Cole Marquard
cdmarqua@ucsc.edu
11/8/2020

CSE13s Fall 2020
Assignment:Sorting
Design Document

This project creates 4 variable length arrays and sorts them using a variety of different sorting Algorithms. This program then prints the different number of comparisons and moves for each algorithm.

optional arguments of "-b" to enable bubble sort,
"-s" to enable shell sort,
"-q" to enable QuickShort,
"-i" to enable Binary Insertion Sort,
"-p n" to set the amount of numbers printed to n default is 100,
"-r x" to set the random seed to x default is 8222022, "-n c" sets the number of elements to be sorted to c, default is 100
, -A will run all of the sorts previously mentioned

Included below is the overarching Pseudocode for my program
Any Differences between my sorting algorithms and the provided pseudo code for them
Is explained within the code itself such as the gap for shell sort.

```
Main(argc,argv){
//set binary variables to know which sorts are being used,the size of the arrays, and the number
//of elements they want printed. Would be done in C using a switch statement.
QuickSort,BubbleSort,ShellSort,BinarySort,arr_size,PrintNum=GetCommandLineArguments(ArgV)

Quick_arr = calloc(arr_size,sizeof(long)) //dynamically allocating the arrays
Bubble_arr = calloc(arr_size,sizeof(long))
Shell_arr = calloc(arr_size,sizeof(long))
Binary_arr = calloc(arr_size,sizeof(long))
For x in range(arr_size){
        Int random = rand() // (RAND_MAX/(1073741823+1)+1) //sets the bounds of the random to
                                //(0,1073741823) from the original RAND_MAX defined within the random lib
        Quick_arr[x] = random
        Bubble_arr[x] = random  //sets all the arrs with the same random numbers
        Shell_arr[x] = random
        Binary_arr[x] = random
}
//This pseudocode below  is the same for every sort with only the names being changed to the
//respective sort. This could have been a function but it only would have saved around 15 lines of
//code which seemed useless.
If(BubbleSort){
        printHeader() //print the header name
        Bubble_sorter = bubble_create() //initialize the sorter
        bubble_sort(Bubble_arr,Bubble_sorter,arr_size) // run the sorting function which gets passed
                                                        // the sorter to count comparisons and moves
        Counter = 0; //used for newlines
        For x in bubble_arr{
                print(x)
                If(counter % 7 ==0){
                        print("\n") //prints a newline
                }
        }
        print("\n") //prints a newline after having printed the specified amount

        free(bubble_sorter) //free up the allocated memory of the sorter object
}
free(Quick_arr); //free all the arrays at the end to avoid memory leaks
free(Shell_arr);
free(Bubble_arr);
free(Binary_array);}
```

PreLab part1

1. How many rounds of swapping do you think you will need to sort the numbers 8, 22, 7, 9, 31, 5, 13 in ascending order using Bubble Sort?

Since the list is 7 elements long I would guess 49 swaps would have to happen since its n^2 Complexity but if it's just “rounds” as in loops through the array it would be 6 loops to get the last 6 Numbers into place the first digit would then automatically be in place if the other 6 are aswell

2. How many comparisons can we expect to see in the worse case scenario for Bubble Sort? Hint: make a list of numbers and attempt to sort them using Bubble Sort.

In the worst case scenario we would see n(n+1)/2 comparisons as the amount of comparisons can be written as n + (n-1) + (n-2).... +1

PreLab part2

1. The worst time complexity for Shell sort depends on the size of the gap. Investigate why this is the case. How can you improve the time complexity of this sort by changing the gap size? Cite any sources you used.

As shown in the Gap sequences part of the wiki page, the gaps used can change the worst case Time complexity by a large amount. By changing the gaps used, it changes the amount of loops It will have to do with the final gap of 1. This is because having gaps allows elements to move Into more sorted positions quicker as compared to moving them 1 array space at a time. However The more gaps used the higher the overhead for running the program. By changing the gap to Quickly move numbers closer to their final position will result in the best time although this Greatly depends on the numbers and amount being sorted.
https://en.wikipedia.org/wiki/Shellsort

2. How would you improve the runtime of this sort without changing the gapp size?

You could use nearly sorted data or make the amount of data your sorting smaller, both of which will reduce the runtime without changing the gapp size.

Pre-lab Part 3

1. Quicksort, with a worse case time complexity of $O(n^2)$, doesn't seem to live up to its name. Investigate and explain why Quicksort isn't doomed by its worst case scenario. Make sure to cite any sources you use.

The only case where n^2 time complexity is achieved is when the pivot chosen happens to be the Smallest or largest element in the list. If this happens everytime a pivot is chosen then it will Reach a time complexity of n^2. This is most commonly the case when all the elements in the Array are equal to one another and is very rare in most cases. Since in most cases this occurrence Is very rare the worst case scenario does not doom quicksort. The average for the sort is n log n. https://en.wikipedia.org/wiki/Quicksort

Pre-Lab Part 4

1. Can you figure out what effect the binary search algorithm has on the complexity when it is combined with the insertion sort algorithm?

Since it still goes through each array element there is at least an n to which the binary search cuts Down on the amount of comparisons exponentially as it constantly divides the amount its checking By 2 which would be log n things to compare to. This would make the algorithm's complexity N log (n) as n elements are compared with log(n) elements in the array.

Pre-lab Part 5

1. Explain how you plan on keeping track of the number of moves and comparisons since each sort will reside within its own file.

I plan on saving them as variables within the struct such as int moves and int comparisons which Will be added to as the algorithm is used. This can then be accessed and printed to see how many Times there were swaps and comparisons.

Design Process

Over the course of this lab I learned how to work around many of the different "quirks" of The c programming language. One of the biggest ones was not having generators for the shell Sort algorithm which I got around by creating an array of gaps which could be iterated through.

Overall the lab was fairly easy to implement and create given the sorting pseudocode Provided.

I really enjoyed implementing some of the sorts such as shell sort and binary insertion sort, However the other sorts, quick and bubble, feel a little too easy and seem mostly like busy work. I would have liked if we got to implement a radix sort or something not as common.