TCSS 343 - Week 1

Jake McKenzie

August 5, 2018

Asymptotics

"You're always you, and that don't change, and you're always changing, and there's nothing you can do about it."

...

Neil Gaiman

"Perhaps thinking should be measured not by what you do but by how you do it."

...

Richard Hamming

"You shouldn't try optimizing something that you can't measure."

...

Elecia White

Exact answers are nice when you can find them. Often times we can't or don't care to find them. As computer scientists we care about the behaivor of the algorithms we employ. If we run into a recurrence relation or summation that expressed loosly the behaivor of these algorithms we'd like to know something about their runtime.

Asymptotic Notation in Seven Words
suppress constant factors and lower-order terms

Constant factors are things that will be language and system dependent while lower order terms are irrelevant for large inputs.

Now consider these definitions:
Big-O Notation
If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to 0$ then $f(n) \in O(g(n))$

This is another way of saying that $f(n) \leqslant c \cdot g(n)$ for some position c.
Big-Ω Notation
If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \infty$ then $f(n) \in \Omega(g(n))$

This is another way of saying that $f(n) \geqslant c \cdot g(n)$ for some position c.
Big-Θ Notation
If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to k$ where k is a positive finite number then $f(n) \in \Theta(g(n))$

This is another way of saying that $f(n) \in O(g(n))$ and $f(n) \in \Theta(g(n))$.
Now while we're in this course, we will typically need to show that these relationships are true when prompted, but it's always good to know that the following is true where $0 < \epsilon < 1 < c$.

$$1 < \log\log n < \log n < n^{\epsilon} < n^{c} < n^{logn} < c^{n} < n^{n} < c^{c^{n}}$$

This asymptotic pecking order above is from Don Knuth's Concrete Mathematics.

Now consider the following truths I found in Tim Roughgarden's Algorithms Illuminated:

$$\max\{f(n), g(n)\} \leqslant f(n) + g(n) \wedge 2 \cdot \max\{f(n), g(n)\} \geqslant f(n) + g(n)$$

(reminder that $\wedge$ is the logical "and" symbol.)

1. Prove the following by using the definitions that I've given prior and the information immediately above to show that $\max\{f(n), g(n)\} \in \Theta(f(n)+g(n))$

2. Arrange the following functions in order of increasing growth rate, with $g(n)$ following $f(n)$ in your list if and only if $f(n) \in O(g(n))$. That means using the limit definitions I gave you or by induction. You cannot simply use Don's asymptotic pecking order I stated but I strongly suggest you use the asymptotic pecking order as a guide to the neighborhood of a correct answer.

a) $\sum_{i=0}^{n} 2^i$

b) n^2

c) $n^{0.9999999} \log n$

d) 1.00001^n

e) $4^{\log n}$

f) $100^{\sqrt{n}}$

g) $\log 2^{\frac{n}{2}}$

h) $1000000n$

3. Arrange the following functions in order of increasing growth rate, with $g(n)$ following $f(n)$ in your list if and only if $f(n) \in O(g(n))$. That means using the limit definitions I gave you or by induction. You cannot simply use Don's asymptotic pecking order I stated but I strongly suggest you use the asymptotic pecking order as a guide to the neighborhood of a correct answer.

a) $n^{\frac{5}{3}}$

b) $\sum_{i=0}^{n}(i+1)$

c) $n\sqrt{n}$

d) $\log n^n$

e) $(\frac{n}{\log n})^3$

f) $2^{\frac{n}{2}}$

g) $\log \log n$

h) $n^{1.5}$

4. Using what you know prove or disprove that if $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ then $f(n) \in \Omega(h(n))$.

5. Using what you know prove or disprove that $f(n) + O(f(n)) \in \Theta(f(n))$.

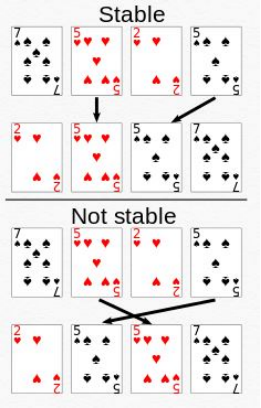
6. Using what you know prove or disprove that $f(n) \in O(f(n)^2)$.

7. Using the technique of splitting the sum, find the result of this summation: $\sum_{k=0}^{\infty} \frac{k^2}{2^k}$. Take advantage of the geometric series $\sum_{k=0}^{\infty} a \cdot r^n = \frac{a}{1-r}$ for $|r| < 1$.

8. What is the code doing and from that information can you make a guess as to the worstcase runtime of this algorithm? (hint: follow what the code is doing by letting $x = 100$ and returning the result. You may use a calculator.)

```
1   //From pages 295-297 of Hacker's Delight by Henry S. Warren Jr
2   int isqrt(unsigned x) {
3       unsigned a, b, m;
4       a = 1;
5       b = (x >> 5) + 8;
6       if (b > 65535) b = 65535;
7       do {
8           m = (a + b) >> 1;
9           if (m*m > x) b = m - 1;
10          else         a = m + 1;
11      } while (b >= a);
12      return a - 1;
13  }
```

9. Argue given the information I provide which of the following in the list of sorts given are stable or unstable. A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.



a) Randomized Quick Sort

b) Bubble Sort

c) Selection Sort

d) Heap Sort

e) Merge Sort

f) Insertion Sort

A. Using what you know prove or disprove that if $O(f(n)O(g(n))) \in O(f(n)g(n))$ then $O(f(n)g(n)) \in f(n)O(g(n))$.

B. Given a number a and a positive integer n, show that the value $a^{2^n} = a^{(2^n)}$ can be computed in $O(n \log n)$ time by repeated squaring. (Assume that multiplying two numbers takes constant time.)

Interview Question: Binary Search

C. Given an already sorted ArrayList of integers write a function in Java that finds the value t from n keys via binary search. Return -1 if t is not in the list otherwise return the index.

D. This problem is concerned with computing all permutations of an array. For example, if the array is $(2, 3, 5, 7)$ one output could be $(2, 3, 5, 7)$, $(2, 3, 7, 5)$, $(2, 5, 3, 7)$, $(2, 5, 7, 3)$, $(2, 7, 3, 5)$, $(2, 7, 5, 3)$, $(3, 2, 5, 7)$, $(3, 2, 7, 5)$, $(3, 5, 2, 7)$, $(3, 5, 7, 2)$, $(3, 7, 2, 5)$, $(3, 7, 5, 2)$, $(5, 2, 3, 7)$, $(5, 2, 7, 3)$, $(5, 3, 2, 7)$, $(5, 3, 7, 2)$, $(5, 7, 2, 3)$, $(5, 7, 2, 3)$, $(7, 2, 3, 5)$, $(7, 2, 5, 3)$, $(7, 3, 2, 5)$, $(7, 3, 5, 2)$, $(7, 5, 2, 3)$, $(7, 5, 3, 2)$. (Any other ordering is acceptable too.)

Write a program which takes as input an array of distinct integers and generates all permutations of that array. No permutation of the array may appear more than once. Hint: How many possible values are there for the first element? Runtime?