CS 2302 Data Structures

Fall 2019

Lab Report #2

Due: 09/19/2019

Professor: Olac Fuentes

TA: Anindita Nath

Programmer: Miriam Olague

Introduction

` For this lab, we were asked to work with Quick Sort and find the kth element the user enters within the sorted list. For this lab, it is key to remember how Quick Sort works and different ways to sort the lists. The main objective of this lab is to get a better understanding about Quick Sort and find different ways to solve a problem.

Proposed Solution Design and Implementation

Part #1 select_bubble:

For this operation, I used a code I created on Java. For this part, I iterated through the entire list, and continuously compared an element to the next one. If the next element was greater than the next one, I swapped the elements. I then printed out the sorted list. After printing the sorted list, I returned the kth position the user entered. On average and worst case scenario, Quicksort takes O(n^2). The number of comparisons that it makes is $n(n - 1)/2$.

Part #1 select_quick:

For this operation, I iterated through the entire list that was between low and high inside of a method called 'partition'. I compared elements to the pivot to see if they were smaller than the pivot. If they were, I swapped the elements. Inside of 'select_quick', I first compared if low was smaller than high, if they were, I called partition and made recursive calls inside of the method, so partition was called continuously until the list was sorted. I then returned the kth position the user entered. The worst case for this is O(n^2), however, on average, it takes O(n log n) comparisons to sort n. The number of comparisons that it makes on average is O(n^2) and worst case scenario is O(n log n)

Part #1 select_modified_quick:

For this operation, I created a method called 'select_modified_firstsplit'. Inside of this method, I created empty lists in which I split the given list into two sub lists, one containing smaller elements than the pivot, one that has elements equal to the pivot, and one that has elements greater than the pivot. I made the first element of the list the pivot and iterated through the entire list. I compared all elements to the pivot and appended them to 'left', 'equal', or 'right'. I added 'equal' to 'right'. I compared the length of the list 'left' with 'k' so that way I could determine if k was inside of 'left' or inside of 'right'. If k was greater than the length of left, I returned 'right', if not, I returned 'left'.

Inside of the method called 'select_modified_quick', I compared low and high, if low was smaller than high, I called the method 'partition' (as mentioned before in **Part #1 select_quick**). This method is almost the same as Part #1 select_quick, the difference is that it was now given the list that contains 'k'. I subtracted k minus the length of the list 'left' so I could determine the true position of 'k'. After that, I then returned the element at position k.

The worst case for this is O(n^2), however, on average, it takes O(n log n) comparisons to sort n. The number of comparisons that it makes on average is O(n^2) and worst case scenario is O(n log n)

Part #2 quick_stack:

For this operation, I created a stack that was the size of the list, in which I saved the elements top, low, and high. I then iterated through the list. I kept saving high, low, and called partition. I created an 'if' statement, in which I checked if the pivot was greater than low, and if it was, I saved low, and decreased the pivot. In another 'if' statement, I did the same but incremented the pivot. I printed the sorted list and returned the kth element. The worst case for this is O(n log n). The worst case scenario for the number of comparisons that it makes is O(n log n).

Part #2 while_loop:

For this operation, I was unable to sort the array.

Experimental Results

Part #1 select_bubble:

For this operation, I did not have a lot of trouble with since I understood how to do it. I had Quick Sort but in Java. I constantly printed the list to check if it was being sorted inside the for loop.

```
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 60

part1 (1) or part2 (2)?: 1
------------------------------------------------------
----------Hello, welcome to Part 1 of Lab 2!----------
------------------------------------------------------

What bullet of Part 1 would you like to access? bubble (b), quick
(q), or modified (m): b
This is select_bubble:
The sorted list is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6,
6, 6, 6, 6, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 15, 15, 15,
15, 15, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 19, 19, 19, 19,
19, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 33, 33,
40, 40, 40, 40, 60, 60, 60, 60, 60, 60, 123, 123, 123, 123, 123,
123, 3452, 34234, 434235]
Element at position  60 is:  3
Time it took to sort and return element:  0:00:00.006639
----------
```

Case 1: Input = 60

```
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 70

part1 (1) or part2 (2)?: 1
----------------------------------------------------
----------Hello, welcome to Part 1 of Lab 2!----------
----------------------------------------------------

What bullet of Part 1 would you like to access? bubble (b), quick
(q), or modified (m): b
This is select_bubble:
The sorted list is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6,
6, 6, 6, 6, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 15, 15, 15,
15, 15, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 19, 19, 19, 19,
19, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 33, 33,
40, 40, 40, 40, 60, 60, 60, 60, 60, 60, 123, 123, 123, 123, 123,
123, 3452, 34234, 434235]
Element at position  70 is:  3
Time it took to sort and return element:  0:00:00.005022
----------
```

Case 2: Input = 70

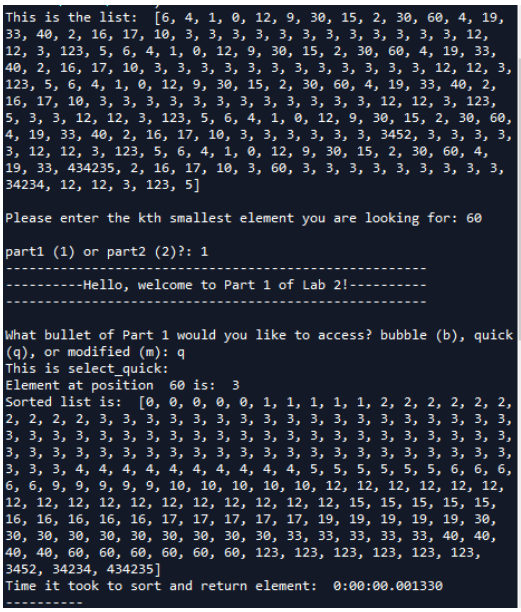
```
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 80

part1 (1) or part2 (2)?: 1
----------------------------------------------------
----------Hello, welcome to Part 1 of Lab 2!----------
----------------------------------------------------

What bullet of Part 1 would you like to access? bubble (b), quick
(q), or modified (m): b
This is select_bubble:
The sorted list is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6,
6, 6, 6, 6, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 15, 15, 15,
15, 15, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 19, 19, 19, 19,
19, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 33, 33,
40, 40, 40, 40, 60, 60, 60, 60, 60, 60, 123, 123, 123, 123, 123,
123, 3452, 34234, 434235]
Element at position  80 is:  3
Time it took to sort and return element:  0:00:00.003247
----------
```

Case 3: Input = 80

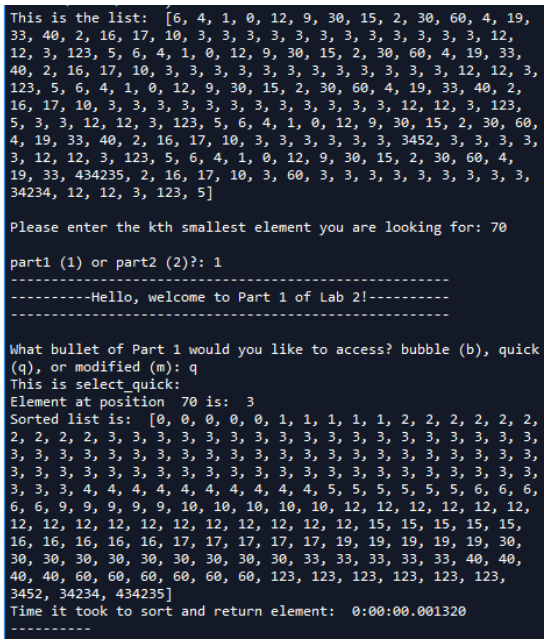
Part #1 select_quick:

For this operation, I attempted to do Quick Sort inside of only one function. I could not find 'k' and return since it was recursive. I also tried splitting the list into 'left', 'right', and 'equal', this did not work since it was also recursive and when I attempted to find 'k', it would

go out of index. If I tried comparing ‘k’ with the length of list ‘left’, it would not recognize it as an int. I then just created a partition that was called inside of ‘select_quick’ and made recursive calls within the method so it could sort the list.



Case 1: input=60



Case 2: Input=70

```
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 80

part1 (1) or part2 (2)?: 1
-----------------------------------------------------
----------Hello, welcome to Part 1 of Lab 2!----------
-----------------------------------------------------

What bullet of Part 1 would you like to access? bubble (b), quick
(q), or modified (m): q
This is select_quick:
Element at position  80 is:  3
Sorted list is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 6,
6, 6, 9, 9, 9, 9, 10, 10, 10, 10, 10, 12, 12, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 15, 15, 15, 15, 15,
16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 19, 19, 19, 19, 19, 30,
30, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 33, 33, 40, 40,
40, 40, 60, 60, 60, 60, 60, 60, 123, 123, 123, 123, 123, 123,
3452, 34234, 434235]
Time it took to sort and return element:  0:00:00.001356
----------
```

Case 3: Input = 80

Part #1 select_modified_quick:

I approached this the same way I approached **Part #1 select_quick**. It was a lot of trial and error since I did not know how to split the list only once. I tried making a recursive call once, but I had no idea how to stop it. I did not know how to send the list that contained the ‘k’th element. I then thought it would be easier to split it inside of a method and send the sub list that contained the ‘k’th element.

```
In [16]: runfile('C:/Users/miria/Documents/---Fuentes CS3----/Lab
2/Code/Lab_2_v2.0.py', wdir='C:/Users/miria/Documents/---Fuentes
CS3----/Lab 2/Code')
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 60

part1 (1) or part2 (2)?: 1
-----------------------------------------------------
----------Hello, welcome to Part 1 of Lab 2!----------
-----------------------------------------------------

What bullet of Part 1 would you like to access? bubble (b), quick
(q), or modified (m): m
This is select_quick_modified:
Element at position  60 is:  3
Sorted list is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5]
Time it took to sort and return element:  0:00:00.002992
----------

In [17]:
```

Case 1: Input=60

```
In [18]: runfile('C:/Users/miria/Documents/---Fuentes CS3----/Lab
2/Code/Lab_2_v2.0.py', wdir='C:/Users/miria/Documents/---Fuentes
CS3----/Lab 2/Code')
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 70

part1 (1) or part2 (2)?: 1
------------------------------------------------------
----------Hello, welcome to Part 1 of Lab 2!----------
------------------------------------------------------

What bullet of Part 1 would you like to access? bubble (b), quick
(q), or modified (m): m
This is select_quick_modified:
Element at position  70 is:  3
Sorted list is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5]
Time it took to sort and return element:  0:00:00.002379
----------
```

Case 2: Input=70

```
In [32]: runfile('C:/Users/miria/Documents/---Fuentes CS3----/Lab
2/Code/Lab_2_v2.0.py', wdir='C:/Users/miria/Documents/---Fuentes
CS3----/Lab 2/Code')
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 80

part1 (1) or part2 (2)?: 1
------------------------------------------------------
----------Hello, welcome to Part 1 of Lab 2!----------
------------------------------------------------------

What bullet of Part 1 would you like to access? bubble (b), quick
(q), or modified (m): m
This is select_quick_modified:
Element at position  80 is:  3
Sorted list is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5]
Time it took to sort and return element:  0:00:00.001105
----------
```

Case 3: Input = 80

Part #2 quick_stack:

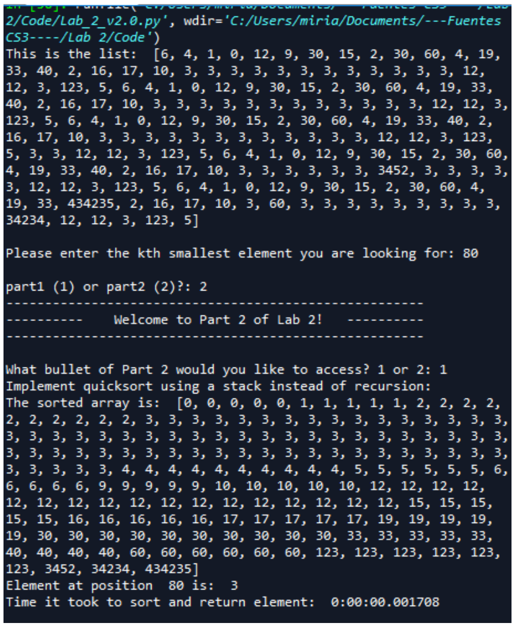
For this operation, I approached this problem the same way I attempted to approach select_quick. I created three temporary lists in which I determined if the lowest was greater than or smaller than the pivot. I was only able to split the list into two sub lists once and did not know how to proceed from there. I then thought about creating a list in which I could save the greatest element, smallest element, and the pivot. I pre-created the size of the stack so it would not go out of index.

```
2/Code/Lab_2_v2.0.py', wdir='C:/Users/miria/Documents/---Fuentes
CS3----/Lab 2/Code')
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 60

part1 (1) or part2 (2)?: 2
-----------------------------------------------------
----------    Welcome to Part 2 of Lab 2!   ----------
-----------------------------------------------------

What bullet of Part 2 would you like to access? 1 or 2: 1
Implement quicksort using a stack instead of recursion:
The sorted array is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6,
6, 6, 6, 6, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 15, 15, 15,
15, 15, 16, 16, 16, 16, 16, 17, 17, 17, 17, 19, 19, 19, 19,
19, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 33, 33,
40, 40, 40, 40, 60, 60, 60, 60, 60, 60, 123, 123, 123, 123, 123,
123, 3452, 34234, 434235]
Element at position  60 is:  3
Time it took to sort and return element:  0:00:00.001449
```

Case 1: Input=60

```
2/Code/Lab_2_v2.0.py', wdir='C:/Users/miria/Documents/---Fuentes
CS3----/Lab 2/Code')
This is the list:  [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19,
33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12,
12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33,
40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3,
123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 12, 12, 3, 123,
5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60,
4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3,
3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4,
19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3, 3, 3, 3, 3, 3,
34234, 12, 12, 3, 123, 5]

Please enter the kth smallest element you are looking for: 70

part1 (1) or part2 (2)?: 2
-----------------------------------------------------
----------    Welcome to Part 2 of Lab 2!   ----------
-----------------------------------------------------

What bullet of Part 2 would you like to access? 1 or 2: 1
Implement quicksort using a stack instead of recursion:
The sorted array is:  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6,
6, 6, 6, 6, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 15, 15, 15,
15, 15, 16, 16, 16, 16, 16, 17, 17, 17, 17, 19, 19, 19, 19,
19, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 33, 33,
40, 40, 40, 40, 60, 60, 60, 60, 60, 60, 123, 123, 123, 123, 123,
123, 3452, 34234, 434235]
Element at position  70 is:  3
Time it took to sort and return element:  0:00:00.000769
```

Case 2: Input=70



Case 3: Input = 80

Part #2 while_loop:

For this operation, as redundant as I sound, I approached this problem the same way I approached select_quick and quick_stack. I was only able to divide the list into two sub lists. I also tried to pre-create a list in which I would append the values every time it went inside of the while loop. I tried approaching the problem the same way I approached ‘quick_stack’ but was unable to replicate everything since I could not make recursive calls (I wanted to call ‘partition’) but that was a recursive function.

Case 1: input = 60

```
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186]
Element at position  70 is: 0
Time it took to sort and return element:  0:00:00.626530
```

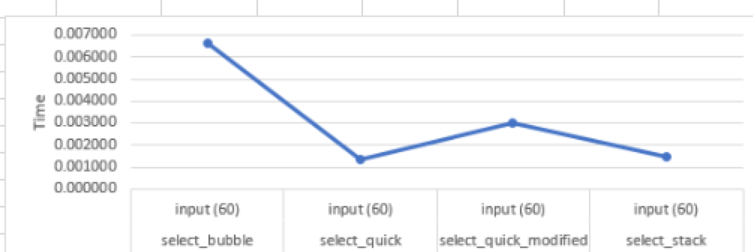
Case 2: input = 70

```
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186, 186,
186, 186, 186, 186]
Element at position  80 is: 0
Time it took to sort and return element:  0:00:00.800681
```

Case 3: Input = 30

Input = 60

Method	Input	Time
select_bubble	input (60)	0.006639
select_quick	input (60)	0.001330
select_quick_modified	input (60)	0.002992
select_stack	input (60)	0.001449



Input = 70

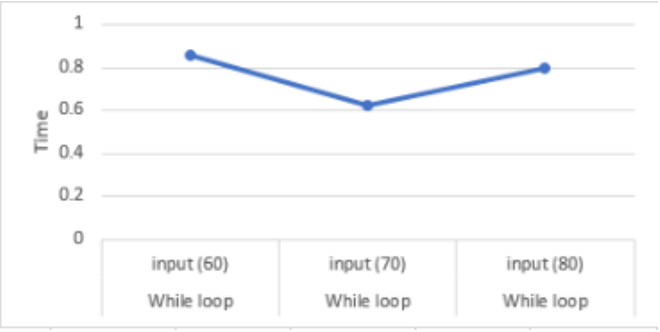
Method	Input	Time
select_bubble	input (70)	0.005022
select_quick	input (70)	0.001320
select_quick_modified	input (70)	0.002379
select_stack	input (70)	0.001449

Input = 80

Method	Input	Time
select_bubble	input (80)	0.003247
select_quick	input (80)	0.001356
select_quick_modified	input (80)	0.001105
select_stack	input (80)	0.001708

While Loop Only

Method	Input	Time
While loop	input (60)	0.858566
While loop	input (70)	0.62653
While loop	input (80)	0.800681



Input = 60; 70; 80

As the results show, there is a visible change between running times between different solutions solving for quick sort. Bubble sort takes more time to sort compared to quick sort. Quick sort is fast but with the modified quick sort it makes even faster since it divided the work into two. Stacks aren't as efficient as the quick sort that has a partition and the modified version of quick sort. For the while loop, I was not able to solve it, therefore I did not include it in the graph since it made all of the other results appear as zeros; the while loop was producing big running times so I decided to not include it. However, I attached a different graph in which the running time for the while loop is displayed.

Conclusion

In conclusion, this lab made me think out of the box. It made me think of many ways to solve Quick Sort, not only one solution. My knowledge on stacks improved and was able to organize the data in an orderly manner so it could be cleaner. I also learned how to iterate

through lists using stacks. The analytical running times do agree with what I see in practice since quick sort is fast to sort compared to bubble sort.

Appendix

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 21 09:55:24 2019

@author: miria
"""

#Course: CS 2302 Data Structures
#Programmer: Miriam Olague
#Lab 2 Bubble sort, Quick sort, and Modified Quick Sort
#Date of last modification: September 22th 2019
#Professor: Olac Fuentes
#Purpose: The purpose of this lab is to find the an element in position k,
#   learn about how to find an element with recursion.
#TA: Anindita Nath


import sys
from datetime import datetime
#-------------------------------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------------------------------
#-----------------------------------------------------------PART 1--------------------------------------------------
#-------------------------------------------------------------------------------------------------------------------
#-----------------------------------------------------------SELECT BUBBLE-------------------------------------------

def select_bubble(L, k):
    print("This is select_bubble: ")
    size = len(L)

    for i in range(size):

        for j in range(0, size-i-1): #This iterates through the list
            if L[j] > L[j+1]: #this is checking if the element is greater than the next one
                temp = L[j] #saving value into temporary variable
                L[j] = L[j+1] #swapping
                L[j+1] = temp #swapping

    print("The sorted list is: ", L)

    returning = L[k] #returning element at position k
    return returning
    print("----------")
#-----------------------------------------------------------QUICKSORT-----------------------------------------------

def partition(L,l,h):
    i = ( l-1 ) #I am saving the index of smaller element
    pi = L[h] #This is the pivot

    for j in range(l, h):
        if   L[j] <= pi: #comparing element with pivot

            i = i+1 #increment index of small by one
            temp = L[i] #saving it into a temporary variable
            L[i] = L[j] #swapping
            L[j] = temp #swapping

    temp2 = L[i+1] #I am saving it into a temporary variable
    L[i+1] = L[h] #swapping the positions of last element with new one
    L[h] = temp2 #saving temp2 into high
```

```
    L[h] = temp2 #saving temp2 into high
    return(i+1)  #increment


def select_quick(L,low,high, k):
    if low < high:
        pi = partition(L,low,high)  #saving the call here

        select_quick(L, low, pi-1, k) #calling itself to keep on sorting
        select_quick(L, pi+1, high, k) #callign itself to keep on sorting

    return L[k] #returning the element at position k

#-----------------------------------------------------------MODIFIED QUICK-----------------------------------------

def select_modified_firstsplit(L, k):
    #this is the function that is making the first split and returnig the list that has
    #   the kth element
    left = [] #temporary
    equal = [] #temporary
    right = [] #temporary

    if len(L) > 1:
        pi = L[0] #making pivot L[0]
        for i in L: #iterating through the whole list
            if i < pi: #appending if i is less than pivot
                left.append(i)
            elif i == pi: #appending if i is equal to the pivot
                equal.append(i)
            elif i > pi: #appending if i is greater than the pivot
                right.append(i)

    for i in range(len(equal)): #I am adding the number that was the pivot to right
        right.append(equal[i])



    if k > len(left): #This is checking if k is inside left or not
        return(right, len(left)) #if k is greater than the length of left, it returns right

    else:
        return(left, len(left)) #if k is within the length of left, it returns left


def select_modified_quick(new, low, high, two, k): #this is modified quick sort

    if low< high:
        pi = partition(new,low,high) #saving the call here

        select_modified_quick(new, low, pi-1, two, k) #calling itself to keep sorting
        select_modified_quick(new, pi+1, high,two, k) #calling itself to keep sorting
    kk = k-two #k is the number the user entered, and I am subtracting 'two' because
        #'two' represents the length of length of left, so that way it returns the right
        #value inside of right
    return new[kk]


#-------------------------------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------------------------------
#-----------------------------------------------------------PART 2-----------------------------------------
```

```
#-------------------------------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------------------------------
#-----------------------------------------------------------PART 2--------------------------------------------------
#-------------------------------------------------------------------------------------------------------------------
#-----------------------------------------------------------STACK---------------------------------------------------

def quick_stack(L, low, high):
    length = high - low + 1 #this is length of list/arr
    s = [0] * length #this is the stack I'm creating, size of the list/array

    upmost = L[-1] #top of stack


    upmost = upmost + 1 #incrementing top's value
    s[upmost] = low #saving the value of low here
    upmost = upmost + 1 #incrementing top's value
    s[upmost] = high #saving high's value here


    while upmost >= 0:  #popping

        # This is popping high and low
        high = s[upmost] #saving the top in high
        upmost = upmost - 1 #decrement index
        low = s[upmost] #saving value of top-1
        upmost = upmost - 1 #decrement index, index out of range error if taken out
        pi = partition(L, low, high)  #calling method partition and saving it as pivot
  #-------------
        if pi-1 > low: #checking left side of pivot

            upmost = upmost + 1 #incrementing index
            s[upmost] = low #saving value
            upmost = upmost + 1 #incrementing index
            s[upmost] = pi - 1 #saving pivot-1
  #-------------
        if pi + 1 < high:  #checking right side of pivot

            upmost = upmost + 1 #incrementing index
            s[upmost] = pi + 1 #saving pivot+1
            upmost = upmost + 1 #incrementing index
            s[upmost] = high #saving value
  #-------------

    print("The sorted array is: ",L) #printing sorted array
    return L[k] #returning element at position k
#-----------------------------------------------------------WHILE LOOP----------------------------------------------

def while_loop(L, low, high, k):
    length = high - low +1
    new_list = [0] * length
    temp = high

    for temp in range(len(L)):
        while temp != low:
            pi = partition(L,low,high)
            new_list.append(pi)
            temp = temp-1


    print(new_list)
```

```
    print(new_list)
    return new_list[k]

#=============================================================================================================================
#=============================================================================================================================

L = [6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
     3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3, 3, 3,
     3, 3, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2, 16, 17, 10, 3, 3, 3, 3, 3, 3,
     3, 3, 3, 3, 3, 3, 12, 12, 3, 123, 5, 3, 3, 12, 12, 3, 123, 5, 6, 4, 1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 40, 2,
     16, 17, 10, 3, 3, 3, 3, 3, 3, 3452, 3, 3, 3, 3, 3, 12, 12, 3, 123, 5, 6, 4,
     1, 0, 12, 9, 30, 15, 2, 30, 60, 4, 19, 33, 434235, 2, 16, 17, 10, 3, 60, 3, 3, 3, 3,
     3, 3, 3, 3, 3, 34234, 12, 12, 3, 123, 5]

#YOU CAN ALSO ASK THE USER TO ENTER A LIST THEMSELVES
"""L = []
size = int(input("What is the size of the list you want to create?: "))
for i in range(0, size):
    num = input("Please enter number: ")
    L.append(num)"""

print("This is the list: ", L)
k = int(input("Please enter the kth smallest element you are looking for: "))

part = input("part1 (1) or part2 (2)?: ")
n = len(L)

if part == '1':
    print("-----------------------------------------------------")
    print("----------Hello, welcome to Part 1 of Lab 2!----------")
    print("-----------------------------------------------------")
    bullet = input("What bullet of Part 1 would you like to access? bubble (b), quick (q), or modified (m): ")

        #**************************************************
    if bullet == "b":
        startTime = datetime.now() #starting time
        print("Element at position ", k, "is: ", select_bubble(L, k))
        print("Time it took to sort and return element: ",datetime.now()-startTime) #printing time
        print("----------")

        #**************************************************

    elif bullet == "q":
        startTime = datetime.now() #starting time

        print("This is select_quick: ")
        print("Element at position ", k, "is: ", select_quick(L,0,n-1, k))  #0 is the beginning of list, n-1 is the last element
        print("Sorted list is: ", L)
        print("Time it took to sort and return element: ",datetime.now()-startTime) #printing time

        print("----------")

        #**************************************************

    elif bullet == "m":

        startTime = datetime.now() #starting time

        L2 = select_modified_firstsplit(L, k) #splitting it into two sub lists
        one = L2[0] #this is the new list
```

```
        one = L2[0] #this is the new list
        two = L2[1] #this is the length of left



        print("This is select_quick_modified: ")
        print("Element at position ",k,"is: ",select_modified_quick(one, 0, len(one)-1, two, k)) #CALLING METHOD
        #0 is the beginning of list, len(one) is the last element of new list
        print("Sorted list is: ",one)
        print("Time it took to sort and return element: ",datetime.now()-startTime) #printing time
        print("----------")
        #************************************************

    else:
        sys.exit()

#=============================================================================================================================================
#=============================================================================================================================================
if part == '2':
    print("-------------------------------------------------------")
    print("----------    Welcome to Part 2 of Lab 2!    ----------")
    print("-------------------------------------------------------")

    bullet2 = input("What bullet of Part 2 would you like to access? 1 or 2: ")

    if bullet2 == '1':
        startTime = datetime.now() #starting time
        print("Implement quicksort using a stack instead of recursion: ")
        print("Element at position ",k, "is: ",quick_stack(L, 0, n-1)) #0 is the beginning of the list,
        #n-1 is the last element in the list
        print("Time it took to sort and return element: ",datetime.now()-startTime) #printing time

    elif bullet2 == '2':
        startTime = datetime.now() #starting time
        print("Implement select_modified_quick(L, k) using a while loop without stacks or recursion: ")
        print("Element at position ",k,"is:",while_loop(L, 0, n-1, k)) #0 is the beginning of the list, n-1 is the last element in the list
        print("Time it took to sort and return element: ",datetime.now()-startTime) #printing time

    else:
        sys.exit()
#=============================================================================================================================================
#=============================================================================================================================================
else:
    sys.exit
```