# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Spring 2022** |
|  | | | | | |
| Points |  | Document author: | **MingkuanPang** | | |
|  | Author’s email: | **Yafking20 @mail.fresnostate.edu. email** | | |
| Laboratory number: | **06 - Quick Sort** | | |
|  | | | | | |

**1. Statement of Objectives**

This lab asks for implementation of quick sort in three ways: The first element be pivot, a random element in the array and the median of the first, middle, and last elements in the array to be pivot.

Finally, analyze and discuss the execution of time complexity for these three ways of implementation.

**2. Experimental Procedure**

**Partition:**

Text

Description automatically generated

The partition function sets one of the elements in the array to be pivot, then divides the array into three parts: A [low…. pivot - 1], A [pivot], A [pivot + 1…. high]. The first part will be the set of all the numbers in the array that less than the pivot, the second part will be the pivot itself. The third part will be all the numbers in the array that greater than the pivot. First, makes pivot to be A[low]:

pivot = A [low], and the set the starting position of the larger number to high + 1: i = high+1. Then start scanning the whole array from the end. Every time if there is an element that greater than the pivot, decrease k by 1 than swap the element and the position of the higher number. After going through the whole array, the array will be

A [low…i…high]. A [low…i-1] will be the sub-array that has all the elements smaller than the pivot, and A [i... high] will be the sub-array that has all the elements greater than the pivot. Eventually, swap the pivot with i-1, then the array will be A [low…. pivot], A [i…. high]. That is A [low…. pivot-1], A[pivot], A [pivot+1…high] where pivot = i-1. It return the position of the pivot as a result.

**Quick sort:**

Text

Description automatically generated

In the quick sort function, put the array into partition function that mentioned above first. Then it will be A [low … pivot - 1], A [pivot], A [pivot+1 … high]. Then do the same thing on those sub array A [low … pivot - 1], A [pivot+1 … high]. When the whole array is sorted, low will meet high. Which will be low=high. Therefore, when low is still less than high (low<high). We keep doing those operations on the array until low = high.

**Create\_vector:**

Text

Description automatically generated

It reads a list of numbers that separate by space and push them in to an array one by one. Finally, return the array as a result.

**Print\_vector:**

Text

Description automatically generated

It prints out the element of an array.

**Pick\_median**

Graphical user interface, text, application

Description automatically generated

Pick\_median will return the median number of three integers. First, calculates the maximum number of those three integers then calculate the minimum number of those three integers. Whoever is between the maximum and minimum will be the median number.

**Median\_partition**

Text

Description automatically generated

Median partition gets the median of the first, middle and last element in the array, then swap it to the element on the index “low”. Then pass the array to the regular partition function.

**Median quick sort**

Text

Description automatically generated

Almost the same as the regular version of quick sort, but median quick sort uses median partition.

**Random\_partition**

Text

Description automatically generated

Random partition makes a random number “i” between low and high, then swap the element on the i and the element on the low. Eventually, pass the whole array to regular partition function.

**Random\_quick\_sort**

Text

Description automatically generated

Random quick sort is almost the same as the regular quick sort, but it uses random\_partition instead of the regular partition.

**3. Analysis**

**Main function:**

Text

Description automatically generated

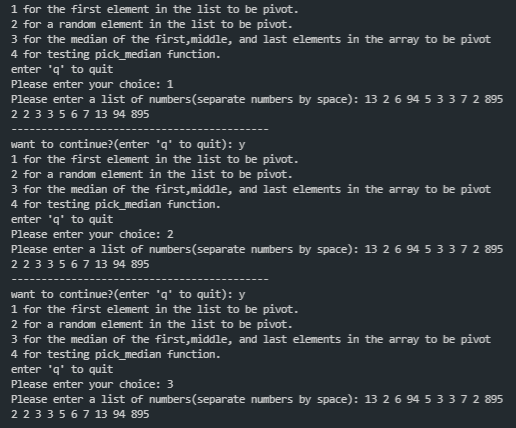
First, the program will ask you to choose which sorting algorithm you want to use, 1 for the regular quick sort, 2 for the random quick sort, 3 for the median quick sort, and 4 for testing the pick\_median function. Enter letter ‘q’ to quit the choose. Then, after making choice, it will give results of those functions returns.

**Output**

**For testing function pick\_median:**



**For testing different types of quick sorts:**



Time complexity for case 1: The first element in the array be the pivot.

**Best case:**

The best case should be dividing the array into exactly half every time after partition.

Therefore, the time complexity should be T(n) = 2\* T(n/2) + O(n) where O(n) is the time complexity for partition. Therefore, by master theorem:

**Worst case:**

For the worst case, the array is already sorted or almost sorted. Thus, the index of the pivot will not change after partition. In that case, every time after the partition, the size of array to be sort only decrease by 1. Therefore, the time complexity will be

Time complexity for case 2: A random element in the array.

**Best case:**

I think the best case will be the same as the case that first element be the pivot. If the array can be divided into exactly half every time after partition. The time complexity will be

**Worst case:**

For the worst case, it still the same as the case that first element be the pivot. If the index of the pivot will not change every time after partition. The time complexity will be

Since is randomly pick, the worst case is very unlikely to happen, unless every element in the array is the same such as ({1,1,1,1,1,1}). The average time complexity for the random quick sort should be the best case which

Time complexity for case 3: The median of the to be the pivot.

**Best case:**

The best case should be the same as the case that first element be the pivot. If the array can be divided into exactly half every time after partition. The time complexity will be

**Worst case:**

For the worst case, it will be the same as the regular quick sort. If the median is exactly the middle one in the array. Then worst case will happen. In the worst case, the index of the pivot will not be changed. Therefore,

Still, the worst case is unlikely to happen than the regular quick sort. Therefore, the average of the time complexity should be

**4. Encountered Problems**

For the implementation of pick median function. I tried pretty hard to calculate the average of those three numbers and compared them to find out which one is the closest one to the average. It took so much extra space.

Text

Description automatically generated

Then I realized I can just calculate the maximum and the minimum, whoever is between the maximum and the minimum will be the median number.

**5. Conclusions**

Even though the time complexity for the best case of quick sort is , it is still can be on the worst case. To avoid the worst case, we can change the way to pick the pivot in the array. Neither random or median quick sort will have the better performance than the regular quick sort.

**6. References**

Chapter 8 counting sort and radix sort. *Introduction to algorithms third edition* – (Thomas H. Cormen, Charles E. Leiseron, Ronald L. Rivest, Clifford Stein)