1

a. Design and analyze an algorithm to compute the number of elements in X + Y in O(n^2 log n) time.

The overall details of my algorithm is for each element in x, we calculate x + y where y is each elements in Y. The addition results will be stored in a list. This step will require O(n^2) time complexity. However, we must remove all the duplicates in the list. To remove duplicates in the list, we construct an array to store the number of unique elements. We start looping through from the first element in the array, if it appears more than once, we simply remove it. This step will take O(n) time.

Below is the pseudo code for my algorithm.

```
def number_of_elements(X, Y) {

#This can be done using python's list method.
# This step takes O(n^2)
Sum = [ x + y for x in X and for y in Y]

# Then, we start removing duplicates
# record number of unique items
Store =[]

 for item in Sum{
# if it is unique,
we put it in store list
If item not in store:
        store += [item]
# if it is not unique, we simply delete it.
If item in Sum:
        delete item in Sum
}
return len(Sum)
}
```

It is obvious that the time complexity for this algorithm is O(n^2) + O(n) = O(n^2) where O(n^2) to construct the Sum list and O(n) to remove duplicates.

b. Design and analyze an algorithm to compute the number of elements in X + Y in O(M log M), where M is the largest absolute value of any element in X ∪ Y

The key to this problem is to reduce this problem to to an easier problem using FFT.
Since we know that FFT is an efficient algorithm to compute DFT in (NlogN)time.
In order to achieve (MlogM), we can generate two samples that are size M from X and Y.
Then, by applying FFT we can calculate the Minikowski sum in M(logM) by applying convolution.