Department of education and science of Ukraine

National technical university of Ukraine

«Kyiv polytechnic institute the name of Igor Sikorsky»

Faculty of informatics and computing engineering

Department of the computing engineering

Laboratory work №3

Discipline: «The algorithms theory»

Topic: «Quick sort method»

EXECUTED:

The first-year student

of FICT group IP-95

*Guskov Danil*

The Student book number- 9505

CHECKED:

Associate Professor

The Computer Engineering Department

PhD, SR

Antoniuk A.I.

Kyiv - 2020

**TASK**

**Goal:**

Implementation of three modifications of the Quick Sort algorithm and comparison of their performance.

**Task variant: 5**

Implement three modifications of the Quick Sort algorithm and compare their performance. The speed of the algorithms is compared on the basis of counting the number of comparisons of the array elements while the algorithms are running.

**SOFTWARE CODE**

**'use strict';**

**const swap = (arr, i, j) => {**

**const temp = arr[j];**

**arr[j] = arr[i];**

**arr[i] = temp;**

**return arr;**

**}**

**let speedCounter = 0;**

**const partition = (items, left, right) => {**

**let separator = items[Math.floor((right + left) / 2)],**

**q = separator,**

**i = left,**

**j = right;**

**while (i <= j) {**

**while (items[i] < q) {**

**speedCounter++;**

**i++;**

**}**

**while (items[j] > q) {**

**speedCounter++;**

**j--;**

**}**

**if (i <= j) {**

**swap(items, i, j);**

**i++;**

**j--;**

**}**

**}**

**return i;**

**}**

**const randomizePartition = (arr, low, high) => {**

**let q = low, i;**

**for (i = low; i < high; i++) {**

**speedCounter++;**

**if (arr[i] > arr[high]) {**

**swap(arr, i, q);**

**q++;**

**}**

**}**

**swap(arr, i, q);**

**return q;**

**}**

**const decomposeRandomize = (items, left, right) => {**

**left = left || 0;**

**right = right || items.length-1;**

**if (left < right) {**

**let index = randomizePartition(items, left, right);**

**decomposeRandomize(items, left, index - 1);**

**decomposeRandomize(items, index + 1, right);**

**}**

**return items;**

**};**

**const decomposeRecursive = (items, left, right) => {**

**let index;**

**left = left || 0;**

**right = right || items.length - 1;**

**if (items.length > 1) {**

**index = partition(items, left, right);**

**if (left < index - 1) {**

**decomposeRecursive(items, left, index - 1);**

**}**

**if (index < right) {**

**decomposeRecursive(items, index, right);**

**}**

**}**

**return items;**

**}**

**const decomposeRecursiveRandomize = (items) => {**

**return decomposeRandomize(items).reverse();**

**}**

**const decomposeNonRecursive = (arr) => {**

**if(!arr || 1 > arr.length) {**

**return null;**

**}**

**var arrLength = arr.length;**

**var startIndex = 0,**

**endIndex = arrLength - 1;**

**// don't use Array.push() and Array.pop() because too slow**

**// use 2 arrays instead of 1 to avoid unnecessary increasing and reducing stackLength**

**var stackStartIndex = [],**

**stackEndIndex = [];**

**var stackLength = 0;**

**var partitionIndex;**

**var i, j, is\_key;**

**do {**

**partitionIndex = partition(arr, startIndex, endIndex);**

**if(partitionIndex > startIndex) {**

**// there is lower values to partition**

**// is there higher values?**

**if(endIndex > partitionIndex + 1) {**

**// we don't do it now, push it into stack for later**

**stackStartIndex[stackLength] = partitionIndex + 1;**

**stackEndIndex[stackLength] = endIndex;**

**stackLength++; // increase counter for next slot**

**}**

**// set new parameter to partition lower values**

**endIndex = partitionIndex;**

**} else if(endIndex > partitionIndex + 1) {**

**// there is no lower values, only higher value, this is worst case!**

**// set new parameter for next partitioning**

**startIndex = partitionIndex + 1;**

**} else {**

**// no valid partitioning index, so we get from stack (if any)**

**if(stackLength > 0) {**

**stackLength--;**

**startIndex = stackStartIndex[stackLength];**

**endIndex = stackEndIndex[stackLength];**

**} else {**

**break; // finished !**

**}**

**}**

**} while(endIndex-1 > startIndex);**

**return arr;**

**}**

**const quickSorts = items => {**

**console.log('');**

**console.log('Input array: ');**

**console.dir(items);**

**console.log('');**

**console.log('QuickSort by Tony Hoare (classic recursive): ');**

**console.dir(decomposeRecursive(Array.from(items)));**

**console.log(`The comparisons of array elements (speed): ${speedCounter}`);**

**console.log('');**

**speedCounter = 0;**

**console.log('QuickSort by Nico Lomuto (recursive, randomize): ');**

**console.dir(decomposeRecursiveRandomize(Array.from(items)));**

**console.log(`The comparisons of array elements (speed): ${speedCounter}`);**

**console.log('');**

**speedCounter = 0;**

**console.log('QuickSort by Tony Hoare (non-recursive): ');**

**console.dir(decomposeNonRecursive(Array.from(items)));**

**console.log(`The comparisons of array elements (speed): ${speedCounter}`);**

**console.log('');**

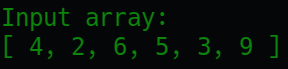
**speedCounter = 0;**

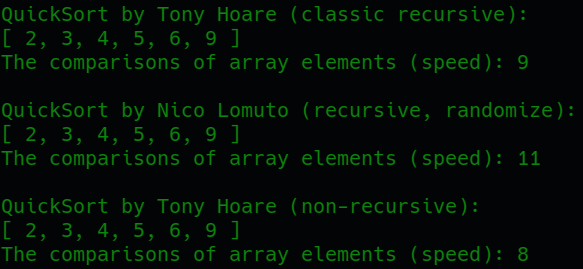
**}**

**const items = [4, 2, 6, 5, 3, 9];**

**quickSorts(items);**

**RESULTS OF THE PROGRAM WORK**

The input: 

Output: ****

**CONCLUSIONS**

Familiarized with the topic of laboratory work.

Have acquired relevant work skills.

An appropriate test program has been developed.

The results of the successful work of the test program above confirm the correctness of the chosen decisions, the ultimate goal of the work has been achieved.