PRACTICAL 1 – FILES AND SORTING

Thi Van Anh DUONG ID 90023112

Diploma of Information Technology, Curtin College

DSA1002: Data Structures and Algorithms

Coordinator: Khurram Hameed

10 July 2022

Student Declaration of Originality

☒	This assignment is my own original work, and no part has been copied from another student's work or source, except where it is clearly attributed,
☒	All facts/ideas/claims are from academic sources are paraphrased and cited/referenced correctly,
☒	I have not previously submitted this work in any form for THIS or for any other unit; or published it elsewhere before
☒	No part of this work has been written for me by another person,
☒	I recognise that should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with Curtin College policy.

Electronic Submission:

☒	I accept that it is my responsibility to check that the submitted file is the correct file, is readable and has not been corrupted,
☒	I acknowledge that a submitted file that is unable to be read cannot be marked and will be treated as a non-submission
☒	I hold a copy of this work if the original is damaged, and will retain a copy of the Turnitin receipt as evidence of submission

Activity 4: Exploring Run Times

Table of runtime results

Sorting Algorithms	Number of elements (N)	Type of order (T)	Time (ms)
Bubble Sort	10 With an array already sorted `A = [1,2,3,4,5,6,7,8,9,10]`	Ascending	0.000035400390625
	200	Ascending	0.000212037501114537
		Descending	0.030484861998047563
		In random order	0.04286020149993419
		Nearly sorted (10% moved)	0.01147370350008714
	500	Ascending	0.0004650679984479211
		Descending	0.15259941650037945
		In random order	0.16560297400064883
		Nearly sorted (10% moved)	0.0712360034995072
	1000	Ascending	0.0009756679992278805
		Descending	0.5825606064991007
		In random order	0.5652005100000679
		Nearly sorted (10% moved)	0.3433270410005207
	2000	Ascending	0.002666810001755948
		Descending	2.3219145069997467
		In random order	1.9636031145000743
		Nearly sorted (10% moved)	1.0529704769996897
Insertion Sort	10 With an array already sorted `A = [1,2,3,4,5,6,7,8,9,10]`	Ascending	0.000035888671875
	200	Ascending	0.000212037501114537
		Descending	0.012818621999031166

		In random order	0.030079204499998013
		Nearly sorted (10% moved)	0.0015529485026490875
	500	Ascending	0.000631563500064658
		Descending	0.08429857299961441
		In random order	0.10659382249832561
		Nearly sorted (10% moved)	0.006661251500190701
	1000	Ascending	0.0012550390001706546
		Descending	0.32241830400198523
		In random order	0.29209448950314254
		Nearly sorted (10% moved)	0.03377994999937073
	2000	Ascending	0.0023620770007255487
		Descending	1.3133213594992412
		In random order	0.9992744999999559
		Nearly sorted (10% moved)	0.09164916200279549
Selection Sort	10 With an array already sorted `A = [1,2,3,4,5,6,7,8,9,10]`	Ascending	0.00013427734375
	200	Ascending	0.008318172498547938
		Descending	0.009403808500792366
		In random order	0.03307227550067182
		Nearly sorted (10% moved)	0.009885920500892098
	500	Ascending	0.057209859500289895
		Descending	0.06439673449858674
		In random order	0.1460182544997224
		Nearly sorted (10% moved)	0.06502334849756153
	1000	Ascending	0.2251995939986955
		Descending	0.2363708305019827

		In random order	0.36627766050150967
		Nearly sorted (10% moved)	0.23883264700089057
	2000	Ascending	0.8878946930017264
		Descending	0.9516805079983897
		In random order	1.1731226424981287
		Nearly sorted (10% moved)	0.9145098735007196

SUMMARY

1. *Time complexity:*

From my observation: The lower it takes to complete sorting, the higher complexity level of the sort algorithms.

- With the same number of elements, same sort type:

Example 1:

Best Case: N = 10, T = Ascending

⇨ Bubble sort: 0.000035400390625~ 0 = Insertion sort: 0.000035888671875~0 < Selection Sort: 0.00013427734375

Worst Case:

```
A = [3,2,4,5,1,7,9,6,8,10]
```

Time execution: Bubble sort: 0.00012255859375 Insertion sort: 0.00014013671875 Selection Sort: 0.000134521484375

⇨ Selection sort has the same amount of time (react similarly)

Average Case, Worst Case (random data being sorted from the table): the bigger data is, the more amount of the each type of sort taking to sort the data.

For example:

N = 1000, Descending:

⇨ Bubble sort: 0.5825606064991007 > Insertion sort: 0.32241830400198523 > Selections sort: 0.2363708305019827

N = 200, Ascending:

Bubble sort: 0.000212037501114537 ~ Insertion sort: 0.000212037501114537

⇨ < Selections sort: 0.008318172498547938

Time taken increasing means the time complexity of them increase. I will discuss more closely in the table below.

Sorting Algorithms	Time Complexity		
	Best Case	**Average Case**	**Worst Case**
Bubble Sort	O(N) – the array already sorted, we will stop comparing and swap after a single pass. (demonstrate in example 1)	$O(N^2)$	$O(N^2)$ – we have to make all comparisons and swap for each pass
Insertion Sort	O(N) - the array already sorted, each element is placed in the right order so no swap required (demonstrate in example 1)	$O(N^2)$	$O(N^2)$ - we have to make all comparisons and swap for each pass
Selection Sort	$O(N^2)$ – eventhough the array already sorted, we have to find the minimum value for each loop. So it takes time to go through the entire array. (demonstrate in example 1)	$O(N^2)$	$O(N^2)$ – in order to find the minimum value for each loop, we have to go through the entire array.

2. *Stability*

Stable sort: is when it sorts the identical elements in the same order as the input.
Unstable sort: is when the identical elements is not guaranteed in the same order as they are in input.

Assume I have an array: A = [5,3,4,5,2]
I distinguish the first element 5 with the second one.
A = [5,3,4,5',2]

With Bubble sort:
A = [5,3,4,5',2] ➔ A = [3,5,4,5',2] ➔ A = [3,4,5,5',2] ➔ A = [3,4,5,2,5'] ➔ … ➔ **A = [2,3,4,5,5']**
(5 comes before 5')
➔ Bubble sore is stable.

With Insertion sort:
A = [5,3,4,5',2] ➔ A = [3,5,4,5',2] ➔ A = [3,4,5,5',2] ➔ A = [3,4,5,2,5'] ➔ **A = [2,3,4,5,5']**
➔ Insertion sort is stable

With Selection sort:
After 1 iteration, 2 will swap with 5:
A = [2,3,4,5',5]
➔ selection sort is unstable because 5 should come before 5' in the input array.

3. *How different type of sorts work:*

Bubble sort: adaptable – can perform lesser comparison and swap per iteration if the data are already sorted or nearly sorted (see explanation in Time Complexity).
Insertion sort: adaptable - can perform lesser comparison and swap per iteration if the data are already sorted or nearly sorted (see explanation in Time Complexity).
Selection sort: work similarly for all types of data sizes.

4. *Advantages and Disadvantages*

Sorting Algorithms	***Advantages***	***Disadvantages***
Bubble Sort	+ Simple to understand, easy to code + Sorting data quickly just in case the data partially be sorted and already sorted + Stable sort	+ Worst Case: $O(N^2)$ => poor speed + Making lots of comparisons and swaps
Insertion Sort	+ Sorting data quickly just in case the data partially be sorted and already sorted (compared to selection sort) + Stable sort	+ Making lots of swap
Selection Sort	+ Less effort per iteration (just find the min value in unsorted array rather than compare all elements) + Easy to code.	+ Unstable sort + Similarly sorting in best case, average case and worst case which takes much time (swaps, comparisons)