lsIsaac Flores
Prof. Miller
CSE13S
October 23, 2022

General Idea:

Sorting.c: Take user input on the command line and set the size, seed, and p elements to the user's input. If not set to the default value. Check which sorting options the user wants and insert a different number in a set for each sorting method. Create an array and dynamically allocate (size * uint32_t bytes) amount of memory to it. Insert random numbers into each element of the array. Run through the set and perform the respective sorting method if the user wants it. Print out the name of the sorting method, the array size, the moves needed and the compares needed to perform the sorting task. Also print out p elements of the sorted array. Free the array and end the program.

Bubble.c: For loop i size amount of times. Each time begin by setting a variable that checks if elements have been swapped to 1. For loop from the top of the list to i in increments of -1. Check if the value of the jth element of the array is less than the element before it. If it is, swap the elements and set the swapped variable to 0. If at the end of the inner for loop, no swaps have occurred, the array is sorted. If not, continue until each value has bubbled up.

Quick.c: Set a pivot that is equal to the approximate middle of the array. Look through each element of the array starting from the 1st element. If the current value is less than the pivot, swap the ith element of the array with the current index and then increment the current index. If no swaps occurred, return. Quick sort the left side of the array and the right side. If the size of the array is less than 8, shell sort it.

Shell.c: Create an array with each gap number. Iterate through this gap number array. Iterate i from the current gap number to the size of the array by 1. Iterate j from i to 1 number below the gap number by decrementing j by the gap number. If the jth element of the array is less than the j - gap_number of the array, swap them. Once done the array is sorted and free the memory allocated for the gap number array.

Heap.c: Make a heap array where each element has at most 2 children that are larger than the parent. Pop the first element of the heap which is the smallest number and transfer it into a new array. Then take the last element of the heap and make it the top of the heap. Sort the heap so that each parent is smaller than its children. Once the heap is empty, the sorted array is now sorted from least to greatest.

Bash Script: For loop each sorting algorithm with a different size each loop. Write the size and moves needed into a data file. Plot the data for each sorting algorithm

Pseudocode:

Sorting.c:

get user input, if the user wants a specific sorting method add its respective number into a set, if they want all sorting methods and all the respective numbers to a set
if the user wants help with usage, print the help and exit the function
get the size of the array from the user and if no value was given set it to 100
get the seed random number generator from the user and if no value was given set it to 13371453
get the number of elements to be printed from the user and if no value was given set it to 100. If the number of elements to be printed is higher than the size of the array, set it to the size of the array
make an array of size length and set each element to a random number bit masked to 30 bits.
check the set for which sorting methods the user wants
complete the respective sorting method and print the name of the sorting method, the size of the array, moves needed, and compares needed to complete the sort.
print the number of elements to be printed from the sorted array if specified

Bubble.c:

```
# for i from 0 to (size of array -1)
        # no values have been swapped
        # for j from (size of array -1) to i
                # if the current value of the array is less than the value before it
                        # swap the current value and the one before it
                        # a value has been swapped
        # if a value hasn't been swapped break
# return sorted array
```

Shell.c:

```
# gap function(n):
        # while n is greater than 1
                # if n is less than or equal to 2, set it equal to 1 or else set it to (5n) // 11
                # add n to an array
        # return array

# shell sort function(array, size):
        # for each gap number
                # for i from the current gap number to the size of the array
```

set j = i and a temporary variable to ith element of the array
while j is greater than or equal to the current gap number and the temporary variable is less than the (j - gap) element of the array
make the jth element of the array equal to the (j - gap) element of the array
subtract the current gap number from j
make the jth element of the array equal the temporary variable
return the sorted array

Heap.c:

left child of n is equal to (2n +1)
right child of n is equal to (2n +2)
parent of n is equal to (n-1) // 2

up heap function(array, n):
while n > 0 and the nth element of the heap is less than its parent value
swap the nth element of the array with its parent value and set n to its parent

down heap function(array, heap size):
set n to 0
while the left child is less than the heap size
if the right child doesn't exist set the smaller element to the left child
if the right child exists, set the smaller element to the left child if the left child's value is smaller than the right child's value or else set it to the right child
if the nth element of the array is less than the smaller element of the array, break
swap the values of the nth element of the array with the smaller element of the array
set n to the smaller element

build heap function(sort array, heap array, size)
for n from 0 to the size of the array
set the nth value array of the heap array to the nth value of the array that needs to be sorted
up heap the heap array

heap sort function(sort array, size)
build the heap array and create a new array that will hold the sorted values
for n = 0 to the size of the array
set the nth element of the sorted array to the first element of the heap array

set the first element of the heap array to the (size - n -1)th element of the heap array

down heap the heap array with the heap size being the (size of the array - n)

return the sorted array

Quick.c:

if the size of the array is less than 8, shell sort it
set a pivot value to the approximate value in the middle of the array
for i = 0 to the size of the array

if the ith element of the array is less than the pivot, swap the current index with the ith element and increment the current index by 1

if no swaps have occurred, return
quick sort the left side of the array with the size of the array being the current index
quick sort the right side of the array with the size being the (current size - current index)
return the sorted array

Bash Script:

for each size of an array from 10 to 20,000 or 1 million to 20 million

run the plotting file for each sorting algorithm with the current size and print 0 elements

write the number of elements and moves needed into a data file

plot the data of each algorithms moves for each array size
log scale the axesa