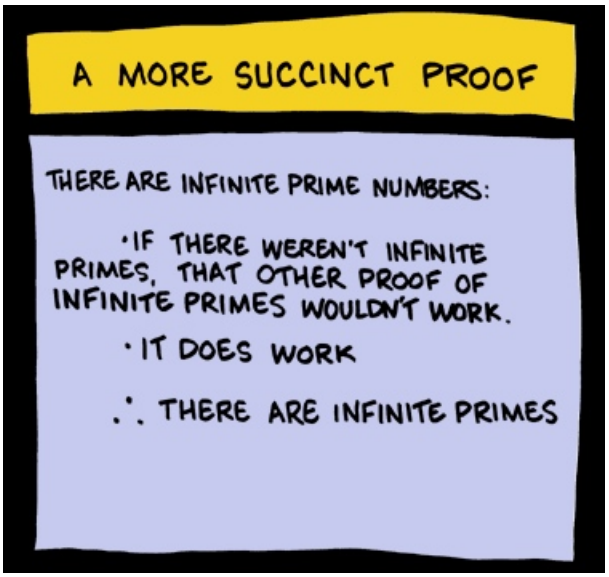
CS140 - Assignment 3

Due: Sunday, Feb. 12 at 8pm



http://www.smbc-comics.com/index.php?db=comics&id=1099

- You must work on this assignment with a partner, though it does not have to be within your learning group. There are an odd number of people in the class right now, so I will allow one group of three, but you need to get permission from me first.
- You must use LaTeX to format your solution.

1. **[4 points]** Give the asymptotic bounds for each of the recurrences below. Assume that $T(n)$ is constant for sufficiently small n. Make your bounds as tight as possible. If you use the master method, you must specify Θ bounds, but only need to specify O if you use another approach.

   (a) $T(n) = 3T(n/2) + n \log n$

   (b) $T(n) = T(n-2) + n$

2. **[5 points]** Algorithm A has a running time described by the recurrence $T(n) = 7T(n/2)+n^2$. A competing algorithm B has a running time described by the recurrence $T(n) = aT(n/4) + n^2$. What is the largest integer value for a such that B is asymptotically faster than A? Explain your answer. (Hint: use the master method.)

3. **[7 points]** Halfsies

   In some situations, there is not a natural ordering to the data so we can't sort it, but we can check equality (e.g. images). Given an array of elements A, we would like to determine if there exists a value that occurs in more than half of the entries of the array. If so, return that value, otherwise, return *null*. Assume you can only check equality of elements in the array which takes time $O(1)$.

   Describe a divide-and-conquer algorithm whose running time is asymptotically better than $\Theta(n^2)$. Provide pseudocode and/or a clear English description of your algorithm and state the worst case running time of your solution (in terms of Θ or O where appropriate) *with justification.*

4. **[10 points]** Given a *sorted* list of unique integers A[1...n], determine if an entry exists such that $A[i] = i$. If an entry exists, return the index, otherwise, return *null*. (Hint: You can do better than $O(n)$. Think divide-and-conquer.) Write **pseudocode** to solve this problem and and *state the worse case running time* (in terms of Θ or O where appropriate). You will be graded on the efficiency of your solutions.

5. **[25 points]** Stock Market Problem

   You're given an array of numbers representing a stock's prices over n days. You goal is to identify the longest consecutive number of days during which the stock's value does not decrease. For example, consider the stock values below:

   ```
   Day:      1   2   3   4   5   6   7   8
   Value:   42  40  45  45  44  43  50  49
   ```

   In this example, the length of the longest consecutive non-decreasing run is 3. This run goes from day 2 to day 4.

   (a) Briefly describe a very simple "naive" algorithm for this problem and explain why the worst-case running time is $\Theta(n^2)$.

   (b) Describe a divide-and-conquer algorithm whose running time is asymptotically better than $\Theta(n^2)$. Provide pseudocode and/or a clear English description of your algorithm. (Note that your algorithm must be a divide-and-conquer algorithm. And yes there are non-divide-and-conquer algorithms that are very, very good!)
   Hint: Like writing recursive functions, when trying to come up with a divide-and-conquer solution, assume that your algorithm works correctly on the divided parts. Then, how do you construct your answer to the overall solution?

   (c) Analyze the running time of your algorithm: what are tight bounds on the best and worst case behavior?

   (Note that next week's assignment will ask you to implement your divide-and-conquer algorithm.)

Extra Credit

[3 points] Halfsies revisited

Here is another algorithm for solving the majority problem above (i.e, "halfsies"):

```
repeat until a candidate majority element is found:
    If there are an odd number of elements, pick one element and see if
    it's the majority.  If it is, mark it as the candidate majority and
    exit the loop.  If not, remove that element and continue to next step.

    Randomly pair up elements.  If they're not equal, throw them both
    away.  If they are equal, keep one of them. repeat.

Finally, check that the candidate majority element is a majority element
in the entire array.  If it is, return it.  If not, return null.
```

1. Give a brief justification (not a proof) for why this works. It doesn't need to be more than a couple of sentences.
2. Write the recurrence for the worst case runtime of this algorithm. Hint: consider two scenarios for the worst case: 1) all elements are identical or 2) n/2 + 1 of the items are identical and the rest are different.
3. What is the worst case runtime of this algorithm?