Algorithms Lab Analysis

Precontext – All tests were conducted on a 9900k @ 4.9/4.3ghz, and 3600mhz/c18 DDR4 RAM, on Java 17.0.6

Algorithms had an upper time limit of 60 seconds before being considered as a fail.

**Insertion Sort:**

1. **What was the algorithms best case (in terms of data order)?**

* Ascending

1. **What was the algorithms worst case (in terms of data order)?**

* Descending

1. **What was the algorithms average case (in terms of data order)?**

* Random

1. **What is the big O efficiency of each algorithm? (Note: You may be able to calculate this, check our class notes, or do online research)**

* O(n²)

1. **Explain the “use case” situations for use each algorithm**

* Small arrays,
* Arrays that are already half-sorted,

1. **For each algorithm, what are the Pre and Post conditions?**

Preconditions:

* 1. Input of integer array with no null elements, of at least length 1
  2. Input of array descriptor (Arraytype),
  3. Arrays must be of a length small enough for the CPU to be able to sort the data within 60 seconds

Postconditions:

* 1. Array is sorted in ascending order
  2. Time results + descriptor saved to results.txt OR Array isn’t sorted and descriptor and results up to 60 seconds are saved to results.txt
  3. Returns number of iterations (unused, too lazy to make methods void)

**Bubble Sort:**

1. **What was the algorithms best case (in terms of data order)?**

* Random

1. **What was the algorithms worst case (in terms of data order)?**

* Descending

1. **What was the algorithms average case (in terms of data order)?**

* Ascending

1. **What is the big O efficiency of each algorithm? (Note: You may be able to calculate this, check our class notes, or do online research)**

* O(n^2)

1. **Explain the “use case” situations for use each algorithm**

* Educational introduction to sorting algorithms (it is fairly basic)
* Small arrays

1. **For each algorithm, what are the Pre and Post conditions?**

Preconditions:

* 1. Input of integer array with no null elements, of at least length 1
  2. Input of array descriptor (Arraytype),
  3. Arrays must be of a length small enough for the CPU to be able to sort the data within 60 seconds

Postconditions:

* 1. Array is sorted in ascending order
  2. Time results + descriptor saved to results.txt OR Array isn’t sorted and descriptor and results up to 60 seconds are saved to results.txt
  3. Returns number of iterations

Selection Sort:

1. **What was the algorithms best case (in terms of data order)?**

* Descending (based on time, iterations # was the same for all, so therefore all three data order types are equally computationally intensive)

1. **What was the algorithms worst case (in terms of data order)?**

* Random (based on time, iterations # was the same for all, so therefore all three data order types are equally computationally intensive)

1. **What was the algorithms average case (in terms of data order)?**

* Ascending (based on time, iterations # was the same for all, so therefore all three data order types are equally computationally intensive)

1. **What is the big O efficiency of each algorithm? (Note: You may be able to calculate this, check our class notes, or do online research)**

* O(n^2)

1. **Explain the “use case” situations for use each algorithm**

* When number of iterations has to be kept to a minimujm
* Small arrays

1. **For each algorithm, what are the Pre and Post conditions?**

Preconditions:

* 1. Input of integer array with no null elements, of at least length 1
  2. Input of array descriptor (Arraytype),
  3. Arrays must be of a length small enough for the CPU to be able to sort the data within 60 seconds

Postconditions:

* 1. Array is sorted in ascending order
  2. Time results + descriptor saved to results.txt OR Array isn’t sorted and descriptor and results up to 60 seconds are saved to results.txt
  3. Returns number of iterations

Gnome Sort:

1. **What was the algorithms best case (in terms of data order)?**

* Random / descending (both same)

1. **What was the algorithms worst case (in terms of data order)?**

* Ascending (based on time)

1. **What was the algorithms average case (in terms of data order)?**

* Random / descending (both same)

1. **What is the big O efficiency of each algorithm? (Note: You may be able to calculate this, check our class notes, or do online research)**

* O(n^2)

1. **Explain the “use case” situations for use each algorithm**

* Small arrays
* Educational introduction to sorting algorithms (it is a fairly basic iteration of bubblesort)

1. **For each algorithm, what are the Pre and Post conditions?**

Preconditions:

* 1. Input of integer array with no null elements, of at least length 1
  2. Input of array descriptor (Arraytype),
  3. Arrays must be of a length small enough for the CPU to be able to sort the data within 60 seconds

Postconditions:

* 1. Array is sorted in ascending order
  2. Time results + descriptor saved to results.txt OR Array isn’t sorted and descriptor and results up to 60 seconds are saved to results.txt
  3. Returns number of iterations

**Universal questions:**

**Which algorithm did you find most efficient in terms of difficulty to implement (ie. Which algorithm was the most “worth it” based on the effort and results)? Why was this the case?**

Gnomesort was the most worth it to implement, as it created the most consistent results for all three of the array types and was relatively indifficult to create an implementation of.

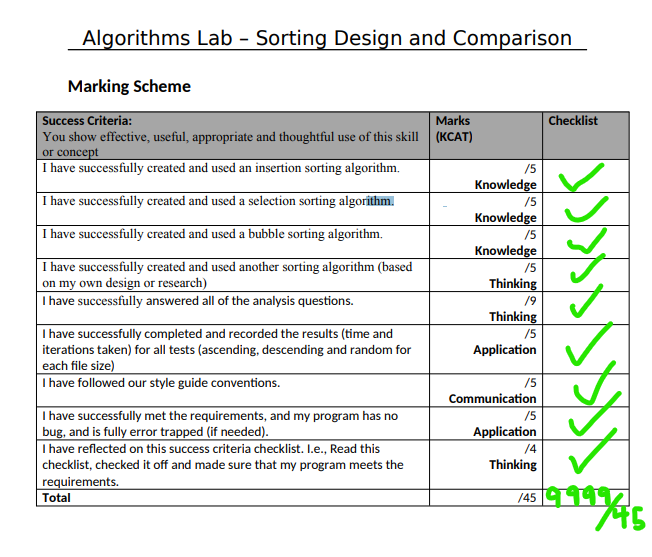
It also has the same big O efficiency of O(n^2) as all the other algorithms.

**For each data order type, what algorithm(s) was the fastest in run time and iterations?**

* Ascending order: Insertionsort
* Descending order: Gnomesort
* Random order: Gnomesort

**For each data order type, what algorithm(s) was the slowest in run time and iterations?**

* Ascending order: Bubblesort
* Descending order: Bubblesort
* Random order: Bubblesort



**Sources:**

<https://developer.nvidia.com/blog/insertion-sort-explained-a-data-scientists-algorithm-guide/#:~:text=The%20worst%2Dcase%20(and%20average,O(n)%20time%20complexity>.

<https://www.happycoders.eu/algorithms/bubble-sort/>

<https://dev.to/pbillingsby/basic-big-o-notation-and-selection-sort-3j3b>

<https://iq.opengenus.org/gnome-sort/>