Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 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

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

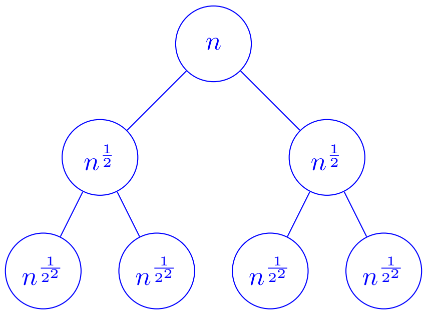
Note: You can replace $\mathcal{O}$ with Θ and you get an alternate (but still true) version of the Master Theorem that produces Θ bounds.

$d_{crit} = \log_b a$ 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.

2 Sorted Array

Given a sorted array A of n (possibly negative) distinct integers, you want to find out whether there is an index i for which $A[i] = i$. Devise a divide-and-conquer algorithm that runs in $O(\log n)$ time.

Solution:

Along the same lines as binary search, start by examining $A[\frac{n}{2}]$. There are three cases for this particular index; $A[\frac{n}{2}] = \frac{n}{2}$, $A[\frac{n}{2}] > \frac{n}{2}$, and $A[\frac{n}{2}] < \frac{n}{2}$.

If $A[\frac{n}{2}] = \frac{n}{2}$, then we have a satisfactory index so we can return true.

If $A[\frac{n}{2}] > \frac{n}{2}$, then no element in the second half of the array can possibly satisfy the condition because each integer is at least one greater than the previous integer (because of the distinct property), and hence the difference of $A[\frac{n}{2}] - \frac{n}{2}$ cannot decrease by continuing through the array. If $A[\frac{n}{2}] < \frac{n}{2}$, then by the same logic no element in the first half of the array can satisfy the condition.

While the algorithm has not terminated, we discard the half of the array that cannot hold an answer, and recurse on the other half applying the same check on the new array's median. If we are left with an empty array, we return false.

At each step we do a single comparison and discard at least half of the remaining array (or terminate), so this algorithm takes $O(\log n)$ time.

3 Quantiles

Let A be an array of length n. The boundaries for the k quantiles of A are $\{a^{(n/k)}, a^{(2n/k)}, \ldots, a^{((k-1)n/k)}\}$ where $a^{(\ell)}$ is the ℓ-th smallest element in A.

Devise an algorithm to compute the boundaries of the k quantiles in time $\mathcal{O}(n \log k)$. For convenience, you may assume that k is a power of 2.

Hint: Recall that QUICKSELECT(A, ℓ) gives $a^{(\ell)}$ in $\mathcal{O}(n)$ time.
Solution: The idea is to find the median of A, partition it into two pieces around this median. Then we recursively find the medians of the two partitions, partition those further, and so on. If we do this $\log k$ times, we will have found all of the k-quantiles.
Finding the median and partitioning A takes $\mathcal{O}(n)$ time. We also do this for two arrays of size $n/2$, four times for arrays of size $n/4$, etc. So the total time taken is

$$\mathcal{O}(n + 2 \cdot n/2 + 4 \cdot n/4 + \cdots + 2^{\log k} n/2^{\log k}) = \mathcal{O}(n \log k)$$

4 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

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

(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 $(e^{2\pi i\theta})^n = e^{2\pi i n\theta}$.
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.