Median Finding
Tom Bohbot
February 2021

1) Why does vanilla binary search not work?
    a. Although a binary search performs in O(log(n)) time, it would not work in this case since we are dealing with two separate sets of accounts and they are independently sorted. To use a binary search we would first need to merge and sort the two sets which would take longer than O(log(n)) time and would therefore be invalid due to time constraints in the requirement doc. Furthermore, once we have a single combined sorted set we would be able to find the median in constant time through calculating the middle index.

2) Describe the core idea of your algorithm:
    a. My algorithm recursively calculates the median of both sets through splitting both sets in half each time until a base occurs. Every time I split the sets in half I compare the medians of each set. If the first set's median is smaller than the seconds then I will return the upper half of the first set and lower half of the second set. Otherwise, I will return the lower half of the first set and upper half of the lower set. I will recursively do this until there is only one element in each set. I then return the smaller element of the two.

3) PseudoCode:

```
def findMedian (set1, set2):
    return recursively_find_median(set1, set2, 0, max_index_set1, 0, max_index_set2)

def recursively_find_median(set1, set2, min1, max1, min2, max2):
    if max1 == min1 and max2 == min2:
        if set1[min1].get_income() < set2[min2].get_income():
            return set1[min1]
        else:
            return set2[min2]
    else:
        median_one = set1[(min1 + max2)/2].get_income()
        median_two = set2[(min2 + max2)/2].get_income()
        if median_one < median_two:
            return recursively_find_median(set1, set2, (max1 + min1 + 1)/2, max1, min2, (max2 + min2)/2)
        else:
            return recursively_find_median(set2, set1, (max2 + min2 + 1)/2, max2, min1, (max1 + min1)/2)
```

4) Recurrence Relation: n = length of each array
   a. Q(n) = $\begin{cases} \Theta(1), \ if\ n = 1 \\ Q\left(\frac{n}{2}\right) + \Theta(1), if\ n > 1 \end{cases}$
   b. Proving Recurrence Relation Using Master Theorem
      i. a = 1
      ii. b = 2
      iii. d = 0
      iv. Between the three cases, this situation matches the first case since a = b^d or equivalently 1 = 2^0 which is 1 = 1.
      v. Case 1: T(n) = Θ (n^dlog(n)) = Θ ($n^o log(n)$) = Θ (log(n))

5) Proof:
   a. Direct Proof:
      i. Before beginning the proof one must understand the axiom that there must always exist some median account from these two sets of accounts since they are non-empty sets. Additionally, the premises of this proof state that both sets have equal length, are each sorted in ascending order, every account is unique, and that the definition for median is (min_index + max_index)/2. One can calculate the medians of each set in constant time through using the definition given for median. Once one has the median of each set they can compare them to deduce which quadrant the absolute median must lie in. If the median of set1 is less than the median of set2 then the median must lie somewhere in the interval between the upper half of set1 and lower half of set2 or vice versa if set2's median is less than set1. One can be sure that the median will lie within the upper half of the smaller medians set and lower half of the higher median as those two quartiles represent the central two quartiles. Since a median must be the nth element it is impossible for the nth element to be located before the smaller median or after the bigger median since those areas represent the lower 25% and upper 75% quartile of both sets and we are searching for the 50% mark (n). One can recursively repeat this process and weed out the local lower and upper quartiles until there is only one element in each set remaining, as that would represent each set's absolute median. They are the absolute median of each set since anything other than the medians were weeded out through the recursive splitting. Once this situation occurs the base case will return the smaller of the two elements. The median will always be the smaller of the two since the desired element is 2n/2 which is equal to the max element of the smaller set.