CS170 - Fall 2011 Problem Set 1 Out: Aug 25, 2011
Due: Sep 2, 2011, 4pm

We will need a photograph from each student taking the class. A passport size photograph (even a xerox copy of you student ID, if clear enough) is OK. Please give to your TA as early as convenient.

Homework will be due in the class homework boxes (283 Soda, second floor) by 4:00pm each Friday (except the first week). Homework will be given out Thursdays.

Please begin your answer to each question on a new sheet of paper, and make sure that each sheet is labeled with your name, section number, GSI name, the assignment number, the question number, and "CS170–Fall 2011".

Turn in each question in a different box in 283 Soda Hall: Question 1 in the box labeled "CS170 - 1", Question 2 in the box labeled "CS170 - 2", etc. Reason: Different problems will be graded in parallel by different readers.

Warning: You risk receiving no credit, or losing points, for any homework that does not conform to the above regulations! Please take the time to write clear and concise solutions; we will not grade messy or unreadable submissions. No late homeworks will be accepted. We will drop the lowest 3 homework scores.

Also, your answers should be written legibly. The descriptions of your algorithms and proofs should be as clear as possible. (This does not mean they should be long, it usually means the opposite.) They should contain enough details to convince a reader that you have all the necessary ideas for a solution. See the class web page for a more detailed description of what we want.

Please read the class web page for rules regarding collaboration (encouraged!) and cheating (forbidden!) on homework.

Note DPV = Dasgupta, Papadimitriou, Vazirani refers to the textbook. So DPV 0.1 refers to Problem 1 in Chapter 0 of DPV.

1. DPV 0.1 . This question gives you pairs of function $f(n)$ and $g(n)$, and asks you to decide if $f = O(g)$, among other things. Here is the level of justification we expect. You should use rules that you should have learned in Ma55 or CS70, namely

   - $f_1 = O(g_1)$ and $f_2 = O(g_2)$ implies $f_1 + f_2 = O(|g_1| + |g_2|)$.
   - $f_1 = O(g_1)$ and $f_2 = O(g_2)$ implies $f_1 * f_2 = O(g_1 * g_2)$.
   - $n^a = O(n^b)$ if $a < b$.
   - $a^n = O(b^n)$ if $a < b$.
   - $\log n = O(n)$ where the log can be in any base $b > 1$.
   - $n = O(b^n)$ for any $b > 1$.
   - if $|f(n)/g(n)|$ approaches a finite nonzero constant as $n \to \infty$, then $f = \Theta(g)$.
   - if $|f(n)/g(n)|$ approaches 0 as $n \to \infty$, then $f = O(g)$ but not $\Omega(g)$.

2. DPV 0.2

3. DPV 0.4

4. DPV 1.4

5. Here is an algorithmic problem that arises in image processing. Let $A[0...n-1, 0...n-1]$ be an n-by-n array (think of it as the values representing an image). For each location (or pixel) (i, j) in the array, we want to compute the sum S of all the A values in a little $(2d+1)$-by-$(2d+1)$ "box" around the pixel; here is the simplest algorithm that does what we want:

```
for i = 0 to n-1
   for j = 0 to n-1
      S[i,j] = 0;
      imin = max(i-d,0)
      imax = min(i+d,n-1)
      jmin = max(j-d,0)
      jmax = min(j+d,n-1)
      for ii = imin to imax
         for jj = jmin to jmax
            S[i,j] = S[i,j] + A[ii,jj]
         end for
      end for
   end for
end for
```

The cost of this algorithm is roughly $O(d^2n^2)$, since for each pixel (i, j) we need to sum $(2d+1)^2$ numbers (fewer near the boundaries).

Derive a new algorithm that lowers this cost to $O(n^2)$, independent of d. Since d is often 10 to 20 in real applications, this is an important speedup.

Hint: It is ok to have another n-by-n array to store intermediate values. So suppose you could quickly compute the array $T[i, j] = \sum_{r=0}^{i} \sum_{s=0}^{j} A[r, s]$, how would you use it to compute S?

6. For those of you disappointed that we are skipping Chapter 1 on number theory, here is one question for you. It also illustrates a useful algorithm that is only "correct with probability > 0". Let $\pi(n)$ be the number of primes less than or equal to n. There is no simple formula for $\pi(n)$, but Lagrange's Prime Number Theorem (sec 1.3.1 of DPV) says that $\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1$, where $\ln n$ is the natural logarithm of n. This theorem means that primes are pretty common, for example the chance that a randomly chosen k-bit number is prime is roughly $1/\ln 2^k \approx 1.44/k$.

   - Use Prime Number Theorem to prove the following fact: given *any* sequence of digits in *any* base, eg 314159265358979323 base 10, there are infinitely many primes that start with these digits, eg when expressed in decimal they start with 314159265358979323.
   - Suppose you want to find a prime number that is exactly n bits long (i.e. n-th bit is 1), where n is large (this is useful in cryptography). Your algorithm is simply "pick a random number that is exactly n bits long, test to see if it is prime, and repeat if it isn't". What is the probability that a random number, that is exactly n bits long, is prime? What is the expected number of iterations of this algorithm before it halts (as a function of n)?
   - Compute the expected number of iterations for $n = 200$ bits.