**//steven guo**

**//bubble sorting program**

**//10/20/19**

**import** java.util.Random;

**public** **class** bubbleSortWithout {

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

{

**int** n = arr.length;

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

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

**if** (arr[j] > arr[j+1])

{

// swap arr[j+1] and arr[i]

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

arr[j] = arr[j+1];

arr[j+1] = 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*();

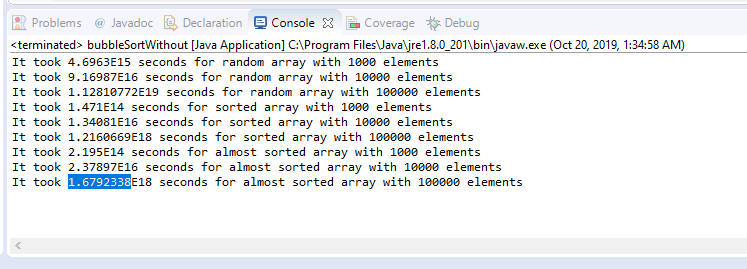
*sort*(*almostSortedArray*(SIZE2, arr3));

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 bubble sort without swap counting is a sorting algorithm which repeatedly goes through an array and compares adjacent elements and swaps them if they are in the wrong order. Bubble sort has a worst-case and average complexity of O(*n*2). The best case is O(n) and that is when the list is already sorted. The distance and direction that elements must move during the sort determine bubble sort's performance because elements move in different directions at different speeds. An element that must move toward the end of the list can move quickly because it can take part in successive swaps. For example, the largest element in the list will win every swap, so it moves to its sorted position on the first pass even if it starts near the beginning. On the other hand, an element that must move toward the beginning of the list cannot move faster than one step per pass, so elements move toward the beginning very slowly. If the smallest element is at the end of the list, it will take *n*−1 passes to move it to the beginning. My theory is that sorting a sorted list with bubble sort will be the fastest out of all the cases of random arrays and almost sorted arrays. This is true for the average of the results. The average for random arrays is 4.9 and 2 for almost sorted arrays while the sorted array has an average of 1.3.