Math 182 Homework #1

Nathan Solomon

January 21, 2025

Problem 0.1.

(a) Suppose $a > 0$ and $f(n) \in O(g(n))$. Then there exist positive numbers C, N such that $f(n) \leq Cg(n)$ whenever $n \geq N$. Since a is positive, that means $f(n)^a \leq C^a g(n)^a$ whenever $n \geq N$, and C^a is positive, so $f(n)^a \in O(g(n)^a)$.

(b)

$$\begin{aligned}
\log(n!) &= \log\left(\prod_{k=1}^{n} k\right) \\
&= \sum_{k=1}^{n} \log(k) \\
&\leq \sum_{k=1}^{n} \log(n) \\
&= n\log(n),
\end{aligned}$$

so $\log(n!)$ is also in $O(n\log(n))$. I also need to show that $\log(n!) \in \Omega(n\log(n))$:

$$\begin{aligned}
\log(n!) &= \log\left(\prod_{k=1}^{n} k\right) \\
&= \sum_{k=1}^{n} \log(k) \\
&\geq \sum_{k=\lceil n/2 \rceil}^{n} \log(k) \\
&\geq \sum_{k=\lceil n/2 \rceil}^{n} \log(\lfloor n/2 \rfloor) \\
&= \lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor) \\
&\in \Omega\left(\frac{n}{2}\log\left(\frac{n}{2}\right)\right) \\
&= \Omega\left(n(\log(n) - \log(2))\right) \\
&= \Omega\left(n\log(n)\right).
\end{aligned}$$

Since $\log(n!)$ is in both $O(n\log(n))$ and $\Omega(n\log(n))$, it is in $\Theta(n\log(n))$.

(c) Suppose there is a degree k polynomial $p(n)$ such that $n^{\log(n)} \in O(p(n))$. Then $n^{\log(n)} \in O(n^k)$, which means there is are positive constants C, N such that $n^{\log(n)} \leq Cn^k$ whenever $n \geq N$. This implies $\log(n) \leq k + \log_n(C)$ for all $n \geq N$, and that inequality can be rewritten as $\log(n)^2 \leq k\log(n) + \log(C)$.

However, if $\log(n)$ is increased enough, that inequality will no longer be true, which contradicts my earlier statement that it is true for any $n \geq N$. Therefore $n\log(n) \notin O(p(n))$.

Problem 0.2.

(a) If $n \leq 1$ then Foo(n) takes $f(1) = 1$ step, otherwise it takes $f(n) = 2f(n-1)+1$ steps. By induction, f increases faster than 2^n, which means for large n, the "+1" in that formula becomes relatively insignificant. Therefore $f(n) \in \Theta(2^n)$.

(b) Assigning $1 \to x$ is one step, then the for loop repeats n times. After the for loop, $x = (n^2+n)/2 \in \Theta(n^2)$, so the while loop repeats $\lceil (n^2+n)/4 \rceil$ times. That means the total number of steps is $\Theta(n^2)$.

(c) The function $n \mapsto \lfloor n/2 \rfloor$ is equivalent to the right bitshift-by-one integer operator. The number of times you need to apply this to an integer in order to get to zero is equal to the number of digits in the binary representation of the integer, ignoring leading zeros, which is $\lfloor \log_2(n) \rfloor \in \Theta(\log(n))$.

Problem 0.3.

(a) The only step that changes elements of the array is the swap command, so the integers in A are only being reorganized, not changed. By induction, after running the inner "for" loop, for each $k \leq i$, the kth element is less than or equal to all elements after it. Therefore, after the outer for loop is done, A will be sorted in increasing order.

(b) By induction, after each step of the inner "for" loop, $A[j+1]$ as a maximum element out of $A[1], A[2], \ldots, A[j+1]$. Therefore, after each step of the outer "for" loop, $A[n-i+1]$ will be a maximum element out of $A[1], A[2], \ldots, A[n-i+1]$. Running the inner "for" loop will not change elements $A[n-i+2], A[n-i+3], \ldots, A[n]$, each of those elements will still be larger or equal to each previous element. Therefore, when the outer "for" loop has finished, every element of the array will be larger or equal to each previous element.

Problem 0.4.

Note that I wrote my solution in Python, which is zero-indexed.

```
def bleh(A):
    n = len(A)
    r = 0
    i = 0
    j = 0
    while i < n and j < n:
        if A[i][j] == 0:
            i += 1
        else:
            r = i
            j += 1
    return r
```

Each step of the "while" loop runs in constant time and increases either i or j by one, which means the entire algorithms runs in $\Theta(n)$ time.

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 $\lim_{n\to\infty} g(n)/f(n) = 0$.