Computer Virus Detection
Tom Bohbot
February 2021

1) Brute Force Algorithm:
    a. A brute force algorithm would be to compare each element to every other element in the array and count how many elements it is equal to, and if it is equal to (viruses.length/2) + 1 then return that index. This algorithm would be O(n^2) since it would require a nested for loop to compare each element to every other element.

2) Describe the core idea of your algorithm:
    a. The core idea of the algorithm is that it recursively splits the array of viruses in half until there is only one element left in each sub-array. Once this base case occurs the recursion bottoms out and will merge back each in pairs. While merging back up the algorithm will look for the most prevalent virus in the new merged array through looping through it and checking if either previous most prevalent virus exists as a most prevalent virus, or -1 if it does not. It is important to note that when the base case occurs there is a most prevalent virus in each sub-array since there is only one element per array. Once this algorithm bubbles back up there will be only (at a maximum) two potential most prevalent viruses. The algorithm will then loop through the whole array like it did before and check if either of these two viruses occurs at least (viruses.length/2) + 1 times, and if it does then it returns it's index, otherwise it will return -1.

3) PseudoCode:

```
def mostPrevalent(viruses, virus_checker):
    return recursiveHelper(viruses, virus_checker, min_index_viruses, max_index_viruses)

def recursiveHelper(viruses, virus_checker, min_index, max_index):
    if (min_index == max_index):
        return min_index
    else:
        int first_half_most_frequent = recursiveHelper(viruses, virus_checker, min_index, (min_index + max_index)/2)
        int second_half_most_frequent= recursiveHelper(viruses, virus_checker, ((min_index+max_index)/2)+1, max_index)
        if first_half_most_frequent == -1 && second_half_most_frequent == -1:
            return -1
        if first_half_most_frequent equals -1 then set it equal to second_half_most_frequent
        if second_half_most_frequent equals -1 then set it equal to first_half_most_frequent
        for i = start; i <= end; i += 1:
            count how many viruses equal viruses[first_half_most_frequent] and viruses[second_half_most_frequent]
        half_of_elemnents = ((end-start+1)/2)
        if count of viruses[first_half_most_frequent] > half_of_elemnents:
            return first_half_most_frequent
        if count of viruses[second_half_most_frequent] > half_of_elemnents:
            return second_half_most_frequent
        else:
            return -1
```

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

5) Proof:
   a. Direct Proof:
      i. Necessary Prerequisite Knowledge:
         1. The most prevalent virus can be defined by being the length of viruses divided by two plus one since it must occur more than fifty percent of the array.
      ii. The most prevalent element virus can be found by recursively finding the most prevalent virus in each half of the array. If a most prevalent virus exists, it must be present in at least one of these halves since, by definition, it is greater than fifty percent of the array. Using the pigeon hole principle the most prevalent element must be most prevalent in at least one half of the array since there are viruses.length/2 + 1 viruses and only viruses.length/2 "pigeon holes".
      The way in which the most prevalent virus in each half is deduced is through recursively splitting each sub array in half until there is only one element in each array. At this point each array has a most prevalent virus since there is only one virus in each array. Then as the recursion stacks back up the array will be built back up in pairs. Every time a pair is merged together the algorithm will check what the new most prevalent virus is. The most prevalent virus (if it exists) in the new array must be one of the previously most prevalent viruses. Knowing those two viruses one can loop through the merged array and check if either of those two elements will still be most prevalent. The merged array's most prevalent virus must be one of these two viruses since it is impossible for a different element to be greater than 50% of the merged array since it can only be, at a maximum, present in 49% of each sub array which would only amount to being 49% in the newly merged array. If there is no most prevalent virus then the algorithm will note that as -1. As the recursion bubbles up it will find out which virus, if any, is most prevalent through repeating this process log(n) times.

6) How does implementation conform to your pseudocode:
    a. Besides for edge cases that would be too low level to demonstrate in the pseudocode my code conforms perfectly to the pseudocode as it implements the recursive part of the pseudocode identically. The implementation adds a few edge cases such as null pointers or if the set size is zero, but otherwise mimics the pseudocode exactly.