TCSS 343 - Week 1

Jake McKenzie

September 18, 2018

Asymptotics with Divide and Conquer

"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

"If one proves the equality of two numbers a and b by showing first that $a \leqslant b$ and then that $b \geqslant a$, it is unfair; one should instead show that they are really equal by disclosing the inner ground for their equality".

. . .

Emmy Noether

"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 $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} \to 0$ 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 $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} \to \infty$ 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 $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} \to k$ 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 $0 < \epsilon < 1 < c$.

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

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 $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$

prove: $\max\{f(n), g(n)\} \leq c(f(n) + g(n))$

Definition of $f(n) \in \Theta(g(n)) := f(n) \in \Omega(g(n)) \wedge f(n) \in O(g(n))$

$\max\{f(n), g(n)\} \leq f(n) + g(n)$ · is given

if $2\max\{f(n), g(n)\} \geq f(n) + g(n)$

then $\max\{f(n), g(n)\} \geq \frac{1}{2}(f(n) + g(n))$

(multiply by $\frac{1}{2}$ on both sides

By the definition of big theta notation
we may say that $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$

3

$c < g =_p h < b =_p e < f < d < a$

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) $\sum_{i=0}^{n} 2^i = \dfrac{2^{n+1}-1}{2-1} = 2^{n+1} - 1 = 2(2^n) - 1$

b) n^2

∞ L'Hopitals

$\lim_{n \to \infty} \dfrac{2(2^n)-1}{1.00001^n} = \lim_{n \to \infty} 2\left(\dfrac{2}{1.00001}\right)^n - \dfrac{1}{1.00001^n}$

c) $n^{0.9999999} \log n$ $\underbrace{\qquad}_{V small}$

$\boxed{a > d}$

$\boxed{b =_p e}$

d) 1.00001^n

e) $4^{\log n} = (2^2)^{\log n} = 2^{\log n^2} = n^2$

f) $100^{\sqrt{n}} = 100^{n^{\frac{1}{2}}}$

$\lim_{n \to \infty} \dfrac{2(2^n)-1}{100^{n^{\frac{1}{2}}}} = \lim_{n \to \infty} \dfrac{2 \cdot 2^n}{100^{n^{\frac{1}{2}}}} - \dfrac{1}{100^{n^{\frac{1}{2}}}}$

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

$\boxed{a > f}$

h) $1000000n$

$\boxed{g =_p h}$

$\lim_{n \to \infty} \dfrac{\frac{n}{2}}{10^6 n} \to \boxed{\dfrac{1}{2 \cdot 10^6}} \quad \lim_{n \to \infty}$

$\to \infty$

$\lim_{n \to \infty} \dfrac{\frac{n}{2}}{n^{0.9999} \log n} = \dfrac{1}{2} \lim_{n \to \infty} \dfrac{\frac{d}{dn} n^{0.1111}}{\log n} = \dfrac{1}{2} \dfrac{0.111 n^{-0.111}}{\frac{1}{n}}$

$\lim_{n \to \infty} \dfrac{n^2}{n^{0.99999} \log n} = \dfrac{n^{1.1111}}{\log n} \lim_{n \to \infty} \to \infty$

$\boxed{g > c}$

$\boxed{f < b}$

4

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) $\sum_{i=0}^{n}(i+1)$ $\dfrac{n(n+1)}{2} + (n-0+1)1 = \dfrac{n^2+n}{2}\,\dfrac{+n}{2} + n + 1 = \left(\dfrac{n^2}{2}\right) \dfrac{3n}{2} + 1$

c) $n\sqrt{n} = n^{\frac{3}{2}}$

d) $\log n^n = n \log n$

e) $\left(\frac{n}{\log n}\right)^3$ $\dfrac{n^3}{(\log n)^3}$ $\lim_{n \to \infty} \dfrac{(\log n)^3}{n^{\frac{5}{3}}}$ $= \lim_{n \to \infty} \dfrac{n^{\frac{4}{3}}}{(\log n)^3} \to \infty$

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

g) $\log \log n$ smallest

h) $n^{1.5}$

$g < d < c =_p h < a < b < e < f$

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

let $f(n) = 2n$

$\quad g(n) = 3n$.

$f(n) \leq c_1(g(n)) \Rightarrow 2n \leq c_1(3n)$

$g(n) \leq c_2(g(n)) \Rightarrow 3n \leq c_2(2n)$

True only for $c_1 = \frac{2}{3}$

$\quad \land c_2 = \frac{3}{2}$

$\quad \forall n$.

if $c_1 = 3$ and $c_2 = 1$

The statement is false

6

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

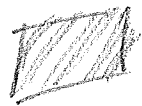$$f(n) + O(f(n)) \leq c_1 f(n)$$

$\uparrow$

upper bound of

$f(n)$

$\therefore$

$f(n) + O(f(n))$ must

be atmost

$$c_2 O(f(n)) + O(f(n)) \leq c_1 f(n)$$

$$(c_2 + 1) O(f(n)) \leq c_1 f(n)$$

$\uparrow$

$\therefore$ Fundametal Theorem of Algebra!

$$c_1 = c_2 + 1 \quad \forall n \in \mathbb{N}$$

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

$$\text{let } f(n) = \frac{1}{n} \qquad \frac{1}{n} \notin O\left(\frac{1}{n^2}\right)$$

and

This cannot be true

for $f(n) \in O(f(n)^2)$

8

Let $P(n): \sum_{k=1}^{n} \frac{k}{2^k} = 2 - 2^{1-n} - n2^{-n}$

7. Prove by induction that: $\sum_{k=1}^{n} \frac{k}{2^k} = 2 - 2^{1-n} - n2^{-n}$.

BC

$P(1): \sum_{k=1}^{1} \frac{k}{2^k} = \frac{1}{2^1} = \frac{1}{2}$ ✓

$P(1): 2 - 2^{1-1} - (1)2^{-1} = 2 - 1 - \frac{1}{2} = \frac{1}{2}$ ✓

IH

$P(\psi): \sum_{k=1}^{\psi} \frac{k}{2^k} = 2 - 2^{1-\psi} - \psi 2^{-\psi}$

IS

$P(\psi+1): \sum_{k=1}^{\psi+1} \frac{k}{2^k} = 2 - 2^{1-\psi-1} - (\psi+1)2^{-\psi-1}$

LHS

RHS

$2 - 2^{-\psi} - (\psi+1)2^{-\psi-1}$

$2 - (2 + \psi + 1)2^{-\psi-1}$

$\boxed{2 - (\psi+3)2^{-\psi-1}}$ ✓

LHS

$\sum_{k=1}^{\psi} \frac{k}{2^k} + \frac{\psi+1}{2^{\psi+1}}$

$2 - 2^{1-\psi} - \psi 2^{-\psi} + (\psi+1)2^{-\psi-1}$

$2 - (4)2^{-\psi-1} - \psi 2^{-\psi-1} + (\psi+1)2^{-\psi-1}$

$2 - (4 + 2\psi - \psi - 1)2^{-\psi-1}$ ✓

$\boxed{2 - (\psi+3)2^{-\psi-1}}$

9

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   }
```

$7b = \dfrac{x}{25} + 2^3 = 2^3$

$m > \sqrt{x}$

This returns back the "integer squareroot"

In the worst case it has $O(\log_2 n)$ runtime.

Note: In the avg case this has $O(\log \log n)$ Runtime! This is very, very good!

① $m = (1 + 8)/2 = 4$

② $4 \times 4 > 100$ not true

③ $a = 5$

④ $m = (5 + 8)/2 = 6$

⑤ $6 \times 6 > 100$ not true

⑥ $a = 6$

⑦ $m = (6 + 8)/2 = 7$

⋮

return $11 - 1$

10

In practive we would say this has a $\log \log n)$ runtime which degrades to $O(\log n)$

So about that integer square root function on the previous page. If you said $O(\log n)$ good job, you were really close. It actually has $O(\log \log n)$ worst case runtime but degrades to $O(\log n)$ given a bad initial guess. The choice of an initial guess is a deep and rich subject in it's own. There's a very famous algorithm, first discovered at Silicon Graphics in the 80's but popularized in the early aughts by a programmer on Usenet who worked at id Software, the creators of Quake.

```
float InvSqrt(float x)
    {
    float xhalf = 0.5f*x;
    int i = *(int*)&x;        // get bits for floating value
    i = 0x5f3759df - (i>>1);  // gives initial guess y0
    x = *(float*)&i;          // convert bits back to float
    x = x*(1.5f-xhalf*x*x);   // Newton step, repeating increases accuracy
    return x;
    }
```


Do you see that 0x5f3759df? That's known as the "magic number" and offers a better than good initial guess for just about any number. The most absolute error fast inverse square root gives is 0.175%. This a deeply cherished algorithm and beloved number by anyone in computer graphics, but sadly hasn't gotten studied too much due to it being an insight found by industry professionals who desperately needed it.

I've actually been implementing my own square root algorithm in a HDL (hardware description language) and I encourage you to make your own personal projects. It need not be for anyone elses benefit or it very much may just be for others benefit!

John Carmack

"In the information age, the barriers to entry into programming just aren't there. The barriers are self imposed. If you want to set off and go develop some grand new thing, you don't need millions of dollars of capitalization. You need enough pizza and Diet Coke to stick in your refrigerator, a cheap PC to work on, and the dedication to go through with it. We slept on floors. We waded across rivers."

If you were a bit confused and overwhelmed what I showed you in 8, please take a look at these. I wanted you to be able to look at some code and analyze it before presenting information as an oracle. Here is a simpler illustration to how the simple divide and conquer approach work. I think you should sit down and really try to understand why this example works as it does. It will help you going forward.
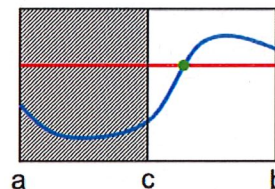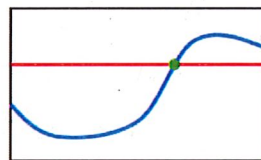
Divide & Conquer (Binary Search)

Bisection (a, b, ε)

    if $(b - a) < \varepsilon$ then

        return a;

    else

        $m \leftarrow (a + b)/2$;

        if $f(m) \leq 0$ then

            return Bisection(c, b, ε);

        else

            return Bisection(a, c, ε);
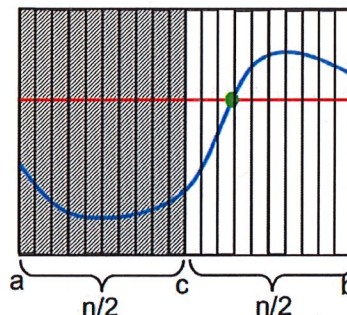
Time Analysis

Let $n = \frac{a-b}{\epsilon}$ be the # of intervals and $c = (a + b)/2$

Always half of the intervals lie to the left and half lie to the right of c
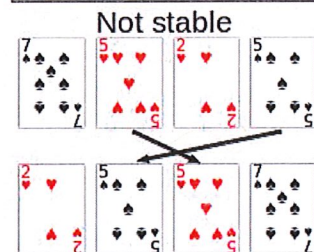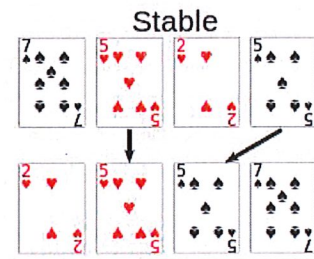
So,

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

i.e., $T(n) = O(\log n) = O(\log(\frac{a-b}{\epsilon}))$

generally speaking most $O(n^2)$ sorts are <u>stable!</u>

A. Argue given what you know 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.

Stable

Not stable

comparator

$\{-1-\}$
↑
value

a) Randomized Quick Sort ← *unstable!*

stable! → b) Bubble Sort

c) Selection Sort *unstable!*

Not stable → d) Heap Sort

most are stable! e) Merge Sort

stable! → f) Insertion Sort

Consider the following list

unsorted $\{\{5, "a"\}, \{0, "b"\}, \{1, "c"\}, \{2, "d"\}, \{5, "e"\}\}$

randomized quicksort sorted $\{\{0, "b"\}, \{1, "c"\}, \{2, "d"\}, \{5, "e"\}, \{5, "a"\}\}$ *unstable*

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

<u>Note:</u> if $O(f(n)O(g(n))) \in O(f(n)g(n))$

then $O(f(n)g(n)) \in O(f(n)O(g(n)))$

<u>Another fact!</u> $O(O(f(n))) \in O(f(n))$

↓ *allows us to say*

$O(f(n)g(n)) \in O(f(n)O(O(g(n))))$

$O(f(n)) \in O(f(n))$ ∴

$$\boxed{O(f(n)g(n)) \in f(n)O(g(n))}$$

13

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.

See attached!

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?

See Attached!