# Problem and Program Briefly

# Generate Arrays to Test

## Random Arrays

The random arrays are generated with Java’s Math.random() method multiplied by an integer and then typecast to int. This gives a floored value from 0 to one less than the multiplied integer.

## Nearly Sorted Arrays

Nearly sorted arrays start out as perfectly sorted arrays from 0 to number of elements.

The array is then sorted into chunks of 4 elements (variable) and these 4 elements are then randomly swapped amongst themselves.

This creates a pseudo-random array where an element is never more than 4 spaces from its sorted index. As shown in fig 1.

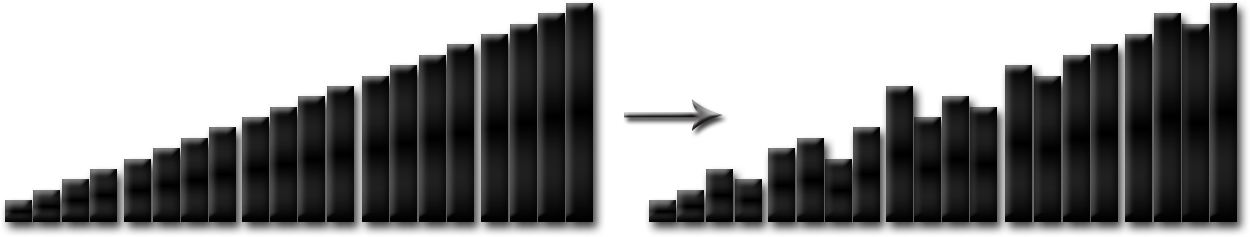


fig 1

## Descending (reverse order) Arrays

A descending array is made with a for loop, giving an incrementing int i that is subtracted from the length of the array. In effect:

descendingArray[i] = arraySize - i;

# Noise and Result Resolution

The fewer elements you have in arrays to be sorted, the less consistent any timing results become. It’s fair to speculate that other processes like setting up and tearing down the sorting algorithm takes too much time from the sorting itself to do decent comparisons.

Therefore, we must expect that too many digits following the decimal point will only reveal noise – even if taken as an average over a large sample size, this will be average noise.

During experimentations, we found no consistency with resolutions higher than 0.01 ms which is set as the cap for us.

As a note for time capture during tests, we used the java nanosecond counter to get good resolution with small arrays.

# Test Results

All tests were run a number of times as detailed in the Test Results below to get an average time value. This is done to minimize variation that might derive from process switching and array differences. The latter of which can be most impactful considering the possibility of a neatly sorted “random” test array.

# Test Observations

# Class Descriptions

## RunTests.class

Main() method   
The main() method implements a number of arrayProperties objects which is a value object containing size of test arrays, number of sorting iterations, and the difference span n of integers possible (from 0 to n-1).

executeTestsAndPrintAverageTime(arrayProperties arrayProperties, Sortable method)  
This method runs the sorting algorithm given as argument method. The sorting algorithm class must implement the Sortable interface which has one signature: “sort(Integer[] array)”.

The method then prints the average time to sort arrays in the output window.

## Tester.class

Tester constructor  
The constructor takes an arrayProperties object to initialize the array data.

run(Sortable sortMethod)  
This method has a for loop to run the sorting algorithm the number of times given by the arrayProperties object. The loop has a simple life of:

* Create a random array.
* Start nanosecond timer.
* Sort array.
* Stop nanosecond timer and increment elapsed time field.

This method will return the total elapsed time for all array sorts.

The interface argument is used to send on to the sort call.

## SelectionSort.class

sort(Integer[] arrayToSort)  
The method loads an array as an argument and with a for loop, it does the two things and then increments i:

* Find smallest element in the array, from element I (0 to start)
* Swap found element with element i

Last element in the array is not searched, as this must be in the right position.

When done the method returns the sorted array.

## InsertionSort.class

sort(Integer[] arrayToSort)  
The method loads an array as an argument and with a for loop that calls method shiftElements().  
When done the method returns the sorted array.

shiftElements()

This method first checks if the second object in the array is smaller than the previous element and saves the result to a boolean. Using this boolean it swaps the second element with the previous one and does the check again IF there are any elements left to check.

When the element is placed, the for loop in sort() is incremented and the next element is shifted back.

## BuiltInQuickSort.class

sort(Integer[] arrayToSort)  
The method runs java.util.Arrays.sort(array) on the input array and returns the sorted array.

## UnitTest Classes

Common for all test classes is that they each have a random array of 100 elements from 0 to 500 with a maximum of 3 repeated elements (element value ‘119’). And a sorted array of the same 100 elements.

The sort method is called and the resulting array is compared to the expected array.

The written code as of publication is attached as a printed appendix.