CS 2302

Fall 2019

Lab Report #2

John Rodriguez

Due: September 20, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Introduction

For lab 2 we were given the task to implement bubble sort, quick sort, and a modified quick sort functions in our program that use recursion and return the kth smallest element in that list after it’s sorted. The modified quick sort will only do one recursive call instead of the normal two and the call it does is determined if k is smaller or more than or equal to the pivot. The second part of the lab was to implement quick sort using a stack and replicate the modified quick sort function using a while loop instead of recursion or a stack. This lab’s purpose is to become more familiar with sorting algorithms and the run time analysis of them.

Proposed Solution Design and Implementation

I approached this lab by first focusing on part one of the lab. I then created three functions being the bubble sort function, the normal quick sort function, and then modified quick sort function. The first thing I focused on was implementing bubble sort into my program and then adding the ability to return the kth smallest element in the list. I named this function select\_bubble which takes in a list a k which is used to find the kth smallest element in the sorted list.

The next function of the program that I focused on was the normal quick sort code that takes in a list, the high index of the list, the low index of the list, and finally k which is again used to find the kth smallest element in the sorted list. To implement the quick sort functions, I needed to create a partition function which makes a pivot in the list and places all elements in the list that are smaller than the pivot to the left of the pivot and all elements that are larger to the right of the pivot. The partition function takes in the list, the low index of the list, and finally the high index of the list. After implementing the partition function into my program, I created the function select\_quick which is the normal quick sort implementation that uses recursion and at the end of the function I returned the kth smallest element in the list after it has been sorted.

I then implemented the function called select\_modified\_quick which is the quick sort algorithm, but it only sorts the part of the list that k is in relative to the pivot. To do this I created a conditional that would only choose one of the recursive calls in the depending if k is smaller than the pivot and if not, it would do the other recursive call if k was larger or equal to the pivot. This function also used the partition function. This condition should improve the runtime of this function when compared to the select\_quick function.

After implementing part one of the lab into my program and testing if the functions behaved as intended, I moved onto part two of the lab. The first thing I focused on was creating a quick sort function that used a stack instead of recursion. I named this function stack\_select\_quick which takes in a list and k as it’s parameters and still uses the partition function in it. I created the variables high which holds the index of the high in the list, low which holds the low index of the list, size which holds the size of the stack, with the size variable a created the stack and named it stack and finally created the variable called top which holds the index for the top of the stack. I then pushed the initial values of low and high into the stack. In a while loop using a pivot, I then pushed elements smaller than the pivot to the left of the stack and elements larger than the pivot to the right of the stack until the list was ordered. At the end of the function I then returned the kth smallest element in the list. I was unable to implement the second task of part two into my program. I was unable to figure out how to implement quick sort without using recursion or a stack.

Experimental Results

**Test Case #1**

The first test case I did was to see if the functions were working correctly. I created four list that were unordered and identical and had 28 elements inside to see if the different functions could return the kth smallest element in the list. I first tested the program with k given the value of 3 which would return the 4th smallest element in the list. All the functions were able to return the kth smallest element which was 20, and all took 0.0 seconds to do this. All the lists were ordered afterwards except for the select\_modified\_quick function which only ordered elements 15-45 which was enough to find the kth smallest element.

A screenshot of a social media post

Description automatically generatedA screenshot of a cell phone

Description automatically generated

**Test Case #2**

The second test case I did was to use the same list as the first test, but this time give k a value that would be one of the largest elements in the list. I gave k the value of 25 which would return the 26th smallest element in the list. All the functions were able to return the 26th smallest element which was 130 and again all took 0.0 seconds. This time only the three largest elements were sorted for the select\_modified\_quick function and the rest of the functions sorted the list completely.

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**Test Case #3**

The third test case I created a list of random numbers of size 100 then made four copies of that list to test on the functions I did this test after I submitted my source code so this is the code that I added to do this test.

A screenshot of a cell phone

Description automatically generated

When I did this test, I gave k the value of 25 to return the 26th smallest element in the list and all functions returned 20 as the 26th smallest element and all but bubble sort took 0.0 seconds to run. Bubble sort took 0.015616416931152344 seconds to run.

A screenshot of a cell phone

Description automatically generated

**Test Case #4**

My fourth test case was to make a random list with 1000 elements, and I gave k the value of 87 to return the 88th smallest element in the list. I changed my driver code to this to run this test.

A close up of text on a white background

Description automatically generated

When I did this test case bubble sort wouldn’t work so I removed it from the test and tested the other 3 functions. All three functions returned 76 as the 88th smallest function but this time there was a time differences between the quick sort variations. The select\_quick took 0.0 seconds, the select\_modified\_quick took 0.046861886978149414 seconds, and the stack\_select\_quick took 0.04686331748962402 seconds. Again, bubble sort was unable to run with a list this large so a gave it a fail instead of a time, and I discovered that it can’t run with a list bigger than 100.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Time Test 1 | Time Test 2 | Time Test 3 | Time Test 4 |
| bubble | 0.0 | 0.0 | 0.015616416931152344 | FAIL |
| quick | 0.0 | 0.0 | 0.0 | 0.0 |
| modified quick | 0.0 | 0.0 | 0.0 | 0.046861886978149414 |
| stack quick | 0.0 | 0.0 | 0.0 | 0.04686331748962402 |

|  |  |
| --- | --- |
|  | big-O running time |
| Bubble | O(n^2) |
| Quick | O(n^2) |
| Modified Quick | O(n^2) |
| Stack Quick | O(n^2) |

Conclusion

This lab helped me become more familiar with quick sort and bubble sort and how quick sort is generally thought of as the fastest sorting algorithm. I also learned that the normal recursive version of quick sort ran the best out of the different variations that I created. The final thing I learned from this lab was how quick sort can be implemented using a created stack instead of using the most common method of recursion.

Appendix

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class