CSCI 104L Lecture 16: O(n log n) Sorts

**INEFFECTIVE SORTS**

**MergeSort**:
```c
void MergeSort(T a[], int l, int r) {
    if (l < r) {
        int m = floor((l+r)/2);
        MergeSort(a, l, m);
        MergeSort(a, m+1, r);
        Merge(a, l, r, m);
    }
}
```

The recurrence relation for MergeSort is:
- $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
- $T(1) = \Theta(1)$.
There are many ways you can solve such a recurrence relation, including:

- Draw a recursion tree, calculate the amount of work at each level of the tree, and add it up.
- Do an inductive proof.

**QuickSort**

```c
void QuickSort (T a[], int l, int r) {
    if (l < r) {
        int m = partition(a, l, r);
        QuickSort(a, l, m-1);
        QuickSort(a, m+1, r);
    }
}
int partition(T a[], int l, int r) {
    int i = l;
    T p = a[r];
    for (int j = l; j < r; j++) {
        if (a[j] <= p) {
            a.swap(i, j);
            i++;
        }
    }
    a.swap(i, r);
    return i;
}
```

**Question 1.** Is QuickSort stable?

**Question 2.** What would be a loop invariant for the partition function?

**Question 3.** What would be a recurrence relation for QuickSort?

**Question 4.** What would be the best and worst choice of pivot, in terms of runtime?

QuickSort is a prime example of an algorithm for which it makes best sense to do average-case runtime analysis.