CS 371 HW 4

Josh Blaz

3/6/19

1. Chapter 5, Exercise 1:
   To solve this problem, I would begin by querying for the exact median in both databases, m_1 for database 1 and m_2 for database 2. The median of the $2n$ unique values in the databases must be between m_1 and m_2. If m_1 is greater than m_2, then we can choose all values m_1 and below in database 1 and all values m_2 and above in database 2 (and vice versa, if m_1 is less than m_2), this gives us a set of n values in which we can determine the median (the combined values of m_1 and above in database 1 and m_2 and below in database 2). From here,

2. Chapter 5, Exercise 2:

   (Sort and Count (Significant inversions))

Data: List of integers
Result: Number of significant inversions, and a sorted list
if len(L) == 1, return (0,L)
 Split L into L_1 and L_2 evenly.
 (i_L, L_L) = Sort-and-Count(L_1)
 (i_R, L_R) = Sort-and-Count(L_2)
 (i_B, L_B) = Merge(L_L, L_R)
 return $(i_L + i_R + i_B, L_B)$

(Merge function)

Data: Two Lists
Result: Number of significant inversions between and a merged list
L = []
$i = 0, j = 0, i_B = 0$
while *i ¡ len(L_L) and j ¡ len(L_R)* **do**
  Append the smaller of $L_L[i]$ and $L_R[i]$ to L
  if $2 * L_L[i] > L_R[j]$ **then**
    $i_B + = len(L_L) - i$
  end
  Increment either i or j.
end
Return iB, L

This algorithm works the same as the Sort-And-Count algorithm we looked at that just counts inversions, the only real change being that this algorithm only counts an inversion if it's a "Significant Inversion", that is, if there exists some pair $i < j$ where $a_i > 2_{aj}$. Given that there is only one real change to the algorithm and it's a simple change to the comparison, this algorithm runs at $O(nlogn)$ time.

3. Chapter 5, Exercise 3:
Start by splitting the bank cards into two different subsets: S_1 and S_2. Suppose that if one card c is the most common bank card in the original set of cards (combined set), if it is, it must be that the card is also the most common bank card in either S_1 or S_2, or it could be the most common bank card in both sets (in which case it would be the most common bank card in the combined set). If c is not the most common card in both sets, then we simply find the most common cards c_1 and c_2 for sets S_1 and S_2, which will take $O(n)$ time because there are n bank cards in total. We can continually splitting the sets recursively, in which the base case is $n == 1$, simply returning the single last card as the most common card.

This algorithm works because the most common card between two subgroups can be the most common card of either group - it could be the most common card in either S_1 or S_2. Since we recursively split up the groups $n/2$ every time, we will eventually find the most common bank card.

In terms of run time, we do $O(logn)$ splits whenever we recursively split each set until the base case of 1, and $O(n)$ comparisons for every subgroup, we end up with $O(nlogn)$ time.

4. Code.