Algorithm Analysis

Reagan Shirk

September 3, 2020

Efficiency

- You have an insertion sort where the time for the first half is $n - \sqrt{n}$ and the time for the second half is $\sqrt{n}$. Find the overall complexity

$$\begin{aligned} & c_1(n - \sqrt{n}) + c_2\sqrt{n} \\ \text{Aside: } \sqrt{n} &= [(n - \sqrt{n}) + (n - \sqrt{n} + 1) + \cdots + (n + 1)] \\ c_1(n - \sqrt{n}) + c_2\sqrt{n} &= cn^d \\ \sqrt{n}(n - \sqrt{n}) \leq \sqrt{n} &\leq \sqrt{n} \times n \\ \sqrt{n}(n - \sqrt{n}) &= n - n^{1.5} \\ d &= 1.5 \end{aligned}$$

- I didn't really follow this at all but I tried to make sense of what he wrote down. I don't think I did a very good job

Divide and Conquer

- Insertion sort isn't super efficient with the worst case being $\Theta(n^2)$
- Divide and conquer is trying to make shit more efficient, dividing shit into equal halves and solving that way
- Merge sort is a divide and conquer algorithm

Merge Sort

```
MERGE-SORT(A, p, r)
    if p < r                    // check for base case
        q = floor((p + r)/2)    // divide
        MERGE-SORT(A, p, q)     // conquer
        MERGE-SORT(A, q + 1, r) // conquer
        MERGE(A, p, q, r)       // combine
```

- The complexity of a merge sort is $\Theta(n \log(n))$
- Doesn't matter for the complexity, but generally log in this class is $\log_2$, not $\log_{10}$
  - Cheng will usually write it as lg for $\log_2$

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1...n1 + 1] and R[1---n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
```

```
L[n1 + 1] = infinity
R[n2 + 1] = infinity
i = 1
j = 1
for k = p to r
    if L[i] <= R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```