusion.

   (b) Explain where your induction proof relied on the fact that $a \neq 1$.

   (c) What does the sum evaluate to when $a = 1$?

2. **[12 points]** Solving Recurrences

   Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible (in other words, give Θ bounds where possible) and make sure to justify your answers. (Note: you may want to read appendix A.1 in the textbook, which has some summation properties.)

(a) $T(n) = 4T(n/2) + cn$

(b) $T(n) = T(n-1) + n$

(c) $T(n) = T(n-1) + 1/n$

(d) $T(n) = T(9n/10) + n$

3. **[3 points]** An improvement?

   You decide to come up with a new sorting algorithm called <*your_name_here*>sort. You notice that *InsertionSort* seems inefficient in how it finds the correct location to insert the next value. You decide that rather than linearly searching one at a time to find the correct place, you use binary search to find the correct place. Is this an improvement? If yes, state the running time of this new algorithm. If no, explain why this is not an improvement. Be specific and clear.

4. **[20 points]** 3-part sort

   Consider the following sorting algorithm: First sort the first two-thirds of the elements in the array. Next sort the last two thirds of the array. Finally, sort the first two thirds again. Notice that this algorithm does not allocate any extra memory; all the sorting is done inside array A. Here's the code:

```
ThreeSort (A,i,j)
    if A[i] > A[j]
      swap A[i] and A[j]
    if i + 1 ≥ j
      return
    k = ⌊(j − i + 1)/3⌋.
    ThreeSort(A, i, j − k)      Comment: Sort first two-thirds.
    ThreeSort(A, i + k, j)      Comment: Sort last two thirds.
    ThreeSort(A, i, j − k)      Comment: Sort first two-thirds again!
```

   (a) Give an informal but convincing explanation (not a rigorous proof by induction) of why the approach of sorting the first two-thirds of the array, then sorting the last two-thirds of the array, and then sorting again the first two-thirds of the array yields a sorted array. A few well-chosen sentences should suffice here.

   (b) Find a recurrence relation for the worst-case running time of ThreeSort.

   (c) Next, solve the recurrence relation using a recursion tree.

   (d) Double check that you got the right answer in the previous part by solving the recurrence using the master method.

   (e) How does the worst-case running time of ThreeSort compare with the worst-case running times of Insertion Sort, Selection Sort, and Mergesort?

5. **[5 points]** Intersection

Describe an algorithm and *state the worse case running time* (in terms of Θ or O where appropriate) to solve the following problem: given two lists of numbers A and B of lengths m and n respectively, return the intersection of the lists, i.e. all those numbers in A that also occur in B. You can use procedures that we've discussed in class, but no others (e.g. no hashtables). You can assume that in any given list, the numbers are unique. You will be graded on the efficiency of your solutions.