Foundations of Algorithms, Spring 2021: Homework 1

Due: Wednesday, February 10, 11:59pm

- Please see the syllabus and refer to Homework 0 for details on homework requirements.
- In particular note that during implementation you are allowed to use built-in data structures such as Python `list` or Java `LinkedList`. However, if you choose to use a built-in data structure, you must document its performance based on its usage when you evaluate the complexity of your algorithm.
- We'll cover the sorting algorithms soon! Perhaps save Question 5 for last.
- You are not allowed to search on-line for implementations of algorithms or portions of algorithms. You are allowed to reuse code that you wrote yourself in a prior class.
- Java `HashMap` or Python `dict` are never components of a solution to a problem in this class. There is always a different data structure and approach that I am looking for that does not rely on the expected constant-time operations of a hash map.

Problem 1 (10 points)

Rank the following functions by order of growth; that is, find an arrangement $g_1(n), g_2(n), ..., g_{28}(n)$ of functions satisfying $g_i(n) = O(g_{i+1}(n))$ for every $i \in \{1, ..., 27\}$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. Make sure it is clear what your equivalence classes are. You do not have to prove your answers.

$n \log n$	$n^{2/3}$	$n^{3/2}$	$\log n$	$\frac{1}{1000}$	n^2	n^n
$\log_{10} n$	n	2^n	2^{n+1}	$\log \log n$	$n!$	$n + n^2/10^{20}$
2^{2n}	$2^{\log n}$	$\log^2 n$	$\log(n^2)$	$\sqrt{n}$	$\sqrt{\log n}$	$n2^n$
$(n+1)!$	$\log 2^n$	$\frac{1}{n}$	3^n	$n + \log n$	4^n	10^{100}

Remarks:

- In this class we use $\log n$ to denote the logarithm base 2.
- Use Stirling's formula to figure out how to rank $n!$. Stirling's formula is:
$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + O(\frac{1}{n}))$$
- Remember that $\log_2 n$ and 2^n are inverse functions of each other. That is, $2^{\log f(n)} = f(n)$; also $\log 2^{f(n)} = f(n)$.
- Use also this fact: for any constants b_1, $b_2 > 0$:
$$\log^{b_1} n = O(n^{b_2}) \quad \text{and} \quad n^{b_2} \neq O(\log^{b_1} n)$$
In short, logarithm of n raised to any constant power grows slower than any constant power of n.

Problem 2 (8 points)

Suppose you are given asymptotically positive functions f and g. For each of the following statements, either prove that it is true, or give a counterexample demonstrating that it is false:

1. If $f(n) = \Theta(g(n))$ then $2^{f(n)} = \Theta(2^{g(n)})$.
2. If $f(n) = O(g(n))$ then $f(n)^2 = O(g(n)^2)$.
3. If $f(n) = \Theta(g(n))$ then $\frac{1}{f(n)} = \Theta(\frac{1}{g(n)})$.
4. $f(n) = \Theta(f(n-1))$.

Problem 3 (12 points: 8 for implementation / 4 for writeup)

You are given $A[1...n]$, an array containing cumulative snowfall totals for n consecutive days. For example, if there were 6 units of snow on day 1, 7 units of snow on day 2, 0 units of snow on day 3, and 4 units of snow on day 4, the array would contain the values $[6, 13, 13, 17]$. Design an $o(n)$ algorithm that determines if there exists a three-day interval that produced more than half of the total snowfall across the n days. **Note the requirement is $o(n)$, not $O(n)$.** For this problem, you should ignore the cost associated with reading the input. You should also not do any processing of the input as it is read into the array. Design your algorithm assuming the data is already in the array as described above.

Problem 4 (12 points: 8 for implementation / 4 for writeup)

Recall the stable matching problem: you are given two disjoint sets, along with preference lists for each element of each set, ranking all elements of the other set. Design an $O(n^2)$ algorithm that determines if there exist two completely different stable matchings for the given data. Two stable matchings, M_1 and M_2, are considered completely different if they share no common pairings. That is, if pairing $(x, y) \in M_1$, then $(x, y) \notin M_2$.

Problem 5 (15 points: 8 for implementation / 7 for writeup)

Compare several related algorithms. The problem has multiple parts:

- Implement MergeSort, InsertionSort and BucketSort.
- Test the algorithms on random floating point numbers with a uniform distribution in the range $[0, 1)$. Test on input of size $n = 100$, $n = 1000$, $n = 10000$ and $n = 100000$.
- Test the algorithms on random floating point numbers with a Gaussian (normal) distribution with $\mu = 0.5$ and $\sigma = \frac{1}{10000}$. (You should clip these values to stay in the range $[0, 1)$, but in practice this will be unnecessary, as the probability of a value being generated outside this range will be essentially 0. Note that if random variable X has Gaussian distribution with $\mu = 0$ and $\sigma = 1$, then $aX + b$ is a random variable with $\mu = b$ and $\sigma = a$.) Test on input of size $n = 100$, $n = 1000$, $n = 10000$ and $n = 100000$.

- Compare and explain the running times of the three algorithms across all different input sizes and distributions. Do not include the time required to read input or write output in your calculations. If an algorithm takes more than 3 minutes on a particular input, you can terminate the process and just record that the running time is greater than 3 minutes.