

Lab 2: Quick sort

Objective:

to successfully sort all the numbers in an array, using your partition function.

1. Your partition function should return the correct pivot index of the subarray.
2. You must use a dynamically-allocated array for this lab; no other data structures may be used.
3. The following summarizes the commands used in the Quicksort lab: (OK means no error was raised.)

Function	DESCRIPTION	OUTPUT
QuickSort constructor	Create a quickSort array of size <i>capacity</i> . Set initial number of elements (Size) to 0.	OK or Error
addToArray <int>	Add data (an integer value) to quickSort array. Duplicates are allowed.	OK or Error
capacity	Return the capacity of the quickSort array.	size
clear	Delete all inserted nodes from the QuickSort array. (Do not delete QuickSort array - capacity stays the same.)	OK or
size	Return the number of elements currently in the array.	An integer value
private quickSort <start> <end>	quickSort the elements in the quickSort array from index <start> to index <end> (where <end> is one past last element) using median and partition functions.	OK or Error
public quickSort	quickSort all the elements in the quickSort array using median and partition functions.	OK or Error
medianOfThree <start> <end>	1) Calculate the middle index (middle = (start + end)/2), then 2) bubble-sort the values at the start, middle, and end indices. (<right> is one past last element.)	Index of the pivot (middle index); -1 if provided with invalid input
partition <start> <end> <pivot>	Partition the quickSort array (<start>, <end> and <pivot> indexes) around the pivot value. Values smaller than or equal to the pivot should be placed to the start of the pivot while values larger than the pivot should be	Pivot's ending index, -1 if provided with invalid input

	placed to the right of the pivot. (<i><end></i> is one past last element.)	
printArray	Print the contents of the quickSort array as comma separated values (using a <code>toString()</code> function.)	Array values or Empty

Steps:

- **Step 1** - Begin with a main function.
 - a. You will need to write your own main function.
 - b. Use command line arguments for input and output files.
- **Step 2** - Add your QuickSort class.
 - a. Your QuickSort class should contain a dynamically-allocated templated array. Before you focus too much on the actual sorting algorithm, make sure the logistics of the class work correctly. Focus on `addToArray()`, `clear()`, `getSize()`, and `toString()` member functions.
 - b. The QuickSort class should ask the user to input an integer array size called *capacity* and create an array accordingly, by randomly generating *capacity* (for example 100) integers (you can use any random number generator). These created integers should be added to the array by the `addToArray()` function.
- **Step 3** - Write your *medianOfThree()* function in the QuickSort class.
 - a. The Median-of-Three function takes start and right indexes as parameters and then calculates the index in the middle of the indexes (rounding down if necessary). Note that these are being done on the array populated in Step2.b.
 - b. Sort the left, middle, and right numbers from smallest to largest in the array.
 - c. Finally, return the index of the middle value. This will be your pivot value in the `sortAll()` function.
- **Step 4** - Write your *partition()* function in the QuickSort class.
 - a. The `partition()` function should begin by swapping the leftmost element of the array with the pivot index element.
 - b. Now, follow partition the array such that all elements less than or equal to the pivot value are left of the pivot and all elements great than the pivot value are to the right of the pivot.
 - c. The `partition()` function should return the location of the pivot index.
- **Step 5** - Write the private `quickSort()` function, using your *medianOfThree()* and *partition()* functions.
 - a. Note that this function will be recursive. Most people use public `quicksort()` as a starter function that then call another private `quicksort` function to do the sorting.
 - b. To sort, first call your *medianOfThree()* function to sort the first, middle, and last elements and return the pivot index.
 - c. Now, call your *partition()* function, using the index returned from *medianOfThree()* as the pivot index. This function will return a new pivot index where your array is split.
 - d. Finally, recursively call your `sort()` function on the two halves of your array, with one half from the left to the pivot and the other half from the pivot to the right.
 - e. Print the sorted array.