[CENG 315 ALL Sections] Algorithms

Navigation

- Dashboard
  - Site home
  - Site pages
  - My courses
    - 571 - Computer Engineering
      - CENG 223 All Sections
      - CENG 223 Section 2
      - CENG 315 ALL Sections
        - Participants
        - Competencies
        - Grades
        - General
        - October 2 - October 8
        - October 9 - October 15
        - October 15 - October 22
        - October 23 - October 29
        - October 30 - November 5
          - THE1
            - Description
            - Submission view
        - November 6 - November 12
        - November 13 - November 19
        - November 20 - November 26
        - November 27 - December 3
        - December 3 - December 10
        - December 11 - December 17
        - December 18 - December 24
        - December 25 - December 31
        - January 1 - January 7
        - January 8 - January 14
      - CENG 315 Section 1

Description | Submission view

Grade

Reviewed on Wednesday, November 29, 2023, 12:15 PM by Automatic grade
Grade: 100.00 / 100.00

Assessment report **[-]**
[-]**Output of make**
g++ -c -o sol1.o sol1.cpp
[-]**For input 01:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 02:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 03:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 04:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 05:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 06:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 07:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 08:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 09:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 10:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 11:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 12:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 13:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 14:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 15:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct

[-]**For input 16:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 17:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 18:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 19:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct
[-]**For input 20:**
Sorting: Correct
# of Swaps: Correct
Average Distance: Correct
Maximum Distance: Correct
# of Recursive Calls: Correct

Submitted on Sunday, November 5, 2023, 9:25 PM (Download)

the1.cpp

```cpp
#include "the1.h"

// swaps the elements by taking their adresses
void swapVar(unsigned short& var1,unsigned short& var2)
{
    unsigned short temp = var1;
    var1 = var2;
    var2 = temp;
}

// update the necessary variables after swap operation
void updateCtrlVar(int index1, int index2, long& swap, double& avg_dist, double& max_dist )
{
    int distance;
    if(index1<index2)
        distance = (index2 - index1);
    else
        distance = (index1 - index2);

    long double totalDistance = avg_dist*swap;
    totalDistance += distance;
    swap++;
    avg_dist = totalDistance/swap;
    if(distance>max_dist)
        max_dist = distance;

}

// take the median of first, middle and last elements put it in the pivot position
// the below operation doesn't take place if any two of the three values are equal
void medianOfThree(unsigned short* arr, long& swap, double& avg_dist, double& max_dist, bool hoare, int size)
{
    if(size <3) return;

    int midIndex, firstIndex = 0, lastIndex = size-1;

    //find the middile index
    if((size%2)==1)
    {
        midIndex = (size-1)/2;
    }
    else
    {
        midIndex = (size/2)-1;
    }

    if ((arr[firstIndex] < arr[midIndex] && arr[midIndex] < arr[lastIndex]) || (arr[lastIndex] < arr[midIndex] && arr[midIndex] < arr[firstIndex])) //check for middle one
    {
        //middle element is median
        if(hoare == false)          //swap middle element with last element if the sort type is lomuto
        {
            swapVar(arr[midIndex],arr[lastIndex]);
            updateCtrlVar(lastIndex,midIndex,swap,avg_dist,max_dist);
        }
    }
    else if ((arr[midIndex] < arr[firstIndex] && arr[firstIndex] < arr[lastIndex]) || (arr[lastIndex] < arr[firstIndex] && arr[firstIndex] < arr[midIndex])) //check for first one
    {
        //first element is median
        if(hoare == false)          //swap first element with last element if the sort type is lomuto
        {
            swapVar(arr[firstIndex],arr[lastIndex]);
            updateCtrlVar(lastIndex,firstIndex,swap,avg_dist,max_dist);
        }
        else                        //swap first element with middle element if the sort type is hoare
        {
            swapVar(arr[firstIndex],arr[midIndex]);
            updateCtrlVar(midIndex,firstIndex,swap,avg_dist,max_dist);
        }
    }
    else if ((arr[midIndex] < arr[lastIndex] && arr[lastIndex] < arr[firstIndex]) || (arr[firstIndex] < arr[lastIndex] && arr[lastIndex] < arr[midIndex])) //check for last one
    {
        //last element is median
        if(hoare == true)           //swap last element with middle element if the sort type is hoare
        {
            swapVar(arr[lastIndex],arr[midIndex]);
            updateCtrlVar(midIndex,lastIndex,swap,avg_dist,max_dist);
        }
    }
}

int LomutoPartition(unsigned short* arr, long& swap, double& avg_dist, double& max_dist, int size)
{
    unsigned short pivot = arr[size-1];
    int i = -1;
    for(int j=0; j<(size-1); j++)
    {
        if(arr[j] >= pivot)
        {
            i++;
            swapVar(arr[i],arr[j]);
            updateCtrlVar(i,j,swap,avg_dist,max_dist);
        }
    }
    swapVar(arr[i+1],arr[size-1]);
    updateCtrlVar(i+1,size-1,swap,avg_dist,max_dist);
    return(i+1);
}

int quickSortWithLomuto(unsigned short* arr, long& swap, double& avg_dist, double& max_dist, bool median_of_3, int size)
{
    int calls = 1;
    if (size >1)
    {
        if(median_of_3 == true)
            medianOfThree(arr, swap, avg_dist, max_dist, false, size);
        int pivotIndex = LomutoPartition(arr,swap,avg_dist,max_dist,size);
        unsigned short* arrLeft = &arr[0];
        unsigned short* arrRight = &arr[pivotIndex+1];

        calls += quickSortWithLomuto(arrLeft, swap, avg_dist,max_dist,median_of_3,pivotIndex);
        calls += quickSortWithLomuto(arrRight, swap, avg_dist,max_dist,median_of_3,size-pivotIndex-1);
    }

    return calls;
}

int HoarePartition(unsigned short* arr, long& swap, double& avg_dist, double& max_dist, int size)
{
    int midIndex;
    if(size%2==1)
        midIndex = (size-1)/2;
    else
        midIndex = (size)/2-1;
    unsigned short pivot = arr[midIndex];

    int i = -1;
```

```
    int j = size;

    while(true)
    {
        do{
            j--;
        }
        while(arr[j] < pivot);

        do{
            i++;
        }
        while(arr[i] > pivot);

        if(i<j)
        {
            swapVar(arr[i],arr[j]);
            updateCtrlVar(i,j,swap,avg_dist,max_dist);
        }
        else
        {
            return(j);
        }
    }
}

int quickSortWithHoare(unsigned short* arr, long& swap, double& avg_dist, double& max_dist, bool median_of_3, int size)
{
    int calls = 1;
    if (size >1)
    {
        if(median_of_3 == true)
            medianOfThree(arr, swap, avg_dist, max_dist, true, size);
        int pivotIndex = HoarePartition(arr,swap,avg_dist,max_dist,size);
        unsigned short* arrLeft = &arr[0];
        unsigned short* arrRight = &arr[pivotIndex+1];

        calls += quickSortWithHoare(arrLeft, swap, avg_dist,max_dist,median_of_3,pivotIndex+1);
        calls += quickSortWithHoare(arrRight, swap, avg_dist,max_dist,median_of_3,size-pivotIndex-1);
    }

    return calls;
}

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

    max_dist = 0;
    avg_dist = 0;
    swap = 0;

    if(hoare == true)
        return quickSortWithHoare(arr, swap, avg_dist,max_dist, median_of_3, size );
    else
        return quickSortWithLomuto(arr, swap, avg_dist,max_dist, median_of_3, size);

}
```

test.cpp

```
//This file is entirely for your test purposes.
//This will not be evaluated, you can change it and experiment with it as you want.
#include <iostream>
#include <fstream>
#include <random>
#include <ctime>
#include "the1.h"

using namespace std;

void randomFill(unsigned short*& arr, int size, unsigned short minval, unsigned short interval){
    arr = new unsigned short [size];
    for (int i=0; i <size; i++)
    {
        arr[i] = minval + (random() % interval);
    }
}

void print_to_file(unsigned short* arr, int size){
    ofstream ofile;
    ofile.open("sorted.txt");
    ofile<<size<<endl;
    for(int i=0;i<size; i++)
        ofile<<arr[i]<<endl;
}

void read_from_file(unsigned short*& input_array, int& size, bool& hoare, bool& median_of_3){

    char addr[]= "inp03.txt";
    ifstream infile (addr);
    if (!infile.is_open())
    {
        cout << "File \""<< addr
             << "\" can not be opened. Make sure that this file exists." <<endl;
        return;
    }
    infile >> hoare;
    infile >> median_of_3;
    infile >> size;
    input_array = new unsigned short[size];
    for(int j=0; j<size; j++){
        infile >> input_array[j];
    }

}

void test(){
    int size = 1 << 8;
    int number_of_recursive_calls;
    long swap=0;
    double avg_dist=0, max_dist=0;
    bool hoare=true, median_of_3=true;
    bool rand_fill = false;
    unsigned short* input_array;
    unsigned short minval=0;
    unsigned short interval= (unsigned short)((1<<16)-1);   // unsigned short 65535 in maximum , you can try to minimize interval for data generation to make your code test more equality conditions


    if(rand_fill)
        randomFill(input_array, size, minval, interval);    //Randomly generate initial array
    else
        read_from_file(input_array, size, hoare, median_of_3);

    //Calculate duration and print output
    std::cout << "Size: " << size << std::endl <<
                 "Median of three: " << median_of_3  << std::endl <<
                 "Hoare: " << hoare << std::endl <<
                 "Array elements: {";
    for(int idx=0; idx < size - 1; idx++) std::cout << input_array[idx] << ", ";
    std::cout << input_array[size-1] << "}" << std::endl;
    number_of_recursive_calls = quickSort(input_array, swap, avg_dist, max_dist, hoare, median_of_3, size);
    std::cout << "Swap: " << swap << std::endl <<
                 "Average Distance: " << avg_dist  << std::endl <<
                 "Maximum Distance: " << max_dist << std::endl <<
                 "Number of recursive calls: " << number_of_recursive_calls << std::endl <<
                 "Sorted array: {";
    for(int idx=0; idx<size-1; idx++) std::cout << input_array[idx] << ", ";
    std::cout << input_array[size-1] << "}" << std::endl;
}

int main(){
    srandom(time(0));
    test();
    return 0;
}
```

VPL