Homework 3

CMPSC 465

Kinner Parikh

June 29, 2023

Problem 1:

I did not work in a group
I did not consult without anyone my group member
I did not consult any non-class materials

Problem 2: *Solving recurrences*

a) $T(n) = 11T(n/5) + 13n^{1.3}$
$W_k = 11^k \times 13(n/5^k)^{1.3}$
$\sum_{k=0}^{\log_5 n} W_k = 13n^{1.3} \sum_{k=0}^{\log_5 n} \left(\frac{11}{5^{1.3}}\right)^k = \Theta(13n^{1.3} \cdot \left(\frac{11}{5^{1.3}}\right)^{\log_5 n}) = \boxed{\Theta(13n^{\log_5 11})}$

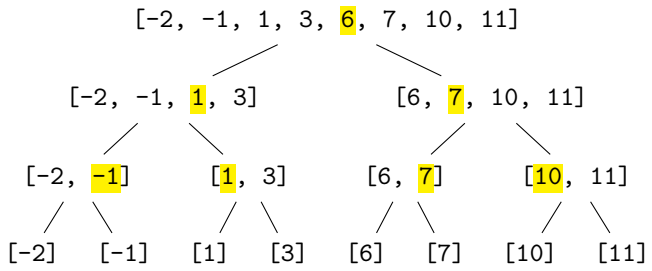
b) $T(n) = 6T(n/2) + n^{2.8}$
$a = 6, b = 2, d = 2.8$
$\log_2 6 < 2.8$, so by Master's theorem, $\boxed{\Theta(n^{2.8})}$

c) $T(n) = 5T(n/3) + \log^2 n$
Lower Bound Case: $1 < \log^2 n$ and $\log_3 5 > 1$
Upper Bound Case: $\log^2 n < n$ and $\log_3 5 > n$
Thus, $\boxed{\Theta(n^{\log_3 5})}$.

d) $T(n) = T(n-2) + \log n$
This will recurse $n/2$ times so total work will be $\sum_{k=1}^{n/2} \log(2k)$.
This will be $\log(2) + \log(4) + ... + \log(\frac{n}{2} - 1) + \log(\frac{n}{2})$, this simplifies to $\log(n^n)$, thus $\boxed{\Theta(n \log n)}$.

Problem 3: *Sorted Array*

The basic structure for this algorithm is that you take an upper(u) and lower(l) bound of indeces and find the average($\texttt{avg} = \frac{u+l}{2}$) to check if the value at that index is equal to, greater than, or less than the index. If it is equal to the index, then we have found our $A[i] = i$. If $i < A[i]$, then we make a recursive call with l staying as is, but $u = \texttt{avg}$. If $i > A[i]$, then we make a recursive call with $l = \texttt{avg}$ and u staying as is. This will eventually converge to find the $A[i] = i$ in $O(\log n)$ time. This cuts the problem in half every single time because we are changing the bounds for which we search. For example, suppose we are given array: `[-2, -1, 1, 3, 6, 7, 10, 11]`. This will be broken down into this tree:



The highlighted numbers are those that will be compared to their index in the array. If the index is less than the value, it continues down the left side of the tree, and the opposite for if the index is greater than the value. Ultimately, this leads us to see that $T(n) = 2T(n/2) + 1$. Using Master's Theorem, we can see that $a = 2, b = 2, d = 0$, which shows us that this will run in $O(\log n)$ time.

Problem 4: *Linear Time Sorting*

The algorithm for this is fairly straightforward. We iterate through the array once to find the largest and smallest value in the array, let us call these x_{max} and x_{min}. We can then create m number of buckets, one for each integer between x_{max} and x_{min}, where $m = x_{max} - x_{min}$. Then passing through the array one more time, we can place the values into their respective buckets. Finally, from these buckets, we can place the values into x in order, and we can say that x is sorted. As an example, take the array:

$$x = \texttt{[4, 9, 1, 4, 2, 0, 10, 2, 5, 2, 1, 8, 2, 6, 2, 5]}$$

We can do one pass through the array and see that $x_{min} = 0$ and $x_{max} = 10$. Then we can make buckets for all the values between 0 and 10. Running through the array one more time, we can place the number into its bucket. This would look like:

```
0 → [0]
1 → [1, 1]
2 → [2, 2, 2, 2, 2]
3 → []
4 → [4, 4]
5 → [5, 5]
6 → [6]
7 → []
8 → [8]
9 → [9]
10 → [10]
```

Then we can combine these in order to get:

$$x = \texttt{[0, 1, 1, 2, 2, 2, 2, 2, 4, 4, 5, 5, 6, 8, 9, 10]}$$

This will run in $O(n+m)$ because we pass through the array twice, which is $O(n)$, and then adding it from the buckets will take $O(m)$ time, so $O(n) + O(m) = O(n + m)$.