# Problem Set 1

**All parts are due on February 18th, 2016 at 11:59PM**. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

# Part A

**Problem 1-1.** [18 points] **Asymptotic behavior of functions**

For each group of functions, arrange the functions in the group in increasing order of growth. That is, arrange them as a sequence $f_1, f_2, \ldots$ such that $f_1 = O(f_2), f_2 = O(f_3), f_3 = O(f_4), \ldots$. For each group, add a short explanation to explain your ordering.

**(a)** [6 points] **Group 1:**

$$\begin{aligned} f_1 &= 4n^4 \\ f_2 &= \log\left(4n^{n^4}\right) \\ f_3 &= (\log n)^{4\log(4n)} \\ f_4 &= (\log n)^4 \\ f_5 &= (\log\log n)^4 \end{aligned}$$

**(b)** [6 points] **Group 2:**

$$\begin{aligned} f_1 &= 4^{4^n} \\ f_2 &= 4^{4^{n+1}} \\ f_3 &= 5^{4^n} \\ f_4 &= 5^{4n} \\ f_5 &= 4^{5n} \end{aligned}$$

**(c)** [6 points] **Group 3:**

$$f_1 = \binom{n}{4}$$
$$f_2 = \binom{n}{n/4}$$
$$f_3 = 4n!$$
$$f_4 = 4^{n/4}$$
$$f_5 = (n/4)^{n/4}$$

**Problem 1-2.** [18 points] **Recurrences**

**(a)** [12 points] **Solving recurrences:**

Give solutions to the following. In all cases, $c$ is a positive real-valued constant, and $n$ is a nonnegative integer. For simplicity, you may ignore roundoffs in your explanations.

1. $T(n) = c$, for $n \leq 1$, and $T(n) = T(n-1) + c$, for $n > 1$.
2. $T(n) = c$, for $n \leq 1$, and $T(n) = T(n-1) + cn$, for $n > 1$.
3. $T(n) = c$, for $n \leq 1$, and $T(n) = T(n/2) + c$, for $n > 1$.
4. $T(n) = c$, for $n \leq 1$, and $T(n) = 2T(n/2) + c$, for $n > 1$.
5. $T(n) = c$, for $n \leq 1$, and $T(n) = 2T(n/2) + cn$, for $n > 1$.
6. $T(n) = c$, for $n \leq 1$, and $T(n) = 3T(n/2) + cn$, for $n > 1$.

**(b)** [6 points] **Setting up recurrences:** Write recurrences describing the time complexity of the following algorithms. You don't need to solve the recurrences.

1. Suppose we are given a list of integers that are sorted in nondecreasing order. Using binary search, determine whether a given integer appears in the list.
2. Suppose we are given an $n \times n$ matrix, with each row and column sorted in nondecreasing order. For example, the matrix below is sorted in this way:

$$\begin{bmatrix} 1 & 4 & 7 & 9 & 12 & 14 & 23 \\ 2 & 4 & 8 & 10 & 15 & 17 & 26 \\ 6 & 8 & 8 & 11 & 64 & 70 & 80 \\ 12 & 13 & 14 & 15 & 65 & 71 & 80 \\ 14 & 22 & 33 & 34 & 65 & 72 & 80 \\ 25 & 26 & 39 & 48 & 66 & 73 & 80 \\ 35 & 36 & 45 & 55 & 70 & 80 & 81 \end{bmatrix}$$

Determine whether a given integer appears in any matrix that is constructed under these conditions. Solve this problem by *writing and setting up a recurrence*[1].

[1]Note that there are other ways to solve this problem that does not require setting up a recurrence, but we specifically want you to set up a recurrence to solve this problem.

**Problem 1-3.** [24 points] **Maximizing stock gain:**

Here is a new version of the stock gain problem introduced in Lecture 1. Now we are given a finite sequence $A[0, .., n-1]$ of positive integer prices, with $n \geq 4$. The problem is to determine the maximum profit achievable by a sequence of four "Buys" and "Sells" for a particular stock. The four transactions should be "Buy", "Sell", "Buy", and "Sell", in that order, that is, we are buying and selling the same stock twice.[2]

Formally, the problem is to find the maximum value of the expression $(A[i_2] - A[i_1]) + (A[i_4] - A[i_3])$ where $0 \leq i_1 \leq i_2 \leq i_3 \leq i_4 \leq n-1$. Note that you can perform multiple transactions on the same day.

**(a)** [8 points] **Brute-force algorithm:** Describe (in words *and* pseudocode) a naive, brute-force algorithm that takes $A$ as input and outputs the maximum profit. Analyze its runtime cost.

**(b)** [16 points] **Much better algorithm:** Describe (in words *and* pseudocode) an $O(n)$ time algorithm for this problem. Analyze its runtime cost.

[2]This problem assumes that the data is provided ahead of time, and we can choose the best dates with full knowledge of the prices. This is clearly not realistic for buying and selling actual stock, but may be useful for analyzing decisions after the fact.

# Part B

**Problem 1-4.** [40 points] **Document distance**

In this problem, we revisit the "Document Distance" problem and algorithms that were introduced in Recitation 2. You will use some of these algorithms, and some variations, to compare text files for the play "Henry IV, Part I" by William Shakespeare, which he wrote in around 1597, to files for three other English comedies:

1. "Henry IV, Part II", by Shakespeare, written at approximately the same time,
2. "The Tempest", by Shakespeare, written around 1610, and
3. "The Pirates of Penzance", by Gilbert and Sullivan, written around 1879.

Files for these plays appear in the plays folder of the problem set. Code for the routines presented in recitation can be found on Stellar.

**(a)** [10 points] **Comparing word frequencies:** Construct a program for determining the "distance" between two documents, defined to be the angle between two vectors representing the frequencies of occurrence of all words in the two documents. You may use any of the routines provided in recitation, or write your own. Specifically, implement the doc_dist function in docdist.py.

On this problem set, we will not grade you based on the efficiency of your code, as long as it is "reasonable", i.e., it runs within the (generous) bounds we allow. To check whether your code is reasonable, you can run tests.py and check if the profiled output shows that the tests ran under a few seconds.

**(b)** [10 points] **Comparing frequencies of word pairs:** Modify your program so that, instead of comparing word frequencies, it compares the frequency of occurrence of *consecutive pairs of words*. Here, the order of words matters. In other words, "thou art" would be a different word pair than "art thou". Implement the doc_dist_pairs function in docdist.py.

**(c)** [10 points] **Comparing frequences of most frequent words:** Now return to considering word frequencies. For better efficiency, we might consider not vectors that count *all* words, but vectors that count just the 50 words that appear most frequently for each file in the comparison. Break ties alphabetically (using default Python string comparison) on the words. Implement the doc_dist_50 function in docdist.py.

**(d)** [5 points] **Analysis:** Analyze the running time for your file comparison programs in parts (a), (b), and (c) above. As parameters, you should use $n$, the total number of word occurrences in the two files being compared, and $m$, the total number of *distinct* words in the two files. (You may assume that you are given the input files as lists of words, and need not analyze the time for producing those lists.) For part (c), you should also use a parameter $k$ representing the number of frequently-occurring words that are being used in the comparison. If you use Python Dictionaries, consider

insertion and lookup to be $O(1)$-time operations, and listing all the elements in a dictionary (iterating on items or keys) to be $O(s)$, where $s$ is the number of items in the dictionary.

**(e)** [5 points] **Conclusion:** Use your program to determine the angle between your vector representing "Henry IV, Part I" and your vector for each of the other three plays. Repeat this for your functions in parts (a), (b), and (c) above. What can you conclude from these results?