Pre-lab part 1

1. You need 10 rounds of swapping to sort the numbers
    a. 8 7 9 22 5 13 31 (rounds 1-4)
    b. 7 8 9 5 13 22 31 (rounds 5-7)
    c. 7 8 5 9 13 22 31 (round 8)
    d. 7 5 8 9 13 22 31 (round 9)
    e. 5 7 8 9 13 22 31 (round 10)
2. 15 comparisons would be expected in the worst case scenario for bubble sort example from question one.
    a. N! Is the worse case comparison for any n number of integers in an array

Pre-lab Part 2

1. The worst time complexity for shell short is always less than or equal to O(n2).
    a. Using the Poonen theorem, the worst time complexity is between O(NlogN)2/(loglogN)2) or O(NlogN)2/log logN or O(N(logN)2)
2. I can't really find ways to improve the runtime other than improving the gap which is already pretty efficient. I would try to implement the pratt gap sequence that the professor mentioned in order to reduce the runtime of this sort.
    a. https://stackoverflow.com/questions/25964453/how-do-i-implement-the-pratt-gap-sequence-python-shell-sort

Pre-Lab Part 3

1. The worst case scenario for the time complexity of quicksort is O(n^2) and that happens when the picked pivot is an extreme(smallest or largest number in the array). In the case of a worst case scenario, the time complexity can be reduced down to O(nLogn) simply by choosing a different pivot that's not an extreme. The general idea is that you find the medians of the numbers within the array and set that as the pivot to start your quick sort.
    a. https://www.geeksforgeeks.org/can-quicksort-implemented-onlogn-worst-case-time-complexity/

Pre-Lab Part 4

1. Combining the binary search with the insertion sorting algorithm reduces the comparison numbers. Since the binary search starts the comparisons at the halfway point which reduces the comparisons. The number of comparisons drop to O(logn) if you include the binary search and if you don't include binary search the number of comparisons is O(n). The number of shifts stays as O(n^2) and the total algorithm stays as O(n^2).

Pre-Lab Part 5

1. I would make a variable called swap and compare which i would increment whenever my sorting algorithm would move a variable or compare the variables in order to move it. I would also have these variables within my sorting files and print them after the function has finished sorting.

Sorting.c

This function prints the values that are sorted and gets user input using getopt()
Void Main (argc, **argv) {
    Initializing the seed to 8222022
    Initializing the size for 100
    Initializing the seed value for 100
    Get opt function
        Switch statement
            Case for all the sortings
            Case for bubble sort
            Case for shell sort
            Case for quick sort
            Case for binary insert sort
            Case input for the amount of values to print
            Case input for the seed value
            Case input for the size of the array

    If the size is greater than print, print value equals to size

    Calls array_create with seeds and size
    Calls bubble sort
    Prints the value in the array
    Frees the array

    Calls array_create with seeds and size
    Calls shell sort
    Prints the value in the array
    Frees the array

    Calls array_create with seeds and size
    Calls quick sort
    Prints the value in the array
    Frees the array

    Calls array_create with seeds and size
    Calls binary insertion sort
    Prints the value in the array
    Frees the array

Returns 0
}

This function creates an array and fills it with random numbers and returns it

```
uint32_t* array_create(seed, size) {
        Set the seed with the parameter seed using srand
        Calloc the size+1 * uint32_t into a uin32_t* variables called array
        For i in range(size) {
                Array[i] = random number using rand() & 0x3FFFFFFF
        }
        Return array
}
```

Bubble.c

Parts of Pseudo code by DDEL

This function sorts the array using bubble sort using the values within the array

```
def  Bubble_Sort(arr):
  Counts the size of the array
  for i in range(len(arr) - 1):
    j = len(arr) - 1
    while j > i:
      compare++
      if arr[j] < arr[j - 1]:
        Swap++
        arr[j], arr[j - 1] = arr[j - 1], arr[j]
      j -= 1
     Print header statements for bubble sort and the value
for move and compare and the size of the array
     Return
```

shell.c

Parts of Pseudo code by DDEL

This function returns the array of gaps between numbers in the array

```
def gap(n):
  Create a newarray with a 100 size with uin32_t
  while n > 1:
    Input n = 1 into the array if n  <= 2 else input 5 * n //
11 into the newarray
  Counts the size of the new array
  Create a array with allocating the size of the newarray
with uint32_t
```

```
  Inputs the value of the new array into the array
  Free the newarray
  Return array
```
This function sorts the array using shell sort using the values within the array
```
def  Shell_Sort(arr):
  Count the size of the arr
  Uint32_t* v = gap(arr)
  for  step in gap(len(arr)):
    for i in range(step , len(arr)):
      for j in range(i, step - 1, -step):\
       compare++
       if arr[j] < arr[j - step]:
         swap++
         arr[j], arr[j - step] = arr[j - step], arr[j]
  Print header statements for shell sort and the value for
move and compare and the size of the array
  Free v
  Return
```

quick.c

Parts of Pseudo code by DDEL

This function returns the pivot point of the array
```
def  Partition(arr , left , right):
  pivot = arr[left]
  lo = left + 1
  hi = right
  while  True:
    while lo  <= hi and arr[hi] >= pivot:
      hi  -= 1
    while lo  <= hi and arr[lo] <= pivot:
      lo += 1
    compare++
    if lo  <= hi:
      swap++
      arr[lo], arr[hi] = arr[hi], arr[lo]
    Else:
      Break
```

```
  swap++
  arr[left], arr[hi] = arr[hi], arr[left]
  return  hi
```

This function quick sorts the array using the values within the array

```
def  Quick_Sort(arr , left , right):
  if left < right:
    index = Partition(arr , left , right)
    Quick_Sort(arr , left , index - 1)
    Quick_Sort(arr , index + 1, right)
  Counts the size of the array
  if left==0 && right == n-1
     Print header statements for quick sort and the value for
move and compare and the size of the array
  return
```

binary.c

Parts of Pseudo code by DDEL

This function binary insertion sorts the array using the values within the array

```
def  Binary_Insertion_Sort(arr):
  for i in range(1, len(arr)):
     value = arr[i]
     left = 0
     right = i
     while  left < right:
       mid = left + ((right - left) // 2)
       if value  >= arr[mid]:
         left = mid + 1
       Else:
         right = mid
     for j in range(i, left ,  -1):
       arr[j - 1], arr[j] = arr[j], arr[j - 1]
  return
```