Lab Assignment 2

Ismael Villalobs

Professor Fuentes

TA: Dita nath

9-24-19

Introduction

For this lab assignment we had the problem of finding the Kth element in a list that was previously sorted using QuickSort, BubbleSort, and a modified version of Quicksort. In addition, we were asked to implement quicksort using stacks, and also only using a while loop. For example, if k == 0, select(L,k) return the smallest in list L, if k == 1 it returns the second smallest and so on.

Proposed Solutions

Part 1 – For bubble sort function it passes 2 parameters, the list and the kth smallest value you want to find. Bubble sort was quite simple as you iterate through the list and swap elements if the n+1 is greater than the n value in the list and returning the kth smallest value from the previously sorted list.

Next, I implemented a quicksort algorithm. For this I took in 3 parameters, the unsorted list the low value which was 0 and the high value which was the last index of the list or length of list minus one. The values that were chosen provided us the necessary indexes for partitioning the list. Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made, and another array holds values greater than the pivot value. Quick sort partitions an array and then calls itself recursively twice to sort the two resulting subarrays.

Finally, we implement a modified version of the previous quicksort, but only making one recursive call. The first recursive function from the original quicksort algorithm I implemented. After the recursive call, the low value will be set to the partitioning index plus one. As with the bubble sort and regular quicksort functions, the value at position k is returned.

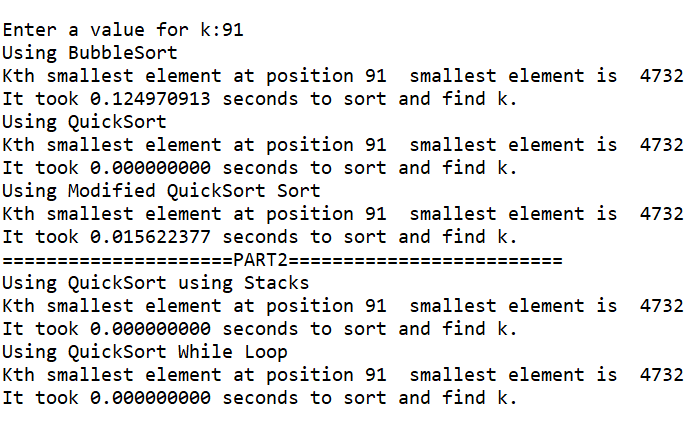
Part 2 –

For this part we were required to do the following: a quicksort implementation with a stack and quicksort only using a while loop. Dr. Fuentes showed us the example of implementing Towers of Hanoi using a stack. When going with a stack implementation for quicksort, the recursive calls must be added to the stack backwards. Using the Towers of Hanoi stack implementation example as a guide I created a class for the Stack record. In my function, I create a stack, again using my implementation for stack record. Next, while the length of the stack is greater than zero, I first get the last element from the stack and then partition to get the pivot value and then I append my left and right sub lists. The value at the k position is returned.

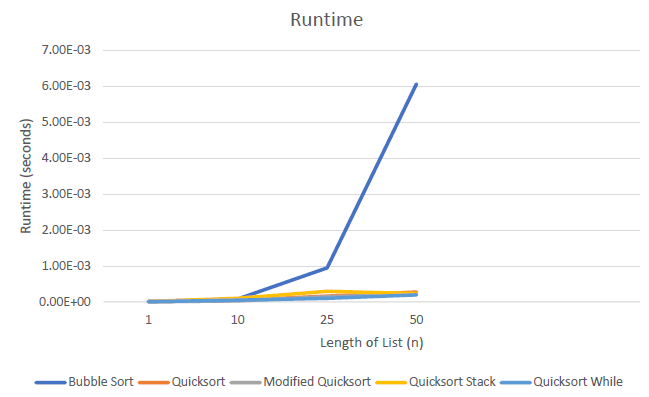
Next, I implemented quicksort with only using a while loop. For this function we had to call out previous partition function in order to get the pivot value. Next, while the pivot is not the user inputted value for k, I iterate through a while loop. Next I check for two conditions: if k is greater than the pivot, I partition through values from pivot + 1 to the highest, otherwise, I partition through values from the low to one minus the pivot value. Once my loop stops iterating through, I return the value at the pivot, since to terminate the loop the pivot must be the same value as k.

Experimental Results

Below is a sample run from my code where I find the 91st smallest value in a list of 100 elements with random values between 0-5000.



Below is a graph with updated time counting to allow for smaller numbers. I used sample sizes from 1 to 50 and analyzed the runtimes.



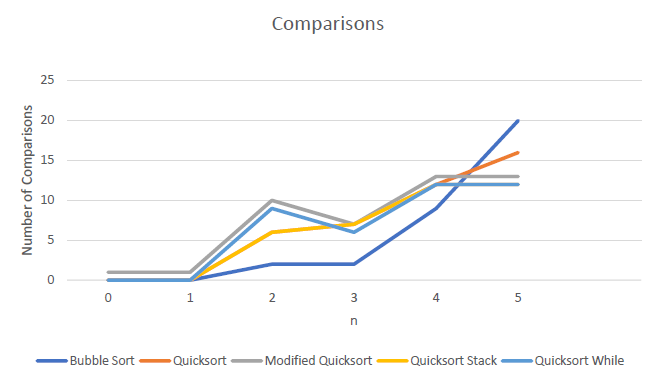
Big O Notation & Runtimes

The length of my lists for testing stop at 100 - after 100 I get a maximum recursion depth

error.

For my experimentations, on average, bubble sort was the slowest out of all of the algorithms especially when the size of the list grew. The worst-case running time for bubble sort is O(n2). Bubble sort is a sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted. The worst-case runtime for a quick sort algorithm is O(n2) based on previous work and experiences. The best or average case runtime for quicksort is O(n \* logn). Modified quicksort appears to have a runtime of O(n).

|  |  |
| --- | --- |
| n | Bubble Sort |
| 2 | 4 |
| 4 | 16 |
| 8 | 64 |
| 16 | 256 |
| 64 | 4096 |



Conclusion

Academic Honesty

“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.”

Name: Ismael Villalobos

Appendix:

'''

Ismael Villalobos

9-22-19

Lab Assignment 2

Professor Fuentes

TA-Dita Nath

Purpose: Implementing sorting algorithms like bubble sort and

quick sort to understand how they function, implement quick sort

using a stack and implement modified with no stack or recursion.

Understand the differences in their run times.

'''

import time

import random

def select\_bubble(L,k):

for i, num in enumerate(L):

try:

if L [i+1] < num:

L[i] = L[i+1]

L[i+1] = num

select\_bubble(L, k)

except IndexError:

pass

return L[k]

def partition(L,low,high):

i = ( low-1 ) # index of smaller element

pivot = L[high] # pivot

for j in range(low , high):

# If current element is smaller than or

# equal to pivot

if L[j] <= pivot:

# increment index of smaller element

i = i+1

L[i],L[j] = L[j],L[i]

L[i+1],L[high] = L[high],L[i+1]

return ( i+1 )

def quickSort(L,low,high):

if low < high:

p = partition(L,low,high)

quickSort(L, low, p-1)

quickSort(L, p+1, high)

return L

def select\_quick(L,k):

if (k < 0) or (k >= len(L)) or (len(L) <= 0):

return False

q = quickSort(L,0,len(L)-1)

return q[k]

def quickSort\_modified(L,low,high):

while low < high:

p = partition(L,low,high)

quickSort\_modified(L, low, p-1)

low = p+1

return L

def select\_modified\_quick(L,k):

if (k < 0) or (k >= len(L)) or (len(L) <= 0):

return -1

quickSort\_modified(L, 0,len(L)-1)

return L[k]

def quickSort\_stack(L,low,high,k):

stack = [[L,low,high]]

while len(stack) >0:

block = stack.pop(-1)

if block[1]>block[2]:

p = partition(block[0],block[1],block[2])

stack.append(L,low, p-1)

stack.append(L,p+1,high)

return L[k]

def select\_quick\_while(L, low, high, k):

pivot = partition(L, low, high)

while pivot != k:

if k > pivot:

pivot = partition(L, pivot + 1, high)

elif k < pivot:

pivot = partition(L, low, pivot - 1)

return L[pivot]

list\_length = 100

L1 = [random.randint(0,5000) for i in range(list\_length)]

#print(L1.sort())

print('=====================PART1=========================')

k = 0

while (True):

try:

k = int(input('Enter a value for k:'))

break

except:

print('Please enter a positive integer number.\n')

if (k >= len(L1)) or k < 0:

print("\nError: Value for k out of bounds. List size:", len(L1), "\n")

else:

#bubble

start = time.time()

pos = select\_bubble(L1,k)

stop = time.time()

runTime= stop-start

print('Using BubbleSort')

print('Kth smallest element at position',k,' smallest element is ',pos)

print("It took","{:.9f}".format(runTime), "seconds to sort and find k.")

#quick

start = time.time()

pos = select\_quick(L1,k)

stop = time.time()

runTime= stop-start

print('Using QuickSort')

print('Kth smallest element at position',k,' smallest element is ',pos)

print("It took","{:.9f}".format(runTime), "seconds to sort and find k.")

#modded quick

start = time.time()

pos = select\_modified\_quick(L1,k)

stop = time.time()

runTime= stop-start

print('Using Modified QuickSort Sort')

print('Kth smallest element at position',k,' smallest element is ',pos)

print("It took","{:.9f}".format(runTime), "seconds to sort and find k.")

print('=====================PART2=========================')

# quicksort with stack implem.

start = time.time()

pos = quickSort\_stack(L1,0,len(L1)-1,k)

stop = time.time()

runTime= stop-start

print('Using QuickSort using Stacks')

print('Kth smallest element at position',k,' smallest element is ',pos)

print("It took","{:.9f}".format(runTime), "seconds to sort and find k.")

# quicksort with a while loop

start = time.time()

pos = select\_quick\_while(L1,0,len(L1)-1,k)

stop = time.time()

runTime= stop-start

print('Using QuickSort While Loop')

print('Kth smallest element at position',k,' smallest element is ',pos)

print("It took","{:.9f}".format(runTime), "seconds to sort and find k.")