CSE 13S Assignment 4 - All Sorts of C Code

Jaren Kawai - jkawai

- General Idea
    - Implement various sorting algorithms in C given Python code. Must take in arguments that determine what algorithm is used, seed used to generate values that are to be put into the original arrays, the length of the original arrays, and whether or not the sorted values in the array are to be printed.
- Bubble Sort
    - For loop (i) from 0 to length of array -1
        - Swapped variable to 0
        - For loop (j) from length of array -1 to i
            - If value at array[j] is less than array[j-1]
                - Swap values
                - Set swapped to 1
        - If swapped == 0
            - Break out of loop
- Shell Sort
    - Set temp variable
    - Gap function
        - If n == 1
            - Return 0
        - If n > 2
            - Return 5n/11
        - Return 1
    - For loop starting at first gap to when gap equals 0 gap equals next gap
        - For loop (i) starting at gap to length of passed in array
            - temp variable equals value of array at value i
            - Set new variable (j) to value of i for reference later
            - While j is greater than gap and temp is less than arr[i-gap]
                - Array at j equals array at i-gap
                - Subtract gap from j
            - Array at i equals temp
- Quick Sort
    - Set variables for left, right, and middle arrays
    - If length of array is less than 8
        - Shell sort
    - Pivot equals middle element
    - Put pivot at front of array
    - Set variable for index of pivot to 0
    - For loop (i) from 1 to length of passed in array

            - If array at i is less than pivot
                - Swap pivot and array at i
        - Put pivot in the middle
        - Recursively call quick sort for left side and right side
- Heap Sort
    - Left child
        - Return 2*n+1
    - Right child
        - Return 2*n+2
    - Parent
        - Return (n-1)/2
    - Up heap
        - While n > 0 and array at n is less than array at parent
            - Swap arr[n] and arr[parent]
            - n = parent
    - Down heap
        - Set variable n to 0
        - Set variable bigger
        - While left child is less than heap size
            - If right child is equal to heap size
                - Bigger equals left child
            - Else if left child is less than right child
                - Bigger equals left child, otherwise right child
            - If array at n is less than array at bigger
                - Break
            - Swap array at n and array at bigger
            - n = bigger
    - Build heap
        - Pass in heap array and original array
        - For loop (i) between 0 and n
            - heap[i] = array[i]
            - Up heap (heap, i)
    - Heap sort
        - Malloc for heap array
        - Build heap with heap, array, and number of elements
        - Sorted list at n equals heap[0]
        - For loop (i) between 0 and number of elements
            - Set array element at i to first element in heap
            - Heap[0] is set to last element - i - 1 (work from back)
            - Down heap rest of heap

- Sorting test harness
    - Array copy function
        - Pass in populated array and empty array
        - Put all elements from populated array into empty array
    - Initialize instance of stats and set
    - Enum sorting options
    - Set options for each test/option
    - Set default values for array length, print length, and seed
    - If extra argument given after r, set seed
        - Check if seed given is greater than 0
    - If extra argument given after -n, create for loop that creates random numbers accordingly
        - Check if argument is greater than 0 and less than 250,000,000
    - If extra argument given after -p set print length to argument
        - Check if argument is greater than or equal to 0
    - Create switch case for each sort function
    - At each sort function, add sort to set
    - Use malloc to allocate memory for array
    - Populate using for loop
        - mtrand() & 0x3FFFFFFF to bit mask to correct amount of bits
    - For loop from 0 to 4 to represent each of the 4 sort functions
        - Check membership in set using function
        - If member is found, call appropriate function
            - For each function
                - Create copy of array using function from above
                - Call sort function passing in stats, copy array, and length
                - Print elements of sorted array depending on user defined constraints
                - Free memory for copied array
        - Free memory for original array