Problem 1: Consider the binary search algorithm we saw in lecture. Suppose *A=(2,4,5,6,9,11,12,15,20)*.

a) Suppose we are looking for the value 12. Which numbers will be compared with 12? How many total comparisons were performed?

12 will be compared with 9 and 12. Two comparisons.

we get to the midpoint of the array. In this case, index 5 value 9 (ceiling function/round up).

Since the midpoint is less than the 12, we look at the right half.(11,12,15,20).

midpoint is 12 and the target is 12. Therefore, the target was found in index 7.

b) Suppose we are looking for the value 16. Which numbers will be compared with 16? How many total comparisons were performed?

16 will be compared to 9, 12, 15 and 20. Four comparisons.

c) Suppose we are looking for the value 0. Which numbers will be compared with 0? How many total comparisons were performed?

0 will be compared with 6 (floor function/round down), 4 and 2. Three comparisons.

d) Suppose now that A=(1,4,3,8,12,11,5) and we are looking for the value 5. Which numbers will be compared to 5? What happens? Why?

5 will be compared with 1,4,3,8,12,11,5. We cannot perform binary search, because when we divide the list into two halves, both halves have a value that is less or equal to 5. The list was not sorted.

Problem 2: Consider the selection sort algorithm we saw in lecture. Suppose *A=(10,5,2,6,20,11)*.

Execute selection sort on *A* and write down what the list looks like after each iteration of the outer loop. How many comparisons were performed before the algorithm terminated?

1- (10,5,2,6,20,11)

3- (2,5,10,6,20,11) five comparisons

4- (2,5,10,6,20,11) four comparison

5- (2,5,6,10,20,11) three comparison

6- (2,5,6,10,20,11) two comparison

7- (2,5,6,10,11,20) one comparison

Total iteration performed before termination are 15

Problem 3: Consider the insertion sort algorithm we saw in lecture. Suppose *A=(12,5,20,6,2,11)*.

Execute insertion sort on *A* and write down what the list looks like after each iteration of the outer loop. How many comparisons were performed before the algorithm terminated?

1- (12,5,20,6,2,11)

2- (5,12,20,6,2,11) one comparison

3- (5,12,20,6,2,11) one comparison

4- (5,6,12,20,2,11) two comparison

5- (2, 5, 6, 12, 20, 11) four comparison

6- (2, 5, 6, 11, 12) two comparison

Total comparison performed before termination are 10

Problem 4: In lecture we talked about the trade-off between using linear search on an unsorted list (the phone book) and sorting the list first and then using binary search. We sort the phonebook because we search it enough times that it's worth the computational expense involved in sorting it. Suppose we have an unsorted list of length 25,000. How many worst case searches would be necessary for it to make it worth it to first sort the list with selection sort so that we could then use binary search instead of linear search.

If we consider linear search worst case would be 25000 for one search

if we consider binary search (unsorted) would be (log n base 2) +1= 14.61 or 15 for one search.

If we consider selection sort search would be n(n-1)/2 = 312500000 for one search

Problem 5: Design an algorithm that takes a list of integers as input *A=(a1, a2, a3, a4, a5, . . . , an)* along with a target value *x*. Your algorithm must locate an ordered pair of integers in *A* whose difference is the value *x* or report that there is no such pair. For example if *A=(1,11,-2,12,8,9)* and *x=10*, then your algorithm should report back either (11,1) or (8,-2). Write your algorithm out in pseudocode (bulleted english is fine).

- start
- Input is a list of integers A[2,3,6,1,8,9];
- Target is the integer 2;

  Outer loop

- For each element on the list A where A[i] is the first element on the list, we will increment i or we will change i index from 2 to 3 and 3 to 6 and so on (i++). until i is less or equal to the length of the list.
- Every time we increment i the previous i will be dropped from the list.

  Inner loop

- For each element after i , j = i+1, increment j until j is equal or less than A.length;
- If (i - j = 2) or (j - i = 2);
- Display the pair was found at index A[i] and A[j];
- Else
- display pair was not found;
- End