Rahul Vaidun
rvaidun@ucsc.edu
1 April 2021

CSE 13S Spring 2021
Assignment 3: Sorting: Putting your affairs in order
Design Document

I. Prelab

Part 1:

1. **How many rounds of swapping will be needed to sort the numbers?**
   8,22,7,9,31,5,13 in ascending order using Bubble Sort
   Round 1: 8, 7, 9, 22, 5, 13, 31
   Round 2: 7, 8, 9, 5, 13, 22, 31
   Round 3: 7, 8, 5, 9, 13, 22, 31
   Round 4: 7, 5, 8, 9, 13, 22, 31
   Round 5: 5, 8, 8, 9, 13, 22, 31
   It will take 5 rounds of swapping
2. **How many comparisons can we expect to see in the worse case scenario for Bubble Sort?**
   In the worst case scenario there will be n^2 comparisons where n is the total amount of items in the array

Part 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?**
   The worst case time complexity for Shell Sort depends on the sequence of gaps because the higher the initial gap the longer it takes to reduce. The gap between the items continually reduces but if you start with a higher gap it will take longer to reduce than if you start with a lower gap

Part 3:

1. **Quicksort, with a worst case time complexity of O(n^2) doesn't live up to its name. Investigate and explain why Quicksort isn't doomed by its worst case scenario.**
   Quicksort isn't doomed by its worst case scenario because by changing the algorithm used to find the pivot we can reduce quicksort time complexity. Using random pivoting helps reduces the expected time complexity to $O(n \log n)$
   I used GeeksforGeeks with help on this problem
   https://www.geeksforgeeks.org/quicksort-using-random-pivoting/

Part 4:

1. **Explain how you plan on keeping track of the number of moves and comparisons since each sort will reside in its own file**
   I will keep track of the number of moves and comparisons by adding an extra header file to keep track of these integers. I can then utilize the extern keyword to reference the variables outside of the file

II. Pseudocode

The pseudocode for all of the sorting algorithms is given in the assignment by Professor Darrel Long. However the pseudocode in the design pdf does not show how to implement data collecting. Statistics for moves can be collected by incrementing the `moves` variable by 1 each time an array element is copied. To find the total number of comparisons, increment the `compares` variable every time a comparison has been made. The stats and compares variables should be defined globally in the test harness. In the .c files for each swap it is important to globally extern those variables since they are being defined in the test harness. The top of the source files for the sorting algorithms should look like:

```
extern uint32_t moves, compares;
```

Implementation of a stack. Pseudocode written in python.

```
class Stack:
 def __init__(self, capacity):
   self.capacity = capacity
   self.top = 0
   self.items = [];

   def stack_empty(self):
       return self.top == 0;

   def stack_print(self):
       print(self.items)

   def stack_full(self):
       return self.top == self.capacity

   def stack_size(self):
       return self.top

   def stack_push(self, x):
       if (self.stack_full()):
           return False
       self.items.append(x)
       self.top += 1
       if max_stack_size < self.size: # Will need to define
max_stack_size outside
           max_stack_size = self.size
       return True

   def stack_pop(self, x):
       if (self.stack_empty()):
           return False
       self.top -= 1
       x = self.items.pop()
       return True
```

Implementation of a queue

```
class Queue:
    def __init__(self, capacity):
        self.head = 0
        self.tail = 0
        self.size = 0
        self.capacity = capacity
        self.items = []

    def queue_empty(self):
        return self.size == 0
```

```python
def queue_full(self):
    return self.size == self.capacity

def queue_size(self):
    return self.size

def enqueue(self,x):
    if (self.queue_full()):
        return False
    self.items.append(x)
    self.tail = (self.tail + 1) % self.capacity
    self.size += 1
    if max_queue_size < self.size: # Will need to define
max_queue_size outside
        max_queue_size = self.size
    return True

def dequeue(self, x):
    if (self.queue_empty()):
        return False
    x = self.items.pop(0)
    self.head = (self.head + 1) % self.capacity
    self.size -= 1
    return True

def queue_print(self):
    print(items)
```

Set

The program utilizes sets when dealing with command line arguments. The implementation of sets are simply multiple bitwise functions on a 32 bit integer. The maximum number of elements in the set is 32. To add a number to the set you do bitwise OR. To check if an element is a member of a set use bitwise AND. The source code for the set file has been posted by Eugene Chou on Piazza.

Below is the implementation of a test harness in python

```python
def main():
   print_elements = 100
   seed = 13371453
   size = 100
   moves = 0
   compares = 0
   max_stack_size = 0
   max_queueu_size = 1
   s = set()
   func_arr = [bubble_sort, shell_sort, quick_stack,
quick_queue]
   try:
     opts, args = getopt.getopt(sys.argv[1:],"habsqQr:n:p:",
["help"])
   except getopt.GetoptError:
       print_help()
       sys.exit(2)
   for o, a in opts:
       if o == "-h":
           print_help()
           sys.exit(2)
       elif o == 'a':
           print("came in")
           s.add(Sorts.BUBBLE)
           s.add(Sorts.SHELL)
```

```
            s.add(Sorts.QUICK_STACK)
            s.add(Sorts.QUICK_QUEUE)
        elif o == '-b':
            s.add(Sorts.BUBBLE)
        elif o == '-s':
            s.add(Sorts.SHELL)
        elif o == '-q':
            s.add(Sorts.QUICK_STACK)
        elif o == '-Q':
            s.add(Sorts.QUICK_QUEUE)
        elif o == '-r':
            seed = int(a)
        elif o == '-p':
            print_elements = int(a)
        elif o == '-n':
            size = int(a)
        else:
            print_help()
            sys.exit(2)
    random.seed(seed)
    arr = []
    for i in range(size):
        arr.append(random.random())

    for sort in s:
        sorting = arr.copy()
        func_arr[sort.value](sorting,len(sorting))
        print(sort)
        print(f"{size} elements, {moves} moves, {compares}
compares")
        if (sort == Sorts.QUICK_STACK):
            print(f"Max Stack Size {max_stack_size}")
        if (sort == Sorts.QUICK_QUEUE):
            print(f"Max Stack Size {max_queue_size}")

        for i in sorting:
            if (i % 5 == 0 and i != 0):
                print()
            print("%13d" % i, end='')

            if (i == print_elements - 1):
                print()

if (__name__ == '__main__'):
    main()
```