Jacob Alspaw
jaa134
EECS 340 - Algorithms
Assignment 2

4.3-1
$T(n) = T(n-1) + n$
GUESS: $T(n) = O(n^2)$
From here we most prove that $T(n) \leq cn^2$ where constant c is > 0.
This leads to the assumption that $T(n-1) \leq c(n-1)^2$
SUBSTITUTION:
$T(n) \leq c(n-1)^2 + n$
$= c(n^2 - 2n + 1) + n$
$= cn^2 - c2n + c + n$
$= cn^2 - (2c-1)n + c$
$\leq cn^2$

$cn^2 - (2c-1)n + c \leq cn^2$
$c + n \leq 2cn$
The above relationship is true when $c > 0.5$ and $n \geq \frac{c}{2c-1}$, which makes $T(n) = O(n^2)$.

4-1c
$T(n) = 16T(n/4) + n^2$
a = 16 and b = 4
$\rightarrow log_4 16 = 2$
$f(n) = n^2$
Case 2 Master Method: $T(n) = \Theta(n^2 logn)$

4-1d
$T(n) = 7T(n/3) + n^2$
a = 7 and b = 3
$\rightarrow log_3 7 \approx 1.77$
$f(n) = n^2$
Case 3 Master Method: $T(n) = \Theta(n^2)$
Regularity Condition: $7(\frac{n}{3})^2 \leq cn^2$ when constant $c \geq 7/9$.
$7(\frac{n}{3})^2 \leq cn^2 \rightarrow \frac{7n}{9}^2 \leq cn^2 \rightarrow 7/9 \leq c$

4-1e

$T(n) = 7T(n/2) + n^2$
a = 7 and b = 2
$\rightarrow log_2 7 \approx 2.81$
$f(n) = n^2$
Case 1 Master Method: $T(n) = \Theta(n^{2.81})$

4-1f

$T(n) = 2T(n/4) + \sqrt{n}$
a = 2 and b = 4
$\rightarrow log_4 2 = 0.5$
$f(n) = \sqrt{n}$
Case 2 Master Method: $T(n) = \Theta(\sqrt{n} logn)$

7.2-5

Minimum depth: This part of the recursion tree can always be found in the smaller part of the partition. The partition will change the amount of elements by α. That is to say, that after i iterations, there will be $\alpha^i n$ elements. At the last iteration, k, there will be only one element remaining.
$\alpha^k n = 1$
$\alpha^k = \frac{1}{n}$
Take log of both sides $\rightarrow k = \frac{-log(n)}{log(\alpha)}$

Maximum depth: This part of the recursion tree can always be found in the larger part of the partition. The partition will change the amount of elements by $1-\alpha$. That is to say, that after i iterations, there will be $(1-\alpha)^i$ elements. At the last iteration, k, there will be only one element remaining.
$(1-\alpha)^k n = 1$
$(1-\alpha)^k = \frac{1}{n}$
Take log of both sides $\rightarrow k = \frac{-log(n)}{log(1-\alpha)}$

Quicksort Video

```
Partition(A,p,r)
    x ← A[p]
    i ← p
    j ← r + 1
    while i < j do
        while A[j] > x do
            j ← j - 1
        while A[i] < x do
            i ← i + 1
        swap(A[i], A[j])
    swap(A[i], A[j])
```

LOOP INVARIANT:
At the start of the j^{th} iteration...
1. All numbers in A[p...i] $\leq$ x
2. All numbers in A[i+1...r-1] $\geq$ x

Initialization: Prior to the first iteration of the loop, i = p and j = r+1. The variable x has been set to the value of A[p];

Maintenance: The invariant holds from the prior iteration. Everything from the left of i will agree with loop invariant condition 1. Everything from the right of j will obey loop invariant condition 2. The first of the nested while loop will skip over any list item where A[j] > x. This inner loop will stop and mark the index of the first instance of a value where A[j] < x. This is the only case we need to consider. The same is true for the second nested while loop; it will skip over any list item where A[i] < x. This inner loop will stop and mark the index of the first instance of a value where A[i] > x. Onceboth of these indexes are found, the values at these locations are swapped. Values are put in their respective sides to abide by loop invariant 1 and 2. The first loop picked a number less than x and moved it to the left partition mentioned in loop invariant 1, while the second loop picked a number greater than x and moved it to the right partition mentioned in loop invariant 2.

Termination: When i $\geq$ j the loop will end. The outer while loop will have repeated the process for all numbers on the wrong side of the partition, preserving the loop invariant. Once index i has moved past index j, the first and second while loops have inspected all numbers up until that point. If there was a number out of its proposed partition, then it would have been swapped. This concludes that at termination of the loop, A[p...i] $\leq$ x and A[i+1...r-1] $>$ x.

Run-time Analysis: $\Theta(n) \rightarrow$ The method will have to examine all elements of the list because it lacks an exit criteria from the body of the loop. On an average case, the outer while loop will move indexes i and j toward the center, performing operations of constant run-time along the way, like swapping or changing index value. Outside of the loop, there are only more constant run-time operations, such as initializing variables or switching list values.