Assignment 4 Design Document

Overview:
Implement 4 sorting algorithms shell, bubble, quick, and heap sorts. There will also be a sorting.c file including a main function that will run test cases on the spring algorithms.

shell.c:

```
shell_sort(array):
    n = length(array)
    while n > 1:
        if n >= 2: n = 1
        else: n = (n*5)/11
        for i from n to length(array):
            j = i
            temp = array[i]
            while (j >= n and temp < array[j - n]:
                array[j] = array[j - n]
                j -= n
            array[j] = temp
```

bubble.c:

```
bubble(array):
    for i from 0 to length(array)-1:
        swapped = False
        for j from length(array)-1 to i:
            if array[j] < array[j-1]:
                swap array[j] and array[j-1]
                swapped=True
        if not swapped: break
```

quicksort.c:

```
SMALL = 8
quicksort(array):
    if length(array) < SMALL:
        shellsort(array)
        return
    pivot = (a [0] + a[len (a) // 2] + a[ -1]) // 3
    move all items greater than pivot to the left
    quicksort all moved items
    move them to the right
    move all items less than the pivot to the left
    quicksort them
```

heapsort.c:

```
l_child (n) :
        return 2 * n + 1

r_child (n) :
        return 2 * n + 2

parent (n) :
        return (n - 1) // 2

up_heap (a, n) :
        while n > 0 and a[n] < a[ parent (n) ]:
                a[n] , a[ parent (n) ] = a[ parent (n) ] , a[n]
                n = parent (n)

down_heap (a, heap_size ) :
        n = 0
        while l_child (n) < heap_size :
                if r_child (n) == heap_size :
                        bigger = l_child (n)
                else :
                        bigger = l_child (n) if a[ l_child (n) ] < a[ r_child (n) ] else r_child (n)
                if a[n] < a[ bigger ]:
                        break
                a[n] , a[ bigger ] = a[ bigger ] , a[n]
                n = bigger

build_heap (a) :
        heap = [0] * len(a)
        for n in range (len(a) ) :
                heap [n] = a[n]
                up_heap (heap , n)
        return heap

heapsort (a) :
        heap = build_heap (a)
        sorted_list = [0] * len (a)
        for n in range (len (a) ) :
                sorted_list [n] , heap [0] = heap [0] , heap [len(a) - n - 1]
                down_heap (heap , len(a) - n)
        return sorted_list
```

sorting.c:

```
main(arguments):
        seed = 13371453
```

size = 100
print_size = 100
take arguments and run associated code:
-a: run test for all sorting algorithms
-s: enable shellsort tests
-b: enable bubblesort tests
-q: enable quicksort tests
-h: enable heapsort tests
-r: seed = optarg
-n: size = optarg
-p: print_size = print_size
-H: print out program usage

set random seed
generate random array of size size
test all enabled sorting algorithms on copies of the array and print the amount of swaps and compares and print the array entries to print_size