[CENG 315 ALL Sections] Algorithms

THE1

Available from: Saturday, November 4, 2023, 11:59 AM
Due date: Sunday, November 5, 2023, 11:59 PM
Requested files: the1.cpp, test.cpp (Download)
Type of work: Individual work

Problem

In this exam, you are asked to complete the **quickSort()** function definition to sort the given array **arr** in **descending** order.

```
int quickSort(unsigned short* arr, long &swap, double & avg_dist, double & max_dist, bool hoare, bool median_of_3, int size);
```

You are expected to implement three variants of quickSort() in one function definition as follows:

- ***Quicksort with Lomuto Partitioning*** is called using the function **quickSort()** with **hoare=false**. You should use the Lomuto partitioning algorithm in the partition step. You can find the relevant pseudocode below.
- ***Quicksort with Hoare Partitioning*** is called using the function **quickSort()** with **hoare=true**. You should use the Hoare partitioning algorithm in the partition step. You can find the relevant pseudocode below.
- ***Quicksort with Median of 3 Pivot Selection*** is called using the function **quickSort()** with **median_of_3=true**. Before partitioning, you should select and arrange a better pivot according to the median of 3 pivot selection algorithm. It should work with the above two partitioning algorithms. It is a simple algorithm: First, find the median of the first, last, and middle *(same as Hoare's middle, meaning the index floor((size-1)/2))* elements. Then, swap this median with the element in the pivot position before calling the partition function. According to the partitioning algorithm, the pivot position may differ. If a swap occurs, update relevant control variables (**swap**, **avg_dist** etc.). ***Clarification:*** *You are not expected to perform any swap operations if there is no strict median.*

For all 3 tasks:
*You should sort the array in **descending** order, count the number of **swap**s executed during the sorting process, calculate the average distance between swap positions as **avg_dist**, find the max distance between swap positions as **max_dist** (both of which are 0 if no swap occurs). Finally, the **quickSort()** function should return the number of recursive calls.*

*You may notice that there will be swaps in which both sides are pointed by the **same** indexes during partitioning. You do not need to handle anything. Just like other swaps, apply the swap, increment your swap variable, and update your average distance.*

For partition tasks follow these pseudocodes exactly:

```
# PSEUDOCODE FOR QUICKSORT WITH CLASSICAL PARTITIONING
PARTITION(arr[0:size-1])

    X←arr[size-1]
    i←-1
    for j←0 to size-2                                  // The last element excluded
        do if arr[j]≥x
            then i←i+1
                 swap arr[i]↔arr[j]
    swap arr[i+1]↔arr[size-1]
    return i+1

QUICKSORT-CLASSICAL(arr[0:size-1])

    if size>1
        then P←PARTITION(arr[0:size-1])
             QUICKSORT-CLASSICAL(arr[0:P-1])         //P is excluded on recursive calls
             QUICKSORT-CLASSICAL(arr[P+1:size-1])
```

```
# PSEUDOCODE FOR QUICKSORT WITH HOARE PARTITIONING
HOARE(arr[0:size-1])

    X←arr[floor((size-1)/2)]       // i.e. 1 when size=3,4 ---- 2 when size=5,6
    i←-1
    j←size
    while True
        do  repeat j←j-1
                until arr[j]≥x
            repeat i←i+1
                until arr[i]≤x
            if i<j
                then swap arr[i]↔arr[j]
                else return j

QUICKSORT-HOARE(arr[0:size-1])

    if size>1
        then P←HOARE(arr[0:size-1])
             QUICKSORT-HOARE(arr[0:P])           //P is now included
             QUICKSORT-HOARE(arr[P+1:size-1])
```

Specifications:

- There are 3 **tasks** to be solved in **36 hours** in this take-home exam.
- You will implement your solutions in **the1.cpp** file.
- You are free to add other functions to **the1.cpp**
- Do **not** change the first line of **the1.cpp**, which is **#include "the1.h"**
- Do **not** change the arguments and return value of the functions **quickSort()** in the file **the1.cpp**
- Do **not** include any other library or write include anywhere in your **the1.cpp** file (not even in comments).

- You are given **test.cpp** file to **test** your work on **ODTUClass** or your **locale**. You can (and you are encouraged to) modify this file to add different test cases.
- If you want to **test** your work and see your outputs you can **compile** your work on your locale as:

```
>g++ test.cpp the1.cpp -Wall -std=c++11 -o test

> ./test
```

- You can test your **the1.cpp** on the virtual lab environment. If you click **run**, your function will be compiled and executed with **test.cpp**. If you click **evaluate**, you will get feedback for your current work and your work will be **temporarily** graded for a **limited** number of inputs.
- The grade you see in lab is **not** your final grade, your code will be reevaluated with **completely different** inputs after the exam.

The system has the following limits:

- a maximum execution time of 32 seconds (your functions should return in less than 1 seconds for the largest inputs)
- a 192 MB maximum memory limit
- an execution file size of 1M.
- Solutions with longer running times will not be graded.
- If you are sure that your solution works in the expected complexity constraints but your evaluation fails due to limits in the lab environment, the constant factors may be the problem.

Evaluation:

- After your exam, black box evaluation will be carried out. You will get full points if you set all the variables as stated.

Example IO:

1)

initial array = {4, 3, 2, 1}, size=4

sorted array = {4, 3, 2, 1}

Classical Lomuto partitioning -> swap=9, avg_dist=0, max_dist=0, n_calls=7

Classical Hoare Partitioning -> swap=0, avg_dist=0, max_dist=0, n_calls=7

2)

initial array = {1, 2, 3, 4} size=4

sorted array = {4, 3, 2, 1}

Classical Lomuto partitioning -> swap=5, avg_dist=0.8, max_dist=3, n_calls=7

Classical Hoare partitioning -> swap=2, avg_dist=2, max_dist=3, n_calls=7

Median of 3 Lomuto partitioning -> swap=6, avg_dist=0.833333, max_dist=2, n_calls=5

Median of 3 Hoare partitioning -> swap=2, avg_dist=2, max_dist=3, n_calls=7

3)

initial array = {5, 23, 3, 98, 45, 1, 90}, size=7

sorted array = {98, 90, 45, 23, 5, 3, 1}

Classical Lomuto partitioning -> swap=6, avg_dist=2.66667, max_dist=5, n_calls=9

Classical Hoare partitioning -> swap=6, avg_dist=1.83333, max_dist=4, n_calls=13

Median of 3 Lomuto partitioning -> swap=7, avg_dist=2.28571, max_dist=5, n_calls=7

Median of 3 Hoare partitioning -> swap=6, avg_dist=3, max_dist=6, n_calls=13

Requested files

the1.cpp

```cpp
#include "the1.h"

//You may write your own helper functions here

int quickSort(unsigned short* arr, long& swap, double& avg_dist, double& max_dist, bool hoare, bool median_of_3, int size){
    //Your code here

}

```

test.cpp