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])
```