Jiawei Wang

Question 1:

1.1

- The time to form $\frac{n}{k}$ groups of k elements, find the median of each group, recursively find the median among all group medians(denoted as x) and partition around x will be: $T(\frac{n}{k}) + \theta(n)$
- The time to recurse on the subset of elements that contains if $rank(x)$ is not the target rank is: $T(\frac{2k-\lceil\frac{k}{2}\rceil}{2k}n) \leq T(\frac{3}{4}n)$
- So the total time is: $T(n) \leq T(\frac{n}{k}) + T(\frac{3}{4}n) + \theta(n)$
- Claim: $T(n) \leq cn$

$$T(n) \leq c\frac{n}{k} + c\frac{3}{4}n + dn \tag{1}$$

$$= \frac{4+3k}{4k}cn + dn \tag{2}$$

$$= cn - \left(\frac{k-4}{4k}cn - dn\right) \tag{3}$$

- If $\frac{k-4}{4k}cn - dn \geq 0$ then $T(n) \leq cn$ is proved
- Since $k \geq 6$, as long as $c \geq \frac{4k}{k-4}d$, $T(n) \leq cn$ is true

1.2

- With k becomes bigger, the runtimes also becomes longer. Because $\frac{k-4}{4k}$ comes closer to $\frac{1}{4}$ so in the recursion, everytime the excluded elements become less, we need more time to get to the targeted element.
- in order to get optimized time, we should use $k = 5$.

Question 2:

- I want to find an algorithm to solve this question by counting the occurrences time for every element, and if one element occurs more than half of the total n time, then we are done. By using Hash Table I can do it in linear time and this algorithm is:

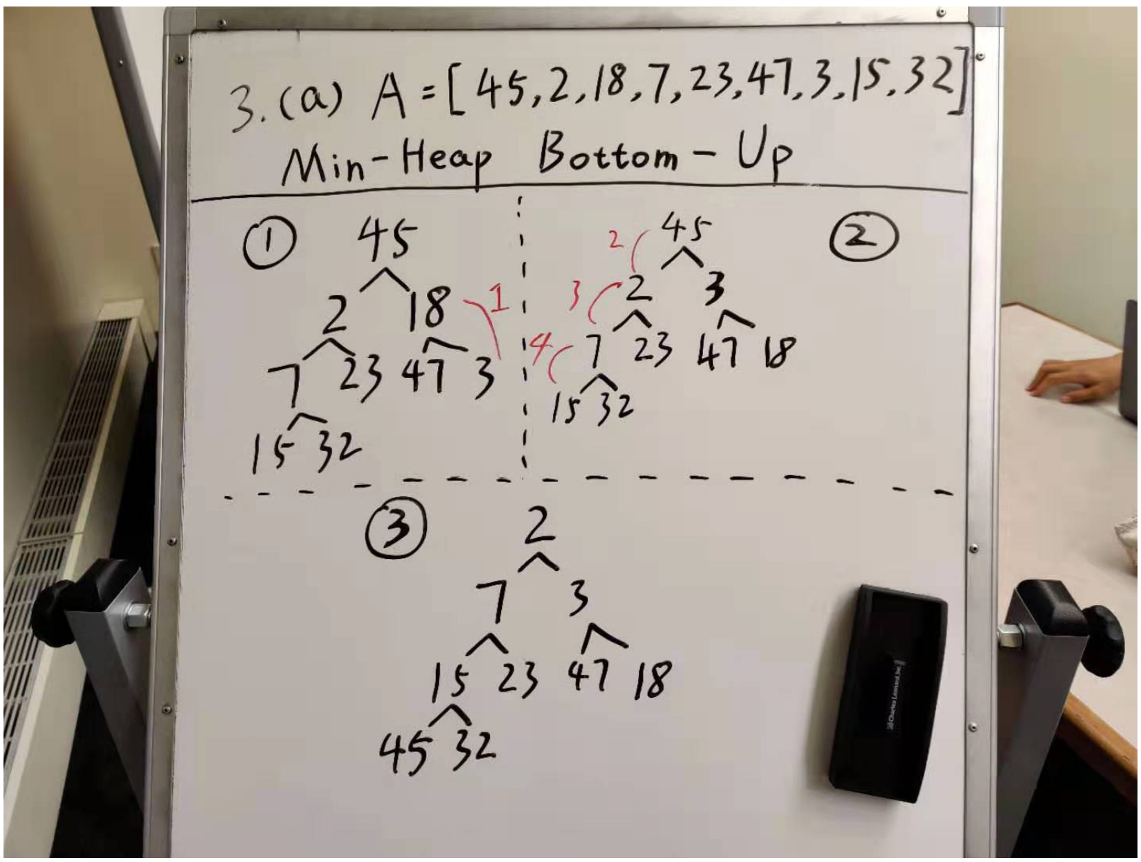
- Make a HashMap, with one loop through the list of n numbers, we can map every element to counts in order to count occurrences, and then return the key with maximum value. If the value is bigger than $\frac{n}{2}$, then we are done.

- Same for the second question, return the value, if the max value is bigger than $\frac{n}{100}$, then there are at least $\frac{n}{100}$ numbers with equal value

Question 3:

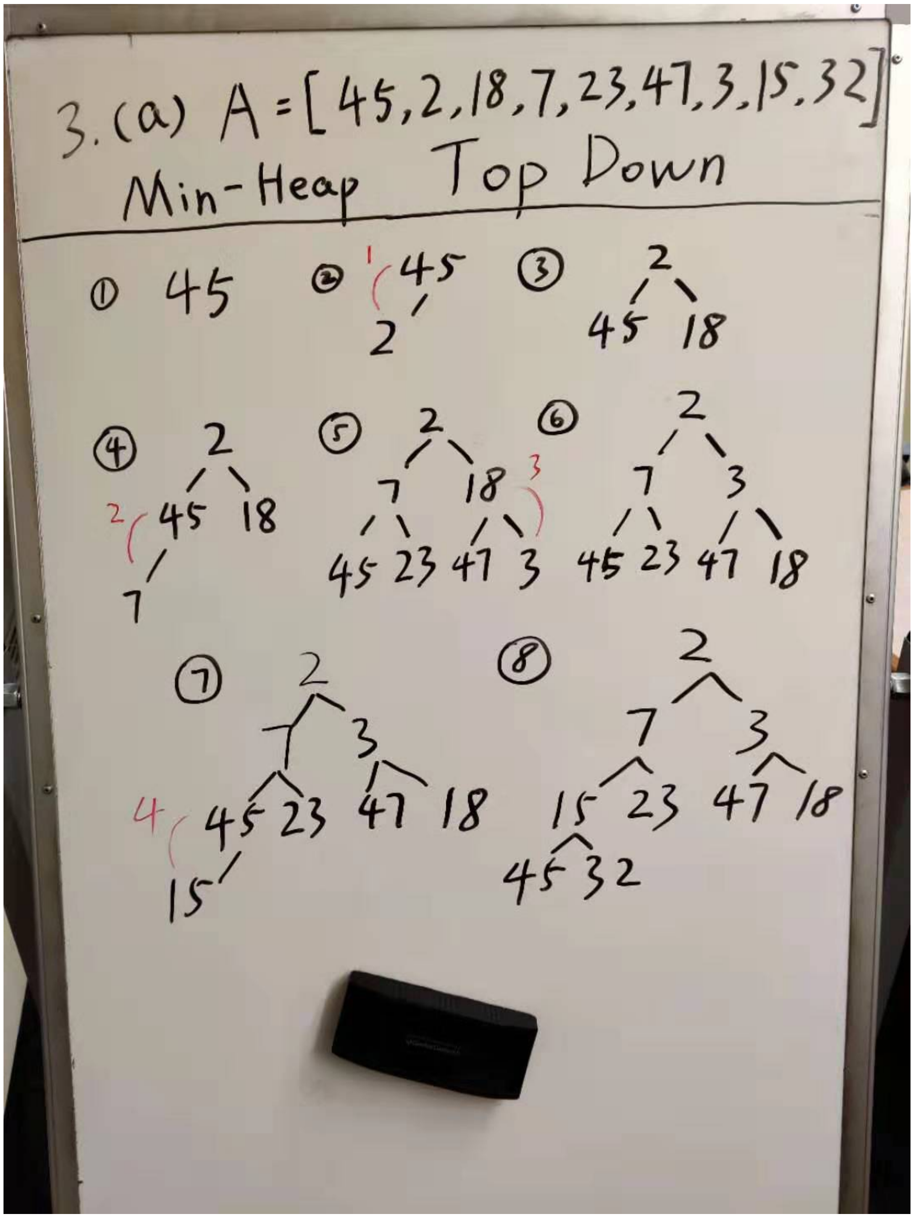
3.1 bottom up

12 comparisons needed. 4 swaps needed.



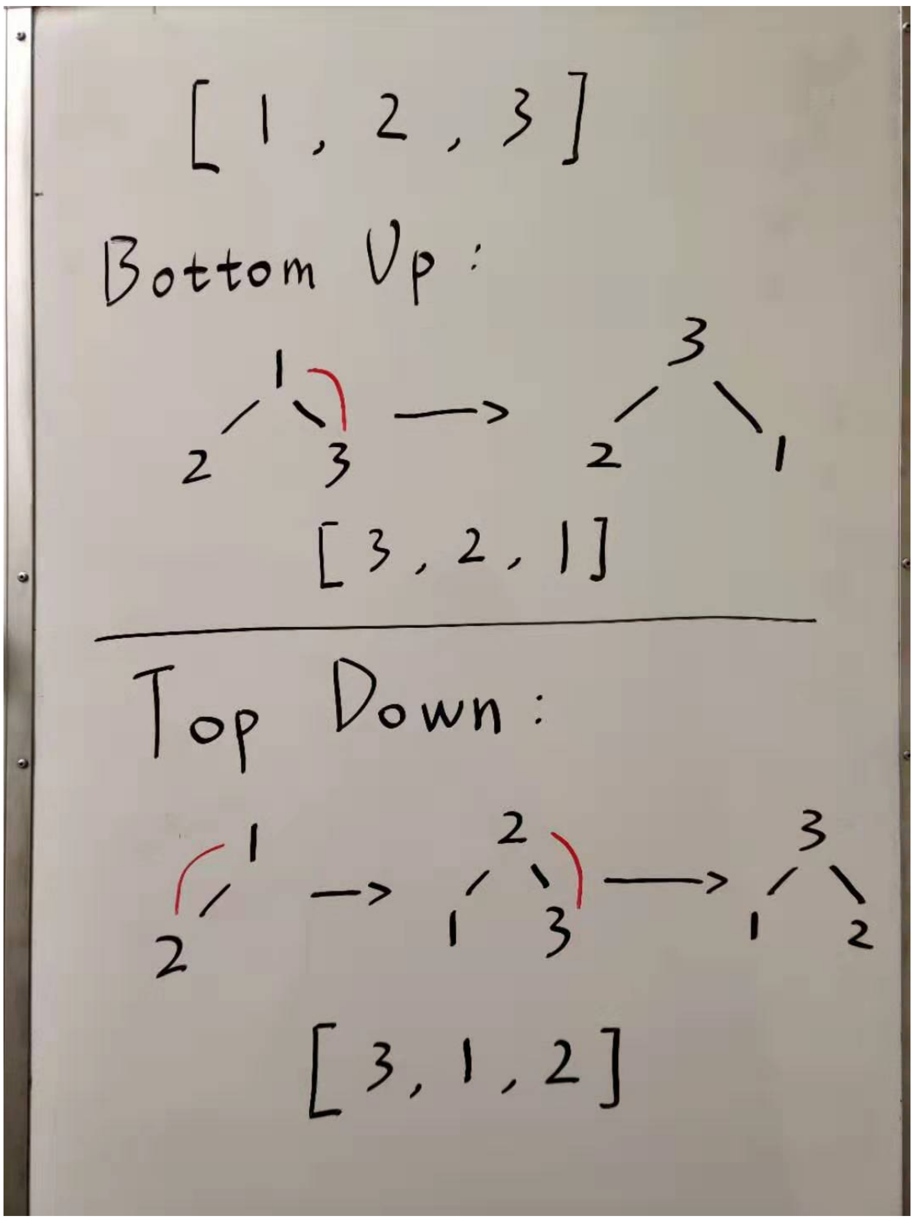
3.2

11 comparisons needed. 4 swaps needed.



3.3

- Although the 2 min heaps are the same, the bottom up and top down methods do not always create the same heap.
- For example: make a max heap using [1,2,3]



Question 4:

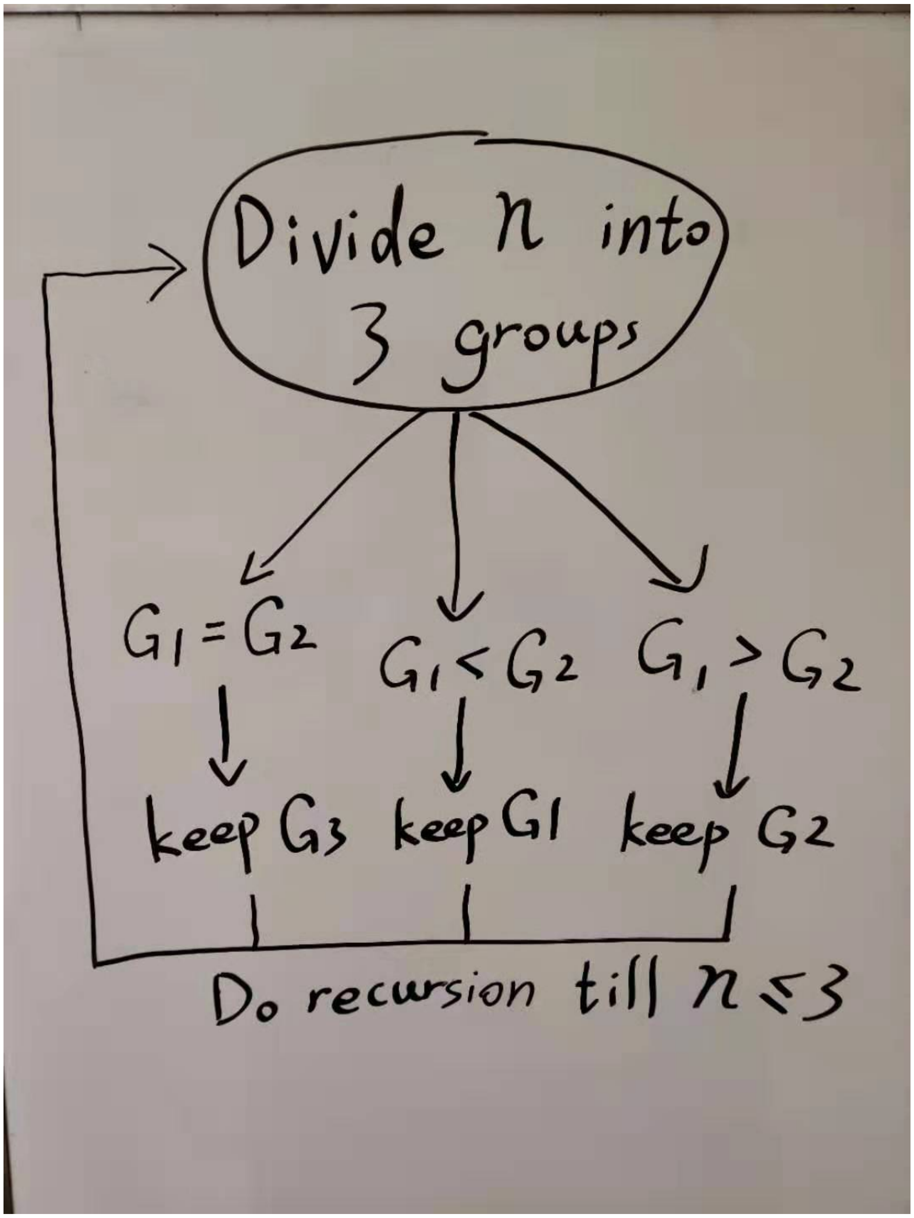
- First we need to find the optimized number of sets for weighings. There are 3 options: 2 (Group1 and Group2), 3 (Group1, Group2 and Group3) and n (Group1 to Groupn)($n \geq 3$):

- If we devide coins for n groups then for the first weighing, we have three possible results: $G1 \leq G2$ which means fake coin is in $G1$, $G1 \geq G2$ which means fake coin is in $G2$, $G1 = G2$ which means fake coin is in the rest of sets, so when this situation happens, we will have to check the G3 to Gn groups, so the more sets we make, the more potential weighings we need to do. Therefore, the optimized number of sets should be no more than 3 sets.

- If we divide coins for 2 groups then we can see G3 as 0, and for the three possible results, our minimum time of weighing should be $min = (G1 + G2, G1 + G3, G2 + G3)$, because $G1 = G2$, so the worst case is to discard the smallest pair, in order to make the pair as big as possible, we must set $G1 = G2 = G3$, so we can say that dividing into 3 same groups is the optimized solution.

  (please turn to next page)

- Decision Tree



- Because the 3 groups are the same and each weighing takes the same $\Theta(1)$ time, we can know that:

$$T(n) = T(\frac{n}{3}) + \Theta(1) \tag{4}$$

- we continue the weighing till $n \leq 3$, that is $T(1) = 1$ and the height of this recursion tree is:

$$h = log_3 n \tag{5}$$

- So the total time should be:

$$T(n) = \sum_{i=1}^{log_3 n} 1 = \Omega(log_3 n) \tag{6}$$