CS 344 - Sections 1,2,3 - Fall 2018
Homework 1.
Covers lectures on Sep 5, 10, 12
Due September 19.

All problems are out of 10 points. 30 points total plus extra credit.

Note for this HW and all future HW: Unless otherwise specified, you may use any algorithm covered in class as a "black box" – for example you can simply write "sort the array in $\Theta(n\log(n))$ time" without having to describe how to do this.

1 Problem 1 (1 point per part)

For each of the following functions, state whether $f(n) = O(g(n))$ or $f = \Omega(g(n))$, or if both are true, then write $f = \Theta(g(n))$. No proofs required for this problem.

1. $f(n) = n^2 - 7n$ and $g(n) = n^3 - 10n^2$
2. $f(n) = (\sqrt{n})^3$ and $g(n) = n^2 - (\sqrt{n})^3$
3. $f(n) = n^{log_3(4)}$ and $g(n) = n\log^3(n)$
4. $f(n) = 2^{\log_2(n)}$ and $g(n) = n$
5. $f(n) = log^5(n)$ and $g(n) = n/\log(n)$
6. $f(n) = 4^n$ and $g(n) = 5^n$
7. $f(n) = \log_4(n)$ and $g(n) = \log_5(n)$
8. $f(n) = n^3$ and $g(n) = 2^n$
9. $f(n) = \sqrt{n}$ and $g(n) = \log^3(n)$.
10. $f(n) = n\log(n)$ and $g(n) = n^2$

2 Problem 2

- Part 1 (3 points): Prove by induction that $\sum_{i=0}^{k} i2^i = (k-1)2^{k+1} + 2$
- Part 2 (4 points): Prove that $\sum_{i=1}^{n} \frac{i^4}{10} = \Theta(n^5)$
- Part 3 (3 points): What is $\sum_{i=1}^{\log_2(n)} 4^i$ equal to in Θ-notation? (No formal proof necessary, just a brief explanation.)

  HINT: use the formula for geometric sum: $\sum_{i=1}^{k} r^k = (r^k - 1)/(r-1)$. This is generally a very useful formula.

3 Problem 3:

Write an algorithm in pseudocode for each of following two problems below. The algorithm should be simple in both cases! Both algorithms should have running time significantly better than n^2.

Part 1: Closest Pair (5 points)

- Input: An array A with n distinct (non-equal) elements
- Output: numbers x and y in A that minimize $|x - y|$, where $|x - y|$ denotes absolute-value(x-y)

Part 2: Remove Duplicates (5 points)

- Input: An array A of size n, with some elements possibly repeated many times.
- Output: a list L that contains the elements of A (in any order), but without duplicates.

For example, if $A = 1, 3, 7, 5, 3, 5, 3, 1, 4$ then the output set L can contain the numbers $1, 4, 7, 5, 3$ in any order.

4 Problem 4 – EXTRA CREDIT

Given an array A, for any pair of indices $j > i$, define the interval $A[i...j]$ to be the subarray consisting of element $A[i], A[i+1], ..., A[j-1], A[j]$. Now, define sum(i,j) to be the sum of all the values in $A[i...j]$; that is, sum(i,j) = $\sum_{k=i}^{j} A[k]$. Note that it is easy to compute sum(i,j) in time $\Theta(j-i)$ by doing a single pass over $A[i...j]$.

Now, consider the following problem:

Maximum Interval Value

- Input: an array A of length n with positive and negative numbers.
- Output: the maximum possible value sum(i,j)

For example, if A = 3,-5,4,-2,-1,4,-3,2, then the maximum interval value is 5, obtained via the interval [4,-2,-1,4]

Now, consider the following naive algorithm for this problem:
Algorithm:

1. For each index i
2. For each index $j > i$
3. Compute and store sum(i,j) in time $\Theta(j - i)$.
4. Return the maximum value sum(i,j) computed in step 3.

Question: Argue that the total running time of the algorithm is $\Theta(n^3)$. You need to argue that the number of steps is *at most* a constant times n^3, and *at least* a constant times n^3.

HINT: to prove that the running time is $\geq Cn^3$, since it is hard to calculate the sum $\sum_{i,j>i}(j-i)$ for all pairs (i,j), you will want to lower bound the sum by consider only the part of the sum where i is sufficiently small, and j is sufficiently big.