1.

Attached in Blackboard as hwWritten3Part1.cpp

2. Find the medium Suppose you are given 2 sorted arrays A and B with n and m items respectively, n>=m. Design an O(log n) time algorithm to find the median of all n+m items. Provide pseudo code for your solution along with analysis.

//PseudoCode

1. get size of A and B
2. make sure size of A is greater than size of B by swapping arrays if necessary
3. get the left half of the length
4. set variables for aMinCount which is the first element of A, and aMaxCount which is just the size of A
5. while aMinCount is not greater than aMaxCount
    a. aCount = aMinCount + ((aMaxCount - aMinCount) / 2);
    b. bCount = leftHalfLen - aCount;
    c. if (aCount > 0 && A[aCount - 1] > B[bCount])
        i. aMaxCount = aCount - 1;
    d. else if (aCount < Alen && B[bCount - 1] > A[aCount])
        i. aMinCount = aCount + 1;
    e. else
        i. If Alen + Blen is odd, the median is the greater of x and y.
        ii. else, find the smaller of the two.

//Analysis

Runs in O(log n) time because we do not search for all n, we compare arrays and try our best to just search the left half of the array not the entire thing.

Anyways we only really want the left half of the array because the median will not be at the end of the array or even near the end of the array. We know that values of this leftHalfLen (left half of the array) can have values from A or B or both. We search for the last value that both arrays contribute, compare the two, and the greater of the two is the median.

However the greater of the two last values will not always be the median, so we use the concept of a BinarySearch to help us understand what to do.

Given leftHalfLen of 6,

1. A must contribute at least 1, at most 6 elements to leftHalfLen

2. Consider the midpoint between min and max, check if the median really is equal to mid, if the middle is median than that's our solution, if not, the min becomes mid+1 or the max becomes mid-1, and find the smaller value in an even n+m array.

To make sure contribution sizes do not exceed the left half of the array, we increase/decrease contribution sizes of A respectively.

//Running program
Attached in Blackboard as hwWritten3Part2.cpp
Please run the program, it will help greatly with explanations as to why the analysis/pseudocode works, thank you.

3.

Median Heap requires two heaps, one min heap and a max heap. Each contains ½ of the data. Elements in min heap greater than or equal to median, elements in max heap less than or equal to median. If min heap has 1 more element than max heap, min heap's top is the median. If max heap has 1 more element than min heap, median is at top of max heap. If heaps are the same size, median is mean of the two top elements from the two heaps, or the lesser of the two, or the greater depending on what median definition we are using.

If one heap has more than 1 element more than the other heap, extract min from max heap or max from min heap and insert into the other heap such that the difference between heap sizes is less than or equal to 1. When inserting new elements, elements greater than current median go into min heap and elements less than current median go into max heap.