# CMPSC 465 – Spring 2021 — Solutions to Homework 3

Josiah Kim, SID 948500821, `juk483`

February 11, 2021

## 1. Acknowledgements

(a) I did not work in a group.

(b) I did not consult with any of my group members.

(c) I did not consult any non-class materials.

## 2. Solving Recurrences

(a) 1)$T(n) = 11T(\frac{n}{5}) + O(n^{1.3})$; $a = 11$, $b = 5$, $d = 1.3$
1.3 ? $log_5 11$
$1.3 < 1.489896$
$T(n) = O(n^{log_5 n})$

2)$T(n) = 11T(\frac{n}{5}) + \Omega(n)$; $a = 11$, $b = 5$, $d = 1$
1 ? $log_5 11$
$1 < 1.489896$
$T(n) = \Omega(n^{log_5 n})$

3)The answers agree. Therefore, $\Theta(n^{log_5 n})$

(b) 1)$T(n) = 6T(\frac{n}{2}) + O(n^{2.8})$; $a = 6$, $b = 2$, $d = 2.8$
2.8 ? $log_2 6$
$2.8 > 2.58$
$T(n) = O(n^{2.8})$

2)$T(n) = 6T(\frac{n}{2}) + \Omega(n)$; $a = 6$, $b = 2$, $d = 1$
1 ? $log_2 6$
$1 < 2.58$
$T(n) = \Omega(n^{log_2 n})$

3)The answers do not agree. Therefore, we can use the upper bound as both the lower and upper bound: $\Theta(n^{2.8})$

(c) RECURSION TREE:
Branching Factor: 5
Height: $log_3 n$
Size: $\frac{n}{3^k}$
Number: $5^k$
$W_k$: $5^k log^2(\frac{n}{3^k})$
Total Work: $\sum_{k=0}^{log_3 n} 5^k log^2(\frac{n}{3^k}) = \Theta(logn)$

(d) FOLDING METHOD:
$T(n) = T(n-2) + O(logn)$
$T(n) = T(n-4) + O(logn) + O(logn)$
$T(n) = T(n-6) + O(logn) + O(logn) + O(logn)$
$T(n) = T(n-2k) + kO(logn)$
$\Theta(logn)$

## 3.Sorted Array

```
def index_match(A, i=0):
        if i == A[i]:
                return i
        if i < A[n/2]:
                index_match(A[1...n/2], i++)
        if i > A[n/2]:
                index_match(A[n/2...n], i+(n/2))
        else:
                return false
```

This function takes two inputs: a list (A) and an index (i). If the index is equal to the value of the integer at the spefic index ($A[i] = i$), then it returns the index/integer.

If not, then it checks if the index is less than or greater than the integer at the median ($A[n/2]$). If the integer at the median is less than the index, it will recursively call the function using the first half of the array and adding 1 to the index. Similarly, if the integer at the median is greater than the index, it will recursively call the function using the second half of the array and adding 1 to the index.

Since this algorithm does not does not iterate over every item in the array it would be quicker than $O(n)$. Furthermore, since the input is partitioned in half for every recursive call, this algorithm has a time complexity of $O(logn)$. If the input size were doubled, the algorithm would take one more step.

## 4.Linear Time Sorting

```
sort(unsorted):
        min = 0
        max = 0
        sorted = []
        counter = {}
        for x in unsorted:
                if x > max:
                        max = x
                if x < min:
                        min = x
        for y in range(min, max):
                counter[y] = 0
        for i in counter:
                for j in unsorted:
                        if i == j:
                                counter[j] += 1
        for z in counter:
                if counter[z] != 0:
                        for w in range(counter[z]):
                                sorted.append(z)
        return sorted
```

This function takes the argument of an unsorted list. The first loop iterates through the unsorted list and finds the min and max values which will have a time complexity of $O(n)$.

The second "for" loop, initializes a dictionary, counter, with the key being the integers from min to max and the values being 0. This will result in the time complexity of $O(M)$.

The third loop iterates over the counter again with its nested loop iterating over the unsorted list for each key of the dictionary. Once there is a match with the key of the dictionary and the integer in the unsorted list, the value is incremented by one. This section will result in the time complexity of $O(Mn)$. The final loop will then iterate over the counter. If the value of a key is not zero, it will append the key to the resulting list called sorted. Since this loop iterates over counter, it will result in another time complexity of $O(M)$.

Finally, the function returns the sorted list. We can see that the time complexity is dependent on M for three out of the four loops. Therefore, without knowing the relationship between n and M, it is safe to say that $O(n + M)$.