# CSCI 430: Homework 4

Annabelle Cormia

October 1, 2017

Tyler Archer and I collaborated on this assignment.

## 1 Chapter 2.3-3

Base Case ($n = 2$): If $n = 2$, then $T(2) = 2$ and $2lg2 = 2$. This implies $T(2) = 2lg2$.
Inductive Hypothesis: Suppose this holds true for all values of $n = 2^k$ for some $k$ s.t. $k > 1$.
Inductive Step: Let $n = 2^{k+1}$. Then:

$2T(n/2) + n$
$2T(2^{k+1}/2) + 2^{k+1}$ By substitution
$2T(2^k) + 2^{k+1}$ By exponent rules
$2(2^k lg2^k) + 2^{k+1}$ By base case
$2^{k+1}(lg2^k) + 2^{k+1}$ By distribution and exponent rules
$2^{k-1}(lg2^k + 1)$ By factoring out $2^{k-1}$
$2^{k-1}lg2^{k-1}$

## 2 Chapter 2.3-4

When there is only one item in the array ($n = 1$), the running time is constant.
When there is more than one item in the array ($n > 1$), the recursion sorts up to $n - 1$ and the running time is $\theta(n)$.
Therefore, the recurrence is:

$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ T(n-1) + O(n) & \text{if } n > 1 \end{cases}$$

## 3 Chapter 2.3-5

binary-search(A, x)
left = A[1]

```
right = A[A.length]
while left <= right
    middle = ⌊(left + right)/2⌋
    if x == A[middle]
        return middle
    elseif x > A[middle]
        left = middle + 1
    else
        right = middle -1
return NIL
```

# 4 Chapter 2.3-6

No, the insertion sort must still insert the sorted element (regardless of using linear search or binary search) so it will still produce a worst-case running time of $\theta(n^2)$.

# 5 Chapter 2-1 a

$k$ produces a time of $\theta(k^2)$.
$n/k$ produces a time of $n/k(\theta(k^2)) = \theta(nk)$.

# 6 Chapter 2-1 b

Merging n/k sublists with n elements and a length of k will produce a worst-case time of $\theta(nlg(n/k))$.

# 7 Chapter 2-2 a

We must prove that it produces A' with the same elements as present in A but now in sorted order.

# 8 Chapter 2-2 b

At the start of each iteration of the for loop of lines 2-4, the subarray $A[j...n]$ contains the same elements in $A[j...n]$ where $A[j]$ will always the the smallest.

Proof:

Initialization: At the beginning, the subarray contains only the smallest element therefore the L.I. trivially holds.

Maintenance: After each iteration, $A[j]$ is replaced with $A[j-1]$ if $A[j-1] > A[j]$. $A[j-1]$ then becomes the smallest element. Otherwise, $A[j-1]$ is already the smallest element.

Termination: When the loop terminates, $A[i]$ is the smallest element.

# 9 Chapter 2-2 c

At the start of each iteration of the for loop of lines 1-4, the subarray $A[1...i-1]$ contains all the sorted elements that are all smaller than all the elements in $A$.

Proof:

Initialization: At the beginning, the subarry contains no elements, therefore the L.I.trivially holds.

Maintenance: After each iteration, A[i] becomes the smallest element of the subarry and the subarray will contain all of the smallest elements in sorted order.

Termination: When the loop terminates, $i = A.length$ and $A[1...n]$ contains all the sorted elements.

# 10 Chapter 2-2 d

The worst-case running time will involve the algorithm will iterate over every element in the array which produces a time of $\theta(n^2)$. Insertion sort has the same running time of $\theta(n^2)$.