Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Asymptotic Notation Intro

In this class, we care a lot about the runtime of algorithms. However, we don't care too much about concrete performance on small input sizes (most algorithms do well on small inputs). Instead we want to compare the *long-term (asymptotic)* growth of the runtimes.

Asymptotic Notation: We define the following notation for two functions $f(n), g(n) \geq 0$:

- $f(n) = \mathcal{O}(g(n))$ if there exists a $c > 0$ where after large enough n, $f(n) \leq c \cdot g(n)$. *(Asymptotically, f grows at most as much as g)*
- $f(n) = \Omega(g(n))$ if $g(n) = \mathcal{O}(f(n))$. *(Asymptotically, f grows at least as much as g)*
- $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$. *(Asymptotically, f and g grow the same)*

If we compare this to the order on the numbers, $\mathcal{O}$ is a lot like $\leq$, Ω is a lot like $\geq$, and Θ is a lot like $=$ (except all are with regard to asymptotic behavior).

For each of the following, state the order of growth using Θ notation, e.g. $f(n) = \Theta(n)$.

(i) $f(n) = 50$

Solution: $f(n) = \Theta(1)$

(ii) $f(n) = n^2 - 2n + 3$

Solution: $f(n) = \Theta(n^2)$

(iii) $f(n) = n + \cdots + 3 + 2 + 1$

Solution: $f(n) = \frac{n(n+1)}{2} = \Theta(n^2)$

(iv) $f(n) = n^{100} + 1.01^n$

Solution: $f(n) = \Theta(1.01^n)$

(v) $f(n) = n^{1.1} + n \log n$

Solution: $n^{1.1}$ grows more than $n \log n$, so $f(n) = \Theta(n^{1.1})$.

Formally, we can notice

$$\lim_{n\to\infty} \frac{n^{1.1}}{n \log n} = \lim_{n\to\infty} \frac{n^{0.1}}{\log n} = \lim_{n\to\infty} \frac{0.1n^{-0.9}}{\frac{1}{n}} = \lim_{n\to\infty} 0.1n^{0.1} = \infty$$

(vi) $f(n) = (g(n))^2$ where $g(n) = \sqrt{n} + 5$

Solution:

$$f(n) = (\sqrt{n} + 5)^2 = n + 10\sqrt{n} + 25$$

$$f(n) = \Theta(n)$$

Solution: In general, we can observe the following properties of $\mathcal{O}/\Theta/\Omega$ from this:

- If $d > c$, $n^c = \mathcal{O}(n^d)$, but $n^c \neq \Omega(n^d)$ (this is sort of saying that n^d grows strictly more than n^c).
- Asymptotic notation only cares about "highest-growing" terms. For example, $n^2 + n = \Omega(n^2)$.
- Asymptotic notation does not care about leading constants. For example $50n = \Theta(n)$.
- Any exponential with base > 1 grows more than any polynomial
- The base of the exponential matters. For example, $3^n = \mathcal{O}(4^n)$, but $3^n \neq \Omega(4^n)$.
- If $d > c$, $n^c \log n = \mathcal{O}(n^d)$.

2 Asymptotics and Limits

If we would like to prove asymptotic relations instead of just using them, we can use limits.

Asymptotic Limit Rules: If $f(n), g(n) \geq 0$:

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = \mathcal{O}(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, for some $c > 0$, then $f(n) = \Theta(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, then $f(n) = \Omega(g(n))$.

Note that these are all sufficient conditions involving limits, and are not true definitions of $\mathcal{O}$, Θ, and Ω.

(a) Prove that $n^3 = \mathcal{O}(n^4)$.

Solution:

$$\lim_{n\to\infty} \frac{n^3}{n^4} = \lim_{n\to\infty} \frac{1}{n} = 0$$

So $f(n) = \mathcal{O}(g(n))$

(b) Find an $f(n), g(n) \geq 0$ such that $f(n) = \mathcal{O}(g(n))$, yet $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$.

Solution: Let $f(n) = 3n$ and $g(n) = 5n$. Then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{3}{5}$, meaning that $f(n) = \Theta(g(n))$. However, it's still true in this case that $f(n) = \mathcal{O}(g(n))$ (just by the definition of Θ).

(c) Prove that for any $c > 0$, we have $\log n = \mathcal{O}(n^c)$.

Hint: Use L'Hôpital's rule: If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$, then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$ (if the RHS exists)

Solution: By L'Hôpital's rule,

$$\lim_{n\to\infty} \frac{\log n}{n^c} = \lim_{n\to\infty} \frac{n^{-1}}{cn^{c-1}} = \lim_{n\to\infty} \frac{1}{cn^c} = 0$$

Therefore, $\log n = \mathcal{O}(n^c)$.

(d) Find an $f(n), g(n) \geq 0$ such that $f(n) = \mathcal{O}(g(n))$, yet $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ does not exist. In this case, you would be unable to use limits to prove $f(n) = \mathcal{O}(g(n))$.

Solution: Let $f(x) = x(\sin x + 1)$ and $g(x) = x$. As $\sin x + 1 \leq 2$, we have that $f(x) \leq 2 \cdot g(x)$ for $x \geq 0$, so $f(x) = \mathcal{O}(g(x))$.

However, if we attempt to evaluate the limit, $\lim_{x\to\infty} \frac{x(\sin x+1)}{x} = \lim_{x\to\infty} \sin x + 1$, which does not exist (sin oscillates forever).

3 Runtime and Correctness of Mergesort

In general, this class is about design, correctness, and performance of algorithms. Consider the following algorithm called *Mergesort*, which you should hopefully recognize from 61B:

```
function MERGE(A[1,...,n], B[1,...,m])
    i, j ← 1
    C ← empty array of length n + m
    while i ≤ n or j ≤ m do
        if i ≤ n and (j > m or A[i] < B[j]) then
            C[i + j − 1] ← A[i]
            i ← i + 1
        else
            C[i + j − 1] ← B[j]
            j ← j + 1
    return C
function MERGESORT(A[1,...,n])
    if n ≤ 1 then return A
    mid ← ⌊n/2⌋
    L ← MERGESORT(A[1,...,mid])
    R ← MERGESORT(A[mid + 1,...,n])
    return MERGE(L, R)
```

Recall that MERGESORT takes an arbitrary array and returns a sorted copy of that array. It turns out that MERGESORT is asymptotically optimal at performing this task (however, other sorts like Quicksort are often used in practice).

(a) Let $T(n)$ represent the number of operations MERGESORT performs given a array of length n. Find a base case and recurrence for $T(n)$, use asymptotic notation. Solve the recurrence using the Master theorem.

Solution:

For the base case, we simply have $T(1) = 1$ (almost nothing is done). For the recursive case $n \geq 2$, we get $T(n) = 2T(n/2) + \mathcal{O}(n)$.

On a high level, MERGE takes two pointers through A and B respectively, and keeps adding the next-smallest element from A or B. We notice that MERGE looks at each element from A or B only once. So given arrays of length m and n, MERGE takes $\mathcal{O}(m+n)$. Note that the $\mathcal{O}(n+m)$ is important here, the time is not just $n+m$ (there's more than one array access for each element in A or B, for example).

A call to MERGESORT involves two recursive calls to pieces of size $n/2$ and one call to MERGE given both halves, so the runtime here is

$$T(n) = \underbrace{2T(n/2)}_{\text{two recursive calls to MERGESORT}} + \underbrace{\mathcal{O}(n)}_{\text{call to MERGE}}$$

Since $\log_2(2) = 1$, the Master theorem gives us $T(n) = \Theta(n \log n)$.

(b) Consider the correctness of MERGE. What is the desired property of C once MERGE completes? What is required of the arguments to MERGE for this to happen?

Solution:

We desire C to be sorted, and to contain all elements from A and B. MERGE requires that A and B are individually sorted.

Question: This falls short of a full proof of correctness for MERGE. How would you fully proof correctness for merge? (*Hint:* a good place to start would be to think about 'invariants', or properties that hold for C as i and j increase)

(c) Using the property you found for MERGE, show that MERGESORT is correct.

Solution:

We establish by induction. Let

$P(n)$: MERGESORT$(A[1, \ldots, n])$ is correct for any array A of length n

For the base case, $P(1)$ is true because a length-1 array is already sorted, and that is what we return.

Now assume for a particular n that $P(k)$ holds for all $k < n$ (this is strong induction). L and R are then guaranteed to be sorted, then by part (b) their merge will be sorted. As L and R contain all the elements of the array, we return a sorted copy of the array.

4 Master Theorem

For solving recurrence relations asymptotically, it often helps to use the *Master Theorem*:

Master Theorem. If $T(n) = aT(n/b) + \mathcal{O}(n^d)$ for $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} \mathcal{O}(n^d) & \text{if } d > \log_b a \\ \mathcal{O}(n^d log\ n) & \text{if } d = \log_b a \\ \mathcal{O}(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

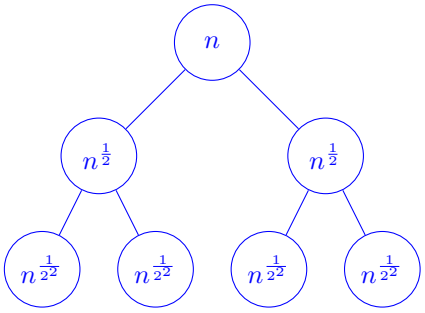
Note: You can replace $\mathcal{O}$ with Θ and you get an alternate (but still true) version of the Master Theorem that produces Θ bounds.

$d_{crit} = \log_b a$ is called the *critical exponent*. Notice that whichever of d_{crit} and d is greater determines the growth of $T(n)$, except in the case where they are perfectly balanced.
Solution: As all things should be.
Solve the following recurrence relations and give a $\mathcal{O}$ bound for each of them.

(a) (i) $T(n) = 3T(n/4) + 4n$
Solution: Since $\log_4 3 < 1$, by the Master Theorem, $T(n) = \mathcal{O}(n)$.

(ii) $T(n) = 45T(n/3) + .1n^3$
Solution: Since $\log_3 45 > 3$, by the Master Theorem, $T(n) = \mathcal{O}(n^{\log_3 45})$.

(b) $T(n) = 2T(\sqrt{n}) + 3$, and $T(2) = 3$.
Hint: Try repeatedly expanding the recurrence.
Solution:
Draw out the tree, and compute the total work done by summing over the work done at every level.



Notice how the size of the subproblem at level i is $n^{\frac{1}{2^i}}$. We stop when our problem size is 2. Thus to find the height of the tree, we need $n^{1/2^k} = 2$. So

$$\begin{aligned} 1 &= \frac{1}{2^k} \log n \\ 2^k &= \log n \\ k &= \log \log n \end{aligned}$$

From this we can construct a summation of all work done over all the levels: $\sum_{i=0}^{\log \log n} 2^i * 3$. This is the geometric sum, which equals $3(2^{\log \log n + 1} - 1)$. Therefore, we get our answer to be $\mathcal{O}(\log n)$.

Another way to do this is to try repeated substitution.

$$\begin{aligned}
&T(n)\\
&T(n) = 2T(n^{1/2}) + 3\\
&T(n) = 4T(n^{1/4}) + 2\cdot 3 + 3\\
&T(n) = 8T(n^{1/8}) + 2^2\cdot 3 + 2\cdot 3 + 3\\
&\quad\vdots\\
&T(n) = 2^k T(n^{1/2^k}) + 3\cdot\sum_{i=0}^{k-1} 2^i \qquad (*)
\end{aligned}$$

To get the base case, we need $n^{1/2^k} = 2$. So

$$\begin{aligned}
1 &= \frac{1}{2^k}\log n\\
2^k &= \log n\\
k &= \log\log n
\end{aligned}$$

Plugging in to (*) gives

$$T(n) = 2^{\log\log n}T(2) + 3\cdot\sum_{i=0}^{\log\log n - 1} 2^i = 3\log n + 3\mathcal{O}(2^{\log\log n}) = \mathcal{O}(\log n)$$

where we have used the fact that $\sum_{i=0}^{n} 2^i = \mathcal{O}(2^n)$.

Note: It is also possible to solve this problem by doing a change of variables:

A priori, this problem does not immediately look like it satisfies the Master Theorem because the the recurrence relation is not a map $n \to n/b$. However, we notice that we may be able to *transform* the recurrence relation into one about another variable such that it satisfies the Master Theorem. Let k be the solution to

$$n = 2^k.$$

Then we can rewrite our recurrence relation as

$$T(2^k) = 2T(2^{k/2}) + 3.$$

Now, if we let $S(k) = T(2^k)$, then the recurrence relation becomes something more manageable:

$$S(k) = 2S(k/2) + 3.$$

By the Master Theorem, this has a solution of $\mathcal{O}(k)$. Since $n = 2^k$, then $k = \log n$ and therefore $\mathcal{O}(k) = \mathcal{O}(\log n)$.

The intuition between the transformation between n and k is that n could be a number and k, the number of bits required to represent n. The recurrence relation is easier expressed in terms of the number of bits instead of the actual numbers.

(c) Consider the recurrence relation $T(n) = 2T(n/2) + n\log n$. We can't plug it directly into the Master Theorem, so solve it by adding the size of each layer.

Hint: split up the $\log(n/(2^i))$ *terms into* $\log n - \log(2^i)$*, and use the formula for arithmetic series.*

Solution:

The amount of work done at the ith layer is $2^i(n/2^i \log(n/2^i))$. Since there are a total of $\log_2 n$ layers, we can find the total work done as follows:

$$\begin{aligned}
\sum_{i=1}^{\log n} 2^i(n/2^i \log(n/2^i)) &= \sum_{i=1}^{\log n} n \log(n/2^i) \\
&= n \sum_{i=1}^{\log n} \left(\log n - \log 2^i\right) = n \log^2 n - n \sum_{i=1}^{\log n} i(\cancelto{1}{\log 2}) \\
&= n \log^2 n - n \times \frac{(\log n)(\log n + 1)}{2} \\
&= n \log^2 n \left(1 - \frac{1}{2}\right) - n \log n \times \frac{1}{2} \\
&= \Theta(n \log^2 n)
\end{aligned}$$

where step (4) uses the formula for arithmetic series, and step (6) discards the constant factors. It then discards the second term because its log factor, $(\log n)$, is dominated by $(\log^2 n)$ in the first term.

Note: in this class, if not explicitly stated we assume that the base for log is 2.

5 Complex numbers review

A *complex number* is a number that can be written in the rectangular form $a + bi$ (i is the imaginary unit, with $i^2 = -1$). The following famous equation (*Euler's formula*) relates the polar form of complex numbers to the rectangular form:

$$re^{i\theta} = r(\cos\theta + i\sin\theta)$$

In polar form, $r \geq 0$ represents the distance of the complex number from 0, and θ represents its angle. The n *roots of unity* are the n complex numbers satisfying $\omega^n = 1$. They are given by

$$\omega_k = e^{2\pi ik/n}, \qquad k = 0, 1, 2, \ldots, n-1$$

(a) Let $x = e^{2\pi i 3/10}, y = e^{2\pi i 5/10}$ which are two 10-th roots of unity. Compute the product $x \cdot y$. Is this a root of unity? Is it an 10-th root of unity?

What happens if $x = e^{2\pi i 6/10}, y = e^{2\pi i 7/10}$?

Solution: $x \cdot y = e^{2\pi i 8/10}$. This is always an 10-th root of unity (it is in general). But because $8/10 = 4/5$, this is also a 5th root of unity.

If $x = e^{2\pi i 6/10}, y = e^{2\pi i 7/10}$, then we 'wind around' and the product becomes $e^{2\pi i 13/10} = e^{2\pi i 3/10}$.

(b) Show that for any n-th root of unity ω, $\sum_{k=0}^{n-1} \omega^k = 0$, when $n > 1$.

Hint: Use the formula for the sum of a geometric series $\sum_{k=0}^{n} \alpha^k = \frac{\alpha^{n+1}-1}{\alpha-1}$. It works for complex numbers too!

Solution: Remember that $\omega^n = 1$. So
$\sum_{k=0}^{n-1} \omega^k = \frac{\omega^n - 1}{\omega - 1} = \frac{1-1}{\omega-1} = 0$

(c) (i) Find all ω such that $\omega^2 = -1$.

Solution: Notice that $(e^{2\pi i\theta})^n = e^{2\pi in\theta}$.

As $-1 = e^{2\pi i(1/2)}$ (Euler's identity!), we are looking for all θ such that $2\theta = \frac{1}{2} + k$ for some integer k.

Setting $k = 0$ gives $\theta = \frac{1}{4}$. Setting $k = 1$ gives $\theta = \frac{3}{4}$. Any other values of k will just repeat these values of θ.

Notice that in general, the square roots of $e^{2\pi i\theta}$ are $e^{2\pi i\theta/2}, e^{2\pi i(\theta+1)/2}$.

(ii) Find all ω such that $\omega^4 = -1$.

Solution: These ω are simply the square roots of the ω we found before. The square roots of $e^{2\pi i 1/4}$ are $e^{2\pi i 1/8}, e^{2\pi i 5/8}$.

The square roots of $e^{2\pi i 3/4}$ are $e^{2\pi i 3/8}, e^{2\pi i 7/8}$, so we are finished.