# Practical 4: Elementary Sorting

**What am I doing today?**

Today’s practical focuses on 3 things:

1. Writing several elementary sorting algorithms
2. Developing a testing framework to assess the performance of your algorithms
3. Summarizing the results

**Instructions**

Try all the questions. Ask for help from the demonstrators if you get stuck.

**\*\*\*Grading: Remember** if you complete the practical, add the code to your GitHub repo which needs to be submitted at the end of the course **for an extra 5%**

# Quick Questions

1. **How many compares does insertion sort make on an input array that is *already sorted*?**

|  |  |
| --- | --- |
| Constant |  |
| Logarithmic |  |
| Linear |  |
| Quadratic |  |

1. **What is a stable sorting algorithm?**

Sorting algorithm is said to be stable if the **order of the same values in the output remains the same as in input array**. This is an example of stable sorting, element 12 appears twice at index 5 and at index 7.

1. **What is an external sorting algorithm?**

**A. Algorithm that uses tape or disk during the sort**

1. **Identify 6 ways of classifying sorting algorithms?**

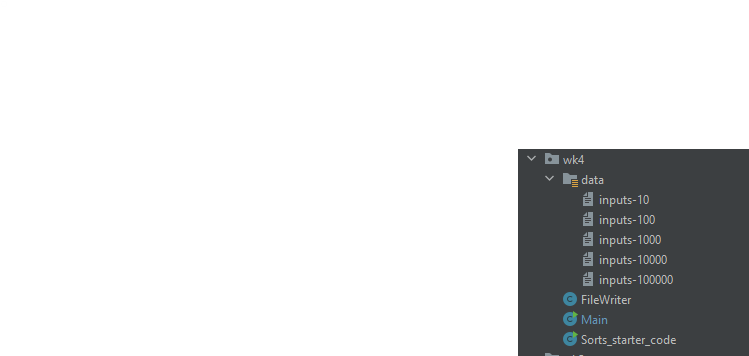
|  |  |
| --- | --- |
| 1. | Time Complexity |
| 2. | Space Complexity |
| 3. | Stability |
| 4. | Internal & External |
| 5. | Recursive & Non-Recursive |
| 6. | Comparison Sort |

Algorithmic Development

Today your mission is to develop a Java class that implements several elementary (and silly) sorting algorithms. The problem we want our algorithms to solve is sort an input array of integers into ascending order and output the resulting array.

Possible steps to follow

1. Create a new java class.



1. Implement the following sorting algorithms as public static functions within your class that take an array of integers and sorts the array, outputting a sorted array of integers:
   1. Selection sort

public static void selectionSort(int[] arr){  
 int min;  
 for (int i = 0; i < arr.length - 1; i++){  
 min = i;  
 for (int j =i+1; j < arr.length; j++){  
 // when element at j is less then the element at min set the new min to j;  
 if (arr[j] < arr[min]){  
 min = j;  
 }  
 }  
 //out of inner for loop - swap values when min dosen't equal i  
 if(min != i){  
 *helperSwap*(arr, i,min);  
 }  
 }  
}

* 1. Insertion Sort

public static void insertionSort(int[] arr){  
 int key, value;  
 for (int i = 1; i < arr.length; i++) { // int i = 1 bc there if i = 0 there is nothing to compare it with on the left  
 key = i;  
 value = arr[i];  
 //checking if the element at arr[i] is less than the element before it.  
 while((key>0) && (value < arr[key-1])){  
 // if so move that element one spot to the right  
 arr[key] = arr[key-1]; // moves element which is larger to the right  
 key--;  
 }  
 arr[key] = value;  
  
 }

1. A silly sort (either from the list below or of your own making)

public static void bogoSort(int[] arr){  
  
 int count = 1; // keeps track of shufflez.  
 while(!*helperIsSorted*(arr)){  
 *helperShuffle*(arr);  
 count++;  
 }  
 System.*out*.println(count + " shuffles were made to make this sort!");  
}

1. Create a simple framework for generating input arrays of various sizes (e.g., 10, 1000, 100,000) and then testing the performance over several runs.

public static void loadInputs(String pathname, int multiplier) {  
 try {  
 PrintWriter fileout = new PrintWriter(pathname);  
 int i;  
 for (i = 0; i < 10 \* multiplier; i++) {  
 fileout.println(i);  
 }  
 fileout.close();  
 System.*out*.print("success...");  
  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
}

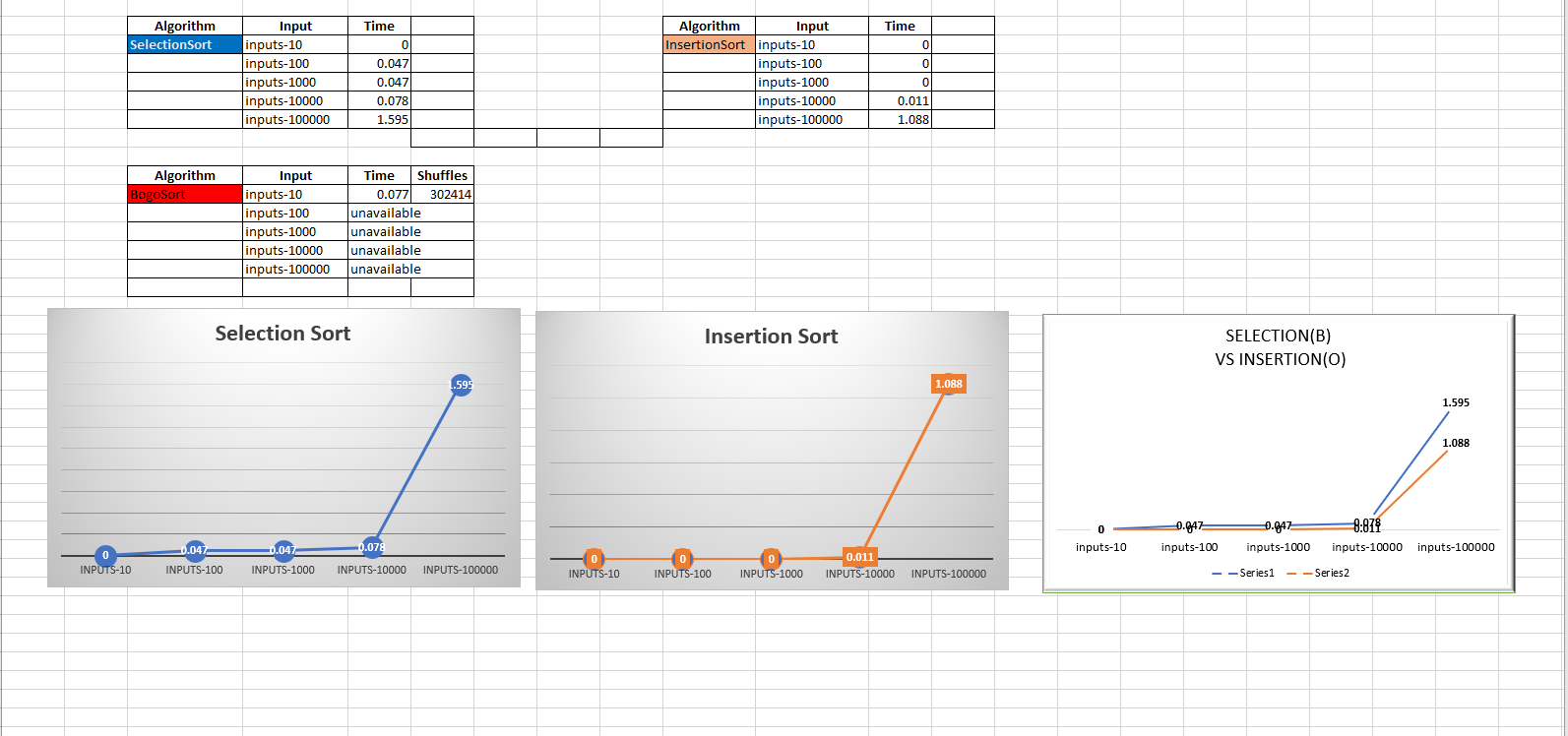
1. Print the resulting sorted array: Implement a function to print out all elements in the array

// prints array  
public static void helperPrintArr(int[] arr){  
 for (int i : arr){  
 System.*out*.print(i + " ");  
 }  
 System.*out*.println();  
}

1. Time the performance of the previous step on your 3 algorithms and output the execution times for various input sizes (e.g. 10,100,1000) on a graph

*\*Bogo Sort could only sort the 10 input file.*

Timing case for Selection sort  
Test 0 - 10 inputs  
 elapsed time = 0.0  
Test 1 - 100 inputs  
 elapsed time = 0.047  
Test 2 - 1000 inputs  
 elapsed time = 0.047  
Test 3 - 10000 inputs  
 elapsed time = 0.078  
Test 4 - 100000 inputs  
 elapsed time = 1.595  
  
Timing case for Insertion sort  
Test 0 - 10 inputs  
 elapsed time = 0.0  
Test 1 - 100 inputs  
 elapsed time = 0.0  
Test 2 - 1000 inputs  
 elapsed time = 0.0  
Test 3 - 10000 inputs  
 elapsed time = 0.011  
Test 4 - 100000 inputs  
 elapsed time = 1.088  
  
Timing case for Bogo sort  
302414 shuffles were made to make this sort!  
Test 0 - 10 inputs  
 elapsed time = 0.077



*\*excel document for graphs can be found in the Practical Resources folder*

*Practical-Reources\Analysis-Results\wk4-sorting.*

1. Justify the results of your experiments for the algorithms by proposing the algorithm complexity in big-O notation.
2. BONUS: adjust your insertion sort algorithm to be an unstable sort

public static void unstableInsertionSort(int[] arr){  
 int key, value;  
 for (int i = 1; i < arr.length; i++) { // int i = 1 bc there if i = 0 there is nothing to compare it with on the left  
 key = i;  
 value = arr[i];  
 //checking if the element at arr[i] is less than the element before it.  
 while((key>0) && (value <= arr[key-1])){  
 // if so move that element one spot to the right  
 arr[key] = arr[key-1]; // moves element which is larger to the right  
 key--;  
 }  
 arr[key] = value;  
  
 }  
}

By changing the condition of the while loop from

* While((key>0) && **( value < arr[key-1]))**

**To**

* While((key>0) && **( value <= arr[key-1]))**

I have now made this insertionSort unstable, as value which are of equal value will now be switching positions, even tho its makes no difference which element is before the other if they are equivalent.

## Java Timer Code Options

**Use System.currentTimeMillis()from the previous lab or implement nanoTime if you would like more precision.**

public static long nanoTime()

Returns the current value of the most precise available system timer, in nanoseconds.

This method can only be used to measure elapsed time and is not related to any other notion of system or wall-clock time. The value returned represents nanoseconds since some fixed but arbitrary *origin* time (perhaps in the future, so values may be negative). This method provides nanosecond precision, but not necessarily nanosecond accuracy. No guarantees are made about how frequently values change. Differences in successive calls that span greater than approximately 292 years (263 nanoseconds) will not accurately compute elapsed time due to numerical overflow.

For example, to measure how long some code takes to execute:

long startTime = System.nanoTime(); // ... the code being measured ... long estimatedTime = System.nanoTime() - startTime;

## Sorting Algorithms PseudoCode

**Selection Sort**

**Steps**

1. Find the smallest card. Swap it with the first card.
2. Find the second-smallest card. Swap it with the second card.
3. Find the third-smallest card. Swap it with the third card.
4. Repeat finding the next-smallest card, and swapping it into the correct position until the array is sorted.

**PseudoCode**

**function sort (int arr[]){**

int temp;

int mind\_index;

for (int i = 0; i < arr.length -1; i++){

min\_index = i;

for (j = i + 1; j < arr.length; j++){

if (arr[min\_index] > arr[j]{

min\_index = j;

}

//swap arr[i] & arr[min\_index]

temp = arr[i]

arr[i] = arr[min\_index]

arr[min\_index] = temp;

**InsertionSort**

**Steps**

1. The first step involves the comparison of the element in question with its adjacent element.
2. And if at every comparison reveals that the element in question can be inserted at a particular position, then space is created for it by shifting the other elements one position to the right and inserting the element at the suitable position.
3. The above procedure is repeated until all the elements in the array are in their correct position.

**PseudoCode**

**Function insertSort (int arr[]){**

for (int i = 1; i < n.length; ++i) {

key = arr[i];

j = i -1;

while j >= 0 and a[j] > key{

a[j+1] = a[j]

j = j-1

a[j+1] = key

}

}

}

### Some Silly Algorithms to pick from

**For fun or Computer Science comedic fun,** implement one of these obscure sorting algorithms and run it through the sequence of steps above:

* **BogoSort:** [**https://en.wikipedia.org/wiki/Bogosort**](https://en.wikipedia.org/wiki/Bogosort)

The stupidest sorting algorithm ever created?

* **Stalin Sort:** [**https://www.quora.com/What-is-Stalin-sort**](https://www.quora.com/What-is-Stalin-sort)

The Stalin sort is a joke sort in which elements that are out of order get removed from a list.

* **Slow Sort**: <https://en.wikipedia.org/wiki/Slowsort>

*“Slow sort is a sorting algorithm. It is of humorous nature and not useful”*

**\*Alternatively, develop your own stupid sorting algorithm**