Winston Yi
wyi10@ucsc.edu
4/23/2021

CSE13s Spring 2021
Assignment 3: Putting your affairs in order
Design Document

PRELAB:

Bubble:

1. 5 Rounds
2. Big O is *O(n^2)*

Shell:

1. Big O depends on the gaps because too small a gap sequence slows down the pass throughs the array when sorting, and large gaps create big sections where nothing was sorted into sections properly. By changing the gap sequence, we can get the time complexity to be much lower. (https://en.wikipedia.org/wiki/Shellsort)

Quicksort

1. Quicksort rarely ever meets the worst case because the chances of it happening in a large array are very slim. If you pick the pivot at random, choosing the worst pivot every time is incredibly rare. Thus, if we go by the average case, which is *nlog2(n)* (https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort) (https://en.wikipedia.org/wiki/Quicksort)

Overall:

1. I plan to keep track of moves and comparisons with a separate header file which will contain global variables that keep track of the moves and compares. This file (compare.c/h) has tools to increment moves and compares as well.

OVERVIEW:

For this assignment, I have created a library that will sort an array. There are four options of sorting available: Bubble sort, Shell sort, Quicksort with a stack, and Quicksort with a queue. In addition to the four sorting functions, I have also created a test harness that will allow the user to test various pseudorandom arrays.

TOP LEVEL DESIGN:

The top level design is shown by the following pseudocode:

```
set.c overview:

check_bit: checks if the bit has been flagged (1)

set_bit: sets the bit to 1

00000 means no flags, 00001 means one flag, etc
```

```
compare.c overview:

swap: swaps two items in an array

compare: compares two int64's

zero: zeroes moves and compares
```

```
sorting.c overview: Sorts a pseudorandom array

main:
  flags = 0;
  loop through program arguments
    flag each argument based on the bit it occupies in flags
    save npr arguments from the program arguments stored in optarg
  set default npr values
  bound p
  iterate through the sorting types (some of the flags)
    switch case to print each sorting type and the sorted array
  handle no arguments given
```

```
shell.c overview:
iterate through pratt sequence
  range = gap_value, and iterate through gap_values < n
    set two temp values, A[range] and gap_value
    set j to range
     while another_var >=gap value and gap value < A[range]
      swap A[j] with A[j-gap value]
      j -= gap value
    set A[j] to A[range]
```

```
bubble.c overview:
start with first index value
if the next index value is greater, go with that one
if not, swap em
continue doing that until you reach EoA
repeat
```

```
overview of quick.c: partition function: finds the correct place in
the array for a specified pivot, moves the pivot there, and returns
the index
set high and low index values to i,j
while i<j
  do increment i
  while A[i] < pivot
  do j -= 1
  while A[j]> pivot
  if i < j
    swap A[i] with A[j] (or A[j] with A[i], however you want   to
think of it)
returns j (index of pivot)
```

```
overview of stack.c: contains many functions
struc Stack:
  top (of stack)
  capacity
  *items
create_stack: creates a stack
  allocate memory for stack
  set top to 0
  set capacity to capacity
  allocate memory for items
stack_delete: deletes stack
  free() em
stack_push: pushes
  top = item pushed
  top += 1
stack_pop: pops
  decrement top
  pointer to x = top
stack_size: returns size
stack_empty: returns T/F if empty
stack_full: returns T/F if full
stack_print: prints stack
  iterate through array
  if not top, print item
```

```
overview of queue.c: contains many functions
 struct Queue:
  head, tail, size, capacity, items
queue_create: same as stack, except head, tail, size = 0
queue_delete: same as stack
queue_print: same as stack
queue_size/empty/full: same as stack
queue_enqueue: same as stack, but make sure to % the q.tail + 1
over capacity to account for the wrap around
queue_dequeue: same as enqueue, but with head instead of tail
```

DESIGN PROCESS:

At first I attempted to learn each sort individually and to follow the pseudocode as a guideline. This was a huge mistake. Not following the pseudocode would lead to very

different results. Thus, I edited all of my sorts to follow the pseudocode exactly which led to great success. After getting the bubble sort to work, I learned how to implement stacks and queues in C. Although I had previous knowledge of stacks and queues, implementing it in C required knowledge of header files, structures, pointers and arrays, how to dereference, data types, and more. After properly making stacks and queues, I then implemented shell sort, then quick sort with both types.

FINAL THOUGHTS:

My initial design for the sorts were off and had vastly different values for the moves and sorts from the design pdf. Reflecting, I believe the sorts to have been incorrect. I have learned about functions, data types, arrays, and allocation in this lab. However, I do believe this lab is too much for the early assignments. There were numerous things to learn and implement that should have been spread over at least two assignments or two weeks. Although the sets code was given, it was given after I finished it myself, leading to massive progress loss.