CSCI 430: Homework 4

John E. Buckley III

October 1, 2017

1 2.3-3

Base Step:
If $n = 2$ then $T(2) = 2$ and $2log2 = 2$ Thus, $T(n) = 2log2$

Hypothesis Step:
Assuming $T(n) = nlogn$ is true if $n = 2^k$ for some integer $k > 1$

Induction Step:
If $n = 2^{k+1}$ then $T(2^{k+1})$
$= 2T(2^{k+1}/2) + 2^{k+1}$
$= 2T(2^k) + 2^{k+1}$
$= 2(2^k log2^k) + 2^{k+1}$
$= 2^{k+1}((log2^k) + 1)$
$= 2^{k+1} log2^{k+1}$

2 2.3-4

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(n-1) + C(n-1) & \text{otherwise.} \end{cases}$$

Where $C(n)$ is the time needed to insert an element into a sorted array of n elements.

3 2.3-5

```
Recursive-Binary-Search(A,p,q,v):
    if p < q
        return NIL
    half= ⌊((p + q)/2)⌋
    if (v == A[half])
        return half
```

```
    else if (v < A[half])
        Recursive-Binary-Search(A,p,q,v)
    else
        Recursive-Binary-Search(A,half+1,q,v)
```

After the first iteration we have N/2 items remaining, after the second iteration we have N/4 remaining, and so on, hence N/2^k. Worst case: Last iteration occurs when N/$2^k \geq 1$ and N/$2^{k_1} < 1$ item remaining. take log of both sides to get $2^k \leq N$ and $2^{k+1} > N$. Number of iterations is K≤logN and K>logN-1, thus the worst case running time is $\Theta(logN)$.

4 2.3-6

No, because it still needs to shift all elements after it to the right, which is linear in the worst case, even if it finds the position in log time.

5 2-1a

Since Insertion-Sort runs in $\Theta(n^2)$ worst-case time, then each list with length k will take $\Theta(n^2)$ worst-case time. To sort $\frac{n}{k}$: $\frac{n}{k} * k^2 = nk$ thus $\Theta(nk)$.

6 2-1b

Merging $\frac{n}{k}$ into $\frac{n}{2k}$ then into $\frac{n}{4k}$ and so on takes $\Theta(n)$ time and since we have lg$\frac{n}{k}$ of these merges then merging into one list will take Θ $(nlg$ $(\frac{n}{k}))$ time.

7 2-2a

We must prove that A' consists of the elements in A and that they are in sorted order.

8 2-2b

Loop Invariant:
Prior to each iteration, the elements in A[j...n] are an alteration of the elements that were originally in A[j...n] such that the first element is the smallest among them.

Initialization:
Initially the subarray contains only one element, A[n], whish is the smallest element of the subarray.

Maintenance:

At every iteration we compare A[j] with A[j-1] and the smaller of the two becomes A[j-1]. The length of the subarray grows by one with each iteration and the first element is the smallest of the subarray.

Termination:
The loop terminates when j=i+1. The length of the subarray grows by one, the first element is the smallest in the subarray, and we swap A[i+1] with A[i].

9 2-2c

Loop Invariant:
Prior to each iteration, the subarray A[1...i-1] consists of the elements in the subarray A[i...n] but in sorted order.

Initialization:
Initially the subarray is empty, which obviously makes it the smallest element of the subarray.

Maintenance:
At every iteration, the elements that are in A[1...i-1] with be the smallest value of the previous iteration, thus A[1...i-1] will be in sorted order and smaller than A[i...n].

Termination: The loop terminates when i=A.length, which, at that point means the array A[1...n] will be in sorted order.

10 2-2d

Bubblesort will iterate over the whole array every time for every element, thus $\Theta(n^2)$ just like the worst-case time for insertion-sort. However, the constants for bubblesort are much larger than insertion-sort, so insertion-sort still runs faster.