PreLab Part 1

1. I think it will take at least 10 swaps
2. The worse case scenario of bubble swap would be if everything is in backwards order, so it will depend on how many elements in the array. O(n^2).

PreLab Part 2:

1. The worst time complexity for Shell sort depends on how big the gap is between first element and last element but it would be O(n^2).
2. 2. I would use a while loop and update it every time so I won't have to continuous iterate again.

PreLab Part 3:

1.Quicksort worst case complexity occurs if an array is already sorted which creates a large complexity time. The good news is that it is already sorted, so it isn't completely doomed.
https://www.geeksforgeeks.org/when-does-the-worst-case-of-quicksort-occur/

PreLab Part 4:

1. The time complexity would be the same of O(n^2) at worst case complexity but the binary search itself would be O(log2n) with insertion..

DESIGN

Main:

// Must input one of the case flags at command line
while(getop(arguments) !=-1):

Case A:
    Store letter A in array
Case b:
    Store letter b in array
Case s:
    Store letter s in array
Case i:
    Store letter i in array
Case p:
    Get number in optarg which will be amount to print
Case r:
    Get number for srand in optarg
Case n :
    Get number for elements number

Allocate pointer to array;
Set srand()

```
For elements in the Letter array:
        If letter = ‘b’ or ‘A’:
                For i in range(0,array size)
                        rand() (less than 30 bits)
                        array[i]=random number

                Bubble_sort(array,array size)
                Print sorted array;
        If letter = ‘s’ or ‘A’:
                For i in range(0,array size)
                        rand() (less than 30 bits)
                        array[i]=random number

                Shell_sort(array,array size)
                Print sorted array;

        If letter = ‘q’ or ‘A’:
                For i in range(0,array size)
                        rand() (less than 30 bits)
                        array[i]=random number

                quick_sort(array,array size)
                Print sorted array;

        If letter = ‘i’ or ‘A’:
                For i in range(0,array size)
                        rand() (less than 30 bits)
                        array[i]=random number

                binary_sort(array,array size)
                Print sorted array;
Print everything including headers

        END
```

// This sort goes through the whole array and checks adjacent values. It swaps and compares accordingly to determine which is smaller number

```
Bubble_sort(array,length):
        For range(i,length):
                If current element is less than element -1:
                        Count comparisons
```

```
                    Swap
                    Count swap
        return
```

// This sorting method finds the gaps between elements and switches from that position. It then halves every time

```
Shell_sort(array,length)

While gap(length) !=1:
        For range(gap,length):
                For range(i,gap-1,-gap):
                        If array[element]<array[element-gap])
                                Swap
                                Count swaps and comparisons
        length/=2;

Gap(length):
        If length <=2 : return 1
        Else:
                Return 5*n/11
```

//Quick sort separates the array into two, and then sorts each one and then combines.

```
Quick_sort(array,length,index 0, index last):
        If index 0<index last:
                index=Partition(array,index 0, index last)
                Quick sort(“,”,”index-1)
                Quick sort(“ , “ , index+1, “)

Partition(index 0, index last):
Create a pivot point which is first element
lo=index0
hi=index last
        while(true):
                While lo <= hi & array[hi] >= pivot point:
                        hi-=1
                While lo <= hi & array[hi] <= pivot point:
                        lo+=i;
                If lo < hi:
                        Swap
                        Count swaps and comparisons
```

```
            Break;
```

//Binary sort is an insertion that finds the middle element and compares it with other and then places at the right location.

```
Binary Sort(array):
    For i in range(1,length):
        l=0
        r=i
        while(l<r):
            Get mid point
        If array[i] >mid point value:
            l=mid+1
        Else:
            r=mid

        For j in range(i,length,-1):
            Swap
            Count swap and comparisons
```