1 Divide and Conquer

Recall from the previous lecture that the idea behind divide and conquer is to:

- Split the problem into parts.
- Recursively solve each part.
- And then somehow recombine these parts to get the final answer.

We will now use this technique to solve several important problems.

2 Sorting

Given a list L of numbers, return L in sorted order.

2.1 Algorithm Idea

An idea is to:

- Divide L into two parts, L_1 and L_2.
- Sort L_1 and L_2.
- Finally, *merge* both L_1 and L_2 together.

2.2 Merge Operation

One question we have is, how do we define the *merge* operation? Well, suppose we have

$$L_1 = [1, 3, 6, 10]$$

$$L_2 = [2, 4, 5, 7, 8, 9]$$

We can combine the two lists by comparing the two smallest elements an then putting the smaller of the two elements in the new list. So, for example, the first element in L_1 is 1 and the first element in L_2 is 2, so we can put 1 into the final list and somehow point L_1 to the second element so we don't consider 1 again.

The algorithm can be described like so:

```
Merge(A, B):
    Let C be the list with the length being len(A) + len(B)
    // Kane uses one-indexing instead of zero-indexing.
    a = 1
    b = 1
    for c = 1 to len(C):
        if (a <= len(A) & (b > len(B) or A[a] < B[b])):
            C[c] = A[a]
            a++
        else:
            C[c] = B[b]
            b++
    Return C
```

This runs in $\mathcal{O}(|A| + |B|)$ time.

2.3 Merge Sort Algorithm

```
MergeSort(L):
    // Every divide & conquer algorithm needs a base case
    if |L| = 1:
        return L

    Split L into approx. equal lists L1, L2
    return Merge(MergeSort(L1), MergeSort(L2))
```

To analyze the runtime, note that the base case runs in $\mathcal{O}(1)$ time; this is obvious. We know that the `Merge` algorithm runs in linear time. Then, the two calls to `MergeSort` takes $2T(n/2 + \mathcal{O}(1))$. We then have (by the Master Theorem):

$$T(n) = \mathcal{O}(n) + 2T(n/2)$$

Therefore, the final runtime is given by the Master Theorem:

$$\mathcal{O}(n \log n)$$

3 Order Statistics

Given a list of numbers L, find the **median** or the **largest element** or the **10th smallest** element of L. Essentially, you want to find something based on the order in some way.

Let's focus on one particular problem: Given L and k, find the kth smallest element of L.

3.1 Easy Algorithm

The easy algorithm is to sort L and then return $L[k]$. The runtime is $\mathcal{O}(n \log n)$, but there are better ways to do this.

3.2 Divide & Conquer Algorithm

First, we pick a *pivot* x, and we sort all elements in L relative to x. That is, every element to the *left* of x is less than x, every element to the *right* of x is greater than x, and every element next to x is equal to x. In this sense, we can sort these elements into categories in $\mathcal{O}(n)$ timme.

We note that the kth smallest element is less than x if and only if $|L_{<x}| \geq k$. If this is the case, then the answer is the kth smallest element of $L_{<x}$. The answer is x if and only if $|L_{<x}| < k$.