Purpose of the lab:

We will implement bubble sort, quick wort, and quick sort(stack and queue method) in this lab. We mainly just translate the codes from python, and we need to write the .c files for stack and queue. We need to analyze the moves and compare steps and analyze the big O time complexity for each sort in writeup pdf.

Command options:

-a run 4 sorts
-b run bubble sort
-s run shell sort
-q run quick sort (stack)
-Q run quick sort (queue)
-r set seed
-n set size
-p set elements

Files:

Bubble.h and bubble.c -bubble sort program
Gaps.h, shell.h, shell.c -shell sort program
Quick.h and quick.c -quick sort program
Stack.c and .h + queue .h and queue.c helper files for quick sort
Set.c and set.h are helper file for main function
Sorting .c - min function

Prelab:

Part 1:

1. 7 rounds, it will run n-1 times.
2. The worst time case would be inverse order. For example 5, 4, 3, 2, 1, so it needs to compare and move for all numbers in order to move to the biggest number to the left.

Part 2:

1. The worst time is o(n^2), if the gap is 1(meaning split whole data to 1 group and sort), then it will be the worst case(you will find out that, it's actually an insertion sort). So different gaps will have different performance based on the data. However, if we improve our gap based on the data(for example, make sure gaps are decreasing until 1), it will be much more efficient.

Part 3:

1. Usually, quicksort is a fast sort, however, if the pivot is the smallest number in the array, like 0, then it doesn't help at all to sort. In contrast, if the pivot is 0, it wastes one round to compare and move and slows down the run time.
   Cite: https://www.geeksforgeeks.org/when-does-the-worst-case-of-quicksort-occur/

Part 4:

1. I will use global variables (use extern) to keep track of moves and comparisons.

Pseudocode:

Swap two elements:

```
Temp = *a
*a = *b
*b = temp
```

Collecting data:

Extern move and compare in .h

Declare move and compare in .c(for example: extern b_move in .h file, b_move = 0 in .c file, and use b_move directly in main.c to print it out)

When swapping two element in sort: 3 moves, because temp = a, a = b, and b = temp for 3 moves

When to count compare: when checking A[]

Stack:

Stack create: allocate memory with malloc first , then ini top capacity and items with calloc

Stack delete: free stack item, then stack pointer, and s = null

Stack_empty: if top is 0 return true

Stack full: if capacity = top return true

Stack size: return top

Stack push: if stack_full return false, if not items[top]=x, top++ and return true

Stack pop: if stack_empty return false, if not top-- and *x = item[top]

Queue:

Queue create: allocate memory with malloc first , then ini head tail size capacity and items with calloc

queue_delete:free stack item, then stack pointer, and s = null

queue_empty: if size is 0 return true

queue full: if capacity = size, return true

Queue size: return size

Enqueue: if queue_full return false, else return true and items[head] = x, size++, head++, head%= capacity

Dequeue: if queue empty, return false, else *x = items[tail], size --, tail ++, tail % = capacity

Main:

Typedef enum sorting: bubble = 0, shell = 1 stack = 2 queue =3

Case a = set complement

Case (the rest) = set insert

```
For sorting variable = bubble; variable < = queue; variable ++
If (setmember()){
switch(variable)
Case bubble:
Case ......
```