Purpose

This program contains a collection of comparison-based sorting algorithms. The user specifies which sorting algorithms to enable and how to display them using command-line options.

Pre-lab

1.1 How many rounds of swapping will need to sort the numbers 8,22,7,9,31,5,13 in ascending order using Bubble Sort?
Incomplete for now.

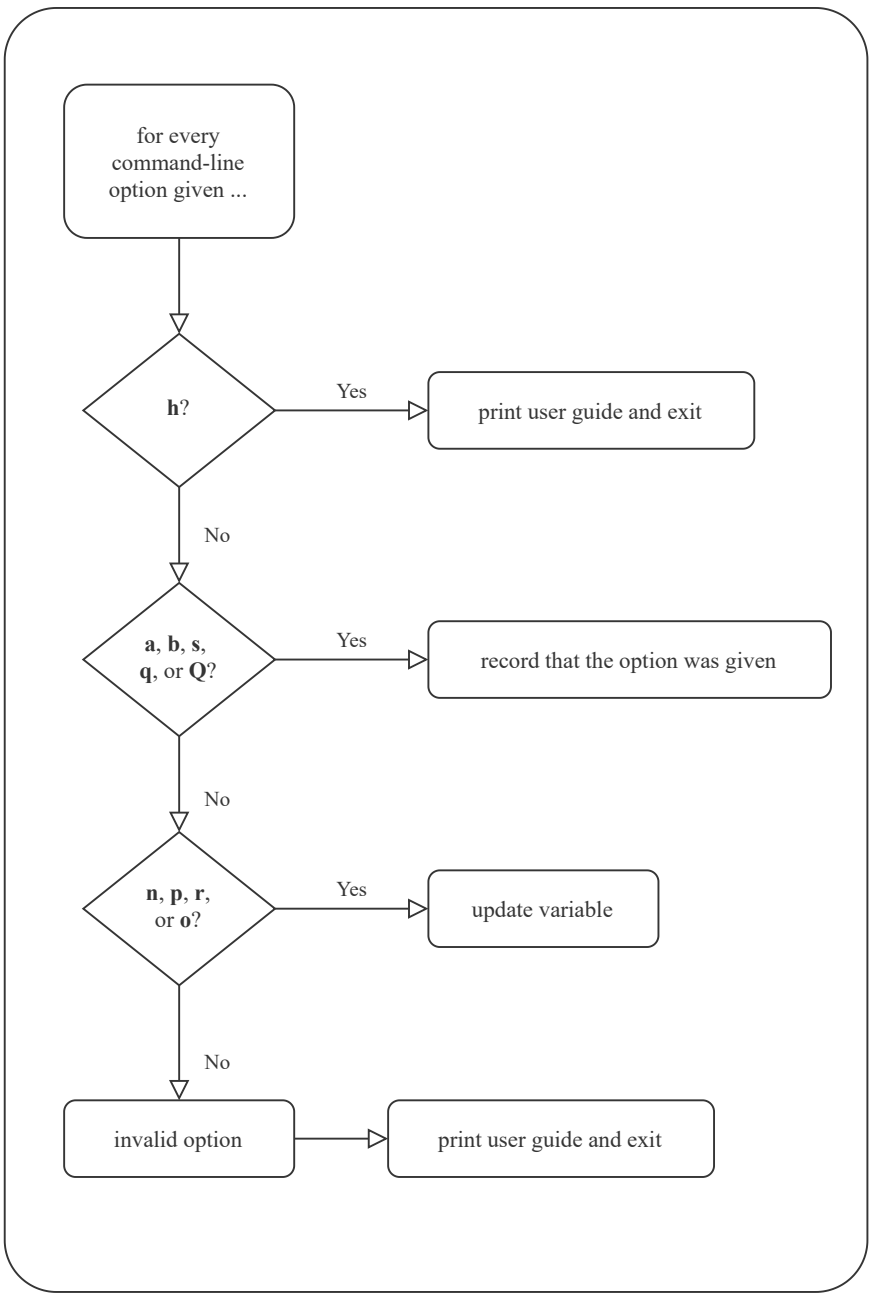
1.2 How many comparisons can we expect to see in the worse case scenario for Bubble Sort? Hint: make a list of numbers and attempt to sort them using Bubble Sort.
The maximum number of comparisons is
(n - 1) + (n - 2) + (n - 3) + (n - 4) + (n - 5) … 3 + 2 + 1
= n(n - 1) / 2

2.1 The worst time complexity for Shell Sort depends on the sequence of gaps. Investigate why this is the case. How can you improve the time complexity of this sort by changing the gap size? Cite any sources you used.
Incomplete for now.

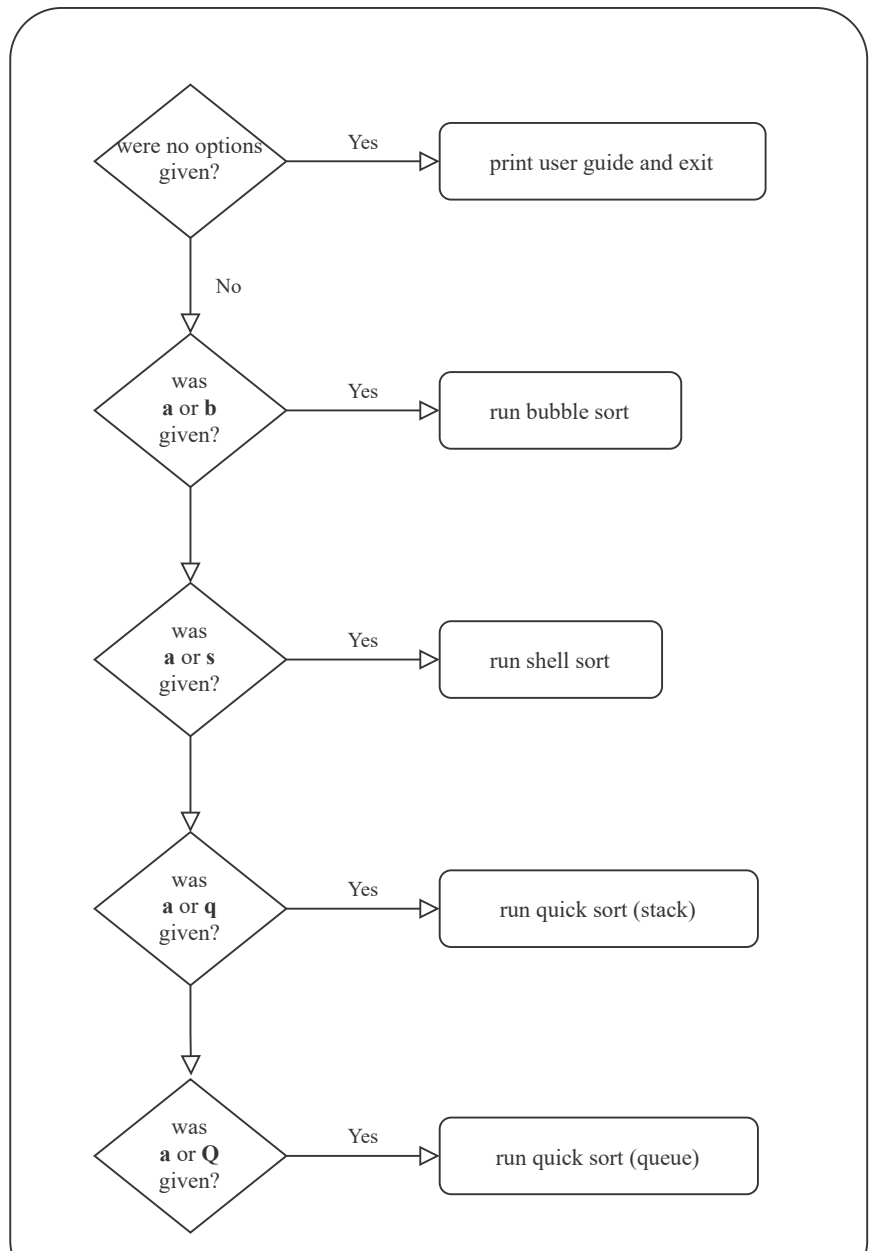
3.1 Quicksort, with a worse case time complexity of O(n2), doesn't seem to live up to its name. Investigate and explain why Quicksort isn't doomed by its worst case scenario. Make sure to cite any sources you use.
Incomplete for now.

4.1 Explain how you plan on keeping track of the number of moves and comparisons since each sort will reside within its own file.
Each sort is run in its own file, and as far as I can tell, I can keep track of the number of moves and comparisons in that same file.

Parse the command-line options



Run the appropriate sort algorithms



Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    swapped = True
    while swapped:
        swapped = False
        for i in range(1, n):
            if arr[i] < arr[i - 1]:
                arr[i], arr[i - 1] = arr[i - 1], arr[i]
                swapped = True
        n -= 1
```

Quick Sort (Stack)

```
def quick_sort_stack(arr):
    lo = 0
    hi = len(arr) - 1
    stack = []
    stack.append(lo) # Pushes to the top.
    stack.append(hi)
    while len(stack) != 0:
        hi = stack.pop() # Pops off the top.
        lo = stack.pop()
        p = partition(arr, lo, hi)
        if lo < p:
            stack.append(lo)
            stack.append(p)
        if hi > p + 1:
            stack.append(p + 1)
            stack.append(hi)
```

Shell Sort

```
def shell_sort(arr):
    for gap in gaps:
        for i in range(gap, len(arr)):
            j = i
            temp = arr[i]
            while j >= gap and temp < arr[j - gap]:
                arr[j] = arr[j - gap]
                j -= gap
            arr[j] = temp
```

Quick Sort (Queue)

```
def quick_sort_queue(arr):
    lo = 0
    hi = len(arr) - 1
    queue = []
    queue.append(lo) # Enqueues at the head.
    queue.append(hi)
    while len(queue) != 0:
        lo = queue.pop(0) # Dequeues from the tail.
        hi = queue.pop(0)
        p = partition(arr, lo, hi)
        if lo < p:
            queue.append(lo)
            queue.append(p)
        if hi > p + 1:
            queue.append(p + 1)
            queue.append(hi)
```