**//steven guo**

**//selection sorting program**

**//10/20/19**

**import** java.util.Random;

**public** **class** selectionSort {

**static** // Java program for implementation of Selection Sort

**void** sort(**int** arr[])

{

**int** n = arr.length;

// One by one move boundary of unsorted subarray

**for** (**int** i = 0; i < n-1; i++)

{

// Find the minimum element in unsorted array

**int** min\_idx = i;

**for** (**int** j = i+1; j < n; j++)

**if** (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first

// element

**int** temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

//function that fills array with random number from 1-10000

**public** **static** **int**[] randArray(**int** size, **int**[] arr)

{

Random rand = **new** Random();

**int** number;

**for**(**int** i = 0; i < size-1; i++)

{

number = rand.nextInt(10000)+1;

arr[i] = number;

}

**return** arr;

}

//function that fills array with sorted numbers

**public** **static** **int**[] sortedArray(**int** size, **int**[] arr)

{

**for**(**int** i = 0; i < size-1; i++)

{

arr[i] = i;

}

**return** arr;

}

//function that fills array with sorted numbers except every 10th element is random

**public** **static** **int**[] almostSortedArray(**int** size, **int**[] arr)

{

Random rand = **new** Random();

**for**(**int** i = 0; i < size-1; i++)

{

**if**(i%10==0)

{

arr[i] = rand.nextInt(10000)+1;

}

**else**

{

arr[i] = i;

}

}

**return** arr;

}

**public** **static** **void** main(String arr[])

{

**final** **int** SIZE = 1000;

**final** **int** SIZE1 = 10000;

**final** **int** SIZE2 = 100000;

**long** time = System.*nanoTime*();

**int**[] arr1 = **new** **int**[SIZE];

**int**[] arr2 = **new** **int**[SIZE1];

**int**[] arr3 = **new** **int**[SIZE2];

*sort*(*randArray*(SIZE, arr1));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for random array with 1000 elements");

time = System.*nanoTime*();

*sort*(*randArray*(SIZE1, arr2));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for random array with 10000 elements");

time = System.*nanoTime*();

*sort*(*randArray*(SIZE2, arr3));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for random array with 100000 elements");

time = System.*nanoTime*();

*sort*(*sortedArray*(SIZE, arr1));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for sorted array with 1000 elements");

time = System.*nanoTime*();

*sort*(*sortedArray*(SIZE1, arr2));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for sorted array with 10000 elements");

time = System.*nanoTime*();

*sort*(*sortedArray*(SIZE2, arr3));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for sorted array with 100000 elements");

time = System.*nanoTime*();

*sort*(*almostSortedArray*(SIZE, arr1));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for almost sorted array with 1000 elements");

time = System.*nanoTime*();

*sort*(*almostSortedArray*(SIZE1, arr2));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for almost sorted array with 10000 elements");

time = System.*nanoTime*();

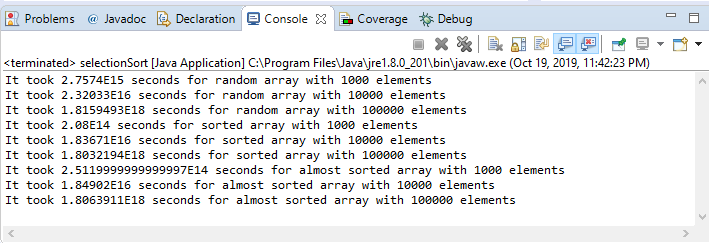
*sort*(*almostSortedArray*(SIZE2, arr3));

System.***out***.println("It took " + ((System.*nanoTime*() - time)/1e-9) + " seconds for almost sorted array with 100000 elements");

time = System.*nanoTime*();

}

}



The time complexity of selection sort is the same in all cases(O(n2)). At every step, the selection sort finds the smallest element and put it in the right place. The minimum element is not known until the end of the array is reached. The worst case is when we want to sort in ascending order and the array is in descending order. The best case is when the array is already sorted. The average case is when the elements are in jumbled order, so not ascending or descending. In theory all arrays with the same number of elements should have around the same time regardless of the numbers being ordered or not. From the results of my program I conclude that my theory is kind of true. When I ran the algorithm with 1000 elements in the array which have different ordering such as random, sorted, and almost sorted, the results hold up to the theory because all three of them had 2.3-2.7 seconds of run time. For the 10000 element result, the time is from 1.8-2.3. For the 100000 element result, the time is around 1.8 seconds. To my surprise, the arrays with large amounts of elements took less time to run. Either way all results are from 1.8-2.7 seconds which is kind of close. It’s a range of .9 seconds.