Homework 1

Math 182, Winter 2022

Question 1: Big O

(a) Prove that for any functions $f, g\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ and any real number $a > 0$, if $f(n) \in O(g(n))$ then $f(n)^a \in O(g(n)^a)$.

(b) Prove that $\log(n!) \in \Theta(n \log(n))$.
Hint: $\log(\cdot)$ converts products to sums.

(c) Prove that $n^{\log(n)}$ is not in $O(p(n))$ for any polynomial $p(n)$.

Question 2: Algorithm Runtimes

For each algorithm below, find a function $f(n)$ such that the algorithm runs in time $\Theta(f(n))$. You should explain the reasoning behind your answer but you do not need to give a formal proof. For partial credit, you may find a function $f(n)$ so that the algorithm's running time is $O(f(n))$ (i.e. f is an upper bound for the running time, but not necessarily a tight upper bound).

Algorithm 1

```
Foo(n):
  if n > 1:
    Foo(n - 1)
    Foo(n - 1)
```

Algorithm 2

```
Bar(n):
  x = 0
  for i = 1, 2, ..., n:
    x = x + i
  while x > 0:
    x = x - 2
```

Algorithm 3

```
Baz(n):
  while n > 1:
    n = floor(n / 2)
    Baz(n)
```

Hint: If you're stuck on algorithm 3, try explicitly counting the number of steps it takes for small values of n and seeing if you notice a pattern.

Question 3: Algorithm Correctness

Two algorithms are shown below. Each one is supposed to take an array of integers, `A`, and sort it in increasing order. For each algorithm, determine whether or not the algorithm is correct. If it is, prove it. If it is not, give an example of an input on which the algorithm does not work correctly.

Sorting algorithm 1

```
Sort(A):
  for i = 1, 2, ..., n:
    for j = 1, 2, ..., n:
      if A[i] < A[j]:
        swap A[i] and A[j]
```

Sorting algorithm 2

```
Sort(A):
  for i = 1, 2, ..., n:
    for j = 1, 2, ..., n - i:
      if A[j] > A[j + 1]:
        swap A[j] and A[j + 1]
```

Question 4: Matrix Rows

Suppose you are given an $n \times n$ matrix of 0's and 1's in which all the 0's in a row come at the end (in other words, each row consists of a sequence of all 1's followed by a sequence of all 0's). You want to find which row of the matrix has the most 1's (if there are multiple rows tied for the most 1's you may output any row with the maximal number of 1's). Find an $O(n)$ time algorithm to solve this problem.

You may assume that the matrix is given as a 2-dimensional array `A` and that `A[i][j]` will give you the element in the i^{th} row and j^{th} column of `A` in $O(1)$ time.

Question 5: Extra Practice – Matrix of Sums
This problem is extra practice; you do not need to turn it in.

Suppose you are given an array A of n integers. You want to find a $n \times n$ array B in which $B[i][j]$ (for $i < j$) is the sum of array entries $A[i]$ through $A[j]$ inclusive; we don't care what $B[i][j]$ is when $j < i$. That is, for $i < j$, we want $B[i][j] = A[i] + A[i+1] + \cdots + A[j]$.

1. Analyze the runtime of the following algorithm for this problem:

   Matrix of Sums Algorithm

   ```
   MatrixSums(A):
     for i=1,...n:
       for j = i+1, ..., n:
         Add array entries A[i] through [j]
         Store the result in B[i][j]
   ```

   That is, find a function f such that the runtime for this algorithm is $\Theta(f(n))$.

2. This algorithm is the most straightforward approach: for each pair (i, j), it simply computes the relevant entry. It is also quite inefficient. Give an algorithm with asymptotically better runtime. That is, design an algorithm with runtime $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.