CAB301 Algorithms and Complexity

Empirical Comparison of Two Algorithms for finding the distance between two closest elements in an array of numbers

Date submitted: 28th May 2017

Table of contents

1 Summary

In this assignment, two algorithms which perform the same function will be compared against each other in terms of run-time characteristics and theoretical predictions.

Both algorithms were compared using the same problem size, computational environment

2 Description of the Algorithms

2.1 MinDistance Algorithm

MinDistance algorithm goes through an array and checks every element in the array against each other. It subtracts the elements against each other and if the distance is less than the previous distance then the value is added to the dmin variable.

```
Algorithm MinDistance(A[0..n − 1])
//Input: Array A[0..n − 1] of numbers
//Output: Minimum distance between two of its elements
dmin ← ∞
for i ← 0 to n − 1 do
    for j ← 0 to n − 1 do
        if i ≠ j and |A[i] − A[j]| < dmin
            dmin ← |A[i] − A[j]|
return dmin
```

2.2 MinDistance2 Algorithm

MinDistance2 algorithm goes through the array and checks one element against all other elements in the array, Unlike MinDistance that goes through every element, MinDistance2 only compares the same pair once and it also avoids going through the same expression in the innermost loop. And subtracts the elements against each other and if the distance is less than the previous distance it is added to dmin.

```
Algorithm MinDistance2(A[0..n − 1])
//Input: An array A[0..n − 1] of numbers
//Output: The minimum distance d between two of its elements
dmin ← ∞
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        temp ← |A[i] − A[j]|
        if temp < dmin
            dmin ← temp
return dmin
```

3 Theoretical Analysis of the Algorithm

3.1 Basic Operation

As stated in the specification sheet of this assignment, the basic operation for both algorithms must be the same, which in this situation is chosen to be the innermost loop. The number displayed by

the basic operation counter is the number of times the for loop executes before it comes up with a solution and displays the distance between two closest elements in an array. Comparing the logic of both for-loops, we can see that the first algorithm starts at 0 and then compares it to the same element, which is 0. Algorithm 2 has a different approach, whereas it starts the innermost for-loop at i+1, making it less redundant. Even though the coding in both situations vary, they still serve the same purpose. Basic operations for the algorithms are as shown below:

Basic operation of MinDistance:

```
for (auto j = 0; j <= n - 1; j++) {
      if (( i != j ) && ( abs(A[i] - A[j]) < dmin )) {
            dmin = abs(A[i] - A[j]);
      }
      *basicOp1 = *basicOp1 + 1;
}
```

Basic operation of MinDistance2:

```
for (auto j = i + 1; j <= n - 1; j++){
auto temp = abs(A[i] - A[j]);
      if (temp < dmin) {
            dmin = temp;
      }
      *basicOp2 = *basicOp2 + 1;
}
```

3.2 Problem Size

The problem size for both algorithms was chosen to be randomly generated arrays with the size of 0 to 10000 elements. A distribution of tests within a range of 10 000 elements is enough to show the behaviour of an algorithm in different computational tasks. The array is unsorted and is incremented by 1 element each iteration.

3.3 Average-case efficiency MinDistance

The average-case efficiency of the MinDistance algorithm is the following.

$$C_{avg} = \sum_{i=0}^{n-1} n - 1 = n^2+1$$

The result is $\in \Theta(n^2)$

3.4 Average-case efficiency MinDistance2

MinDistance2 algorithm has 2 for-loops; an inner-loop and an outer-loop. The outer loop starts at i = 0 and runs n -1 times, as shown in the figure below.

$$\sum_{i=0}^{n-1}$$

The second loop, the inner-loop is dependent on the outer one since it starts iterating from j = i +1 and executes n -1 times, as shown in the figure below.

$$\sum_{j=i+1}^{n}$$

Taking the previous findings into consideration, the Average case-efficiency is as follows.

$$C_{\text{avg}} = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} = \frac{n(n-1)}{2}$$

The result is $\in \Theta(n)$.

4 Methodology, Tools and Techniques

4.1 Programming Language

Programming language used to implement the algorithm as well as its' testing was written in C++, as required by the specification of the assignment. The code was written in CodeBlocks IDE with the GNU GCC 6.3.1 compiler, as well as C++11 chrono timer for maximum precision time values.

4.2 Computing Environment

The assignment's computational tasks and testing was performed on a Dell XPS 13 running Arch Linux x86_64 (kernel 4.10.13-ARCH), i5-5200U (2.7GHz), 8GB RAM.

4.3 Graphical Presentation

All tests presented in this report were done by writing output data to a csv file and then graphically displaying in Excel. This way all the data could be efficiently stored, sorted and displayed in a file and later converted to a graph.
Mathematic equations were written with Microsoft Words' built-in functionality.

5 Experimental Result

An array of 10000 elements was chosen to be used for the following tests.

5.2 Functional Testing

To test that the algorithm was correctly implemented a number of tests were performed. The code and the result can be seen in appendix___. The result of the tests is:

Test 1, Testing an array with 1 element. It shows that the algorithm won't compare it, and returns dmin = numeric_limits<int>::max();.

{1}

Min distance1: 2147483647 Basic operations: 1

Min distance2: 2147483647 Basic operations: 0

Test 2, Testing an array of 2 elements. The results are as expected and the MinDisitance2 algorithm compares the pair only once.

{1,2}

Min distance1: 1 Basic operations: 4

Min distance2: 1 Basic operations: 1

Test 3, Testing an array of 3 elements.

{1, 2, 4}

Min distance1: 1 Basic operations: 9

Min distance2: 1 Basic operations: 3

Test 4, Testing an array of 5 elements.

{1,3,6,9,22}

Min distance1: 2 Basic operations: 25

Min distance2: 2 Basic operations: 10

Test 5, Testing an array of 5 elements with the same value.

{3,3,3,3,3}

Min distance1: 0 Basic operations: 25

Min distance2: 0 Basic operations: 10

Test 6, Testing an empty array, None of the algorithm runs, and returns dmin = numeric_limits<int>::max();.

{}

Min distance1: 2147483647 Basic operations: 0

Min distance2: 2147483647 Basic operations: 0

Test 7, Testing an array of 12 elements, descending.

{100,90,80,70,60,55,50,40,30,20,10,0}

Min distance1: 5 Basic operations: 177

Min distance2: 5 Basic operations: 99

5.2 Average-Case execution Time

6 Conclusion

7 Source Code

Attached to the report folder

8 Appendix

Figure 1a – Execution Time of MinDistance

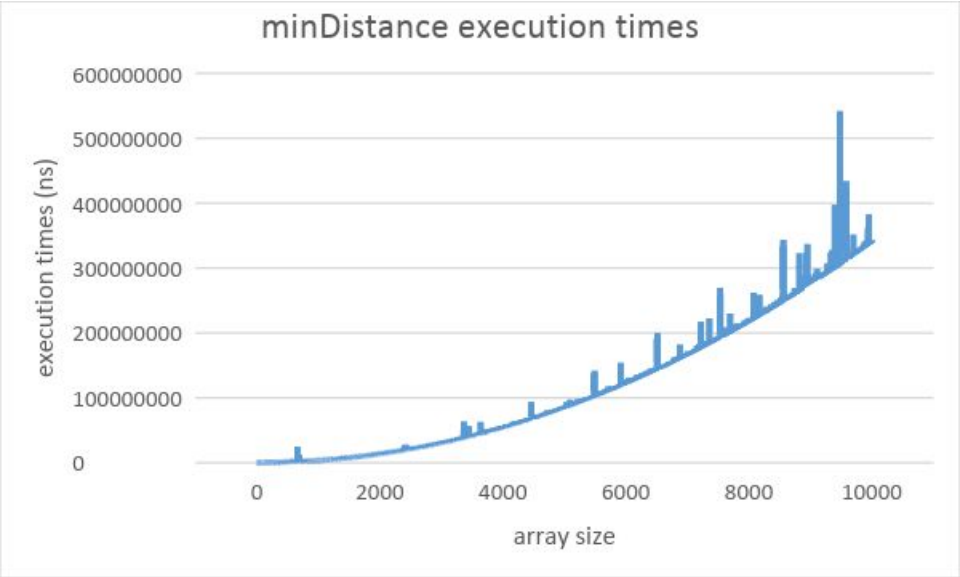


Figure 1b – Execution Time MinDistance2

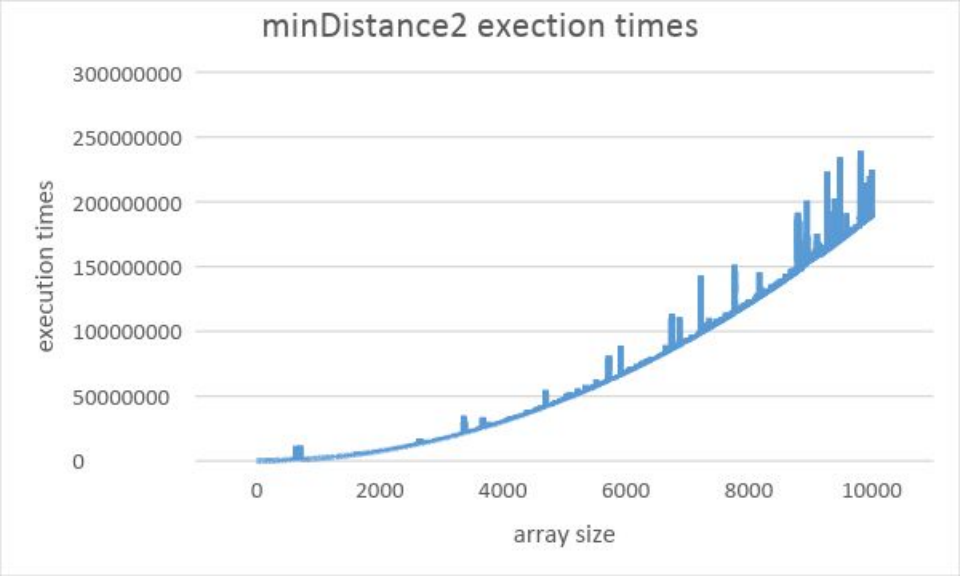


Figure 2a – Basic operations of MinDistance

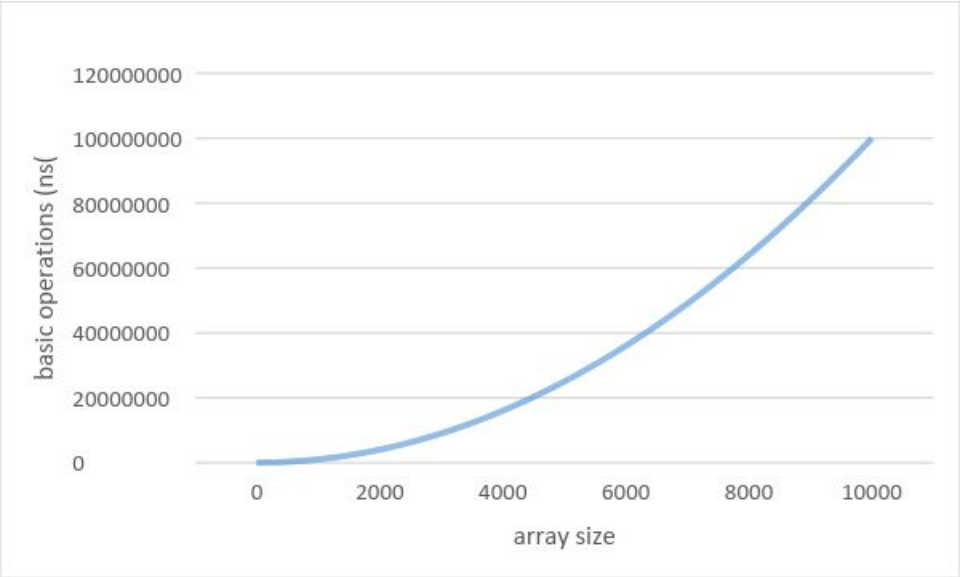


Figure 2b – Basic operations of MinDistance2

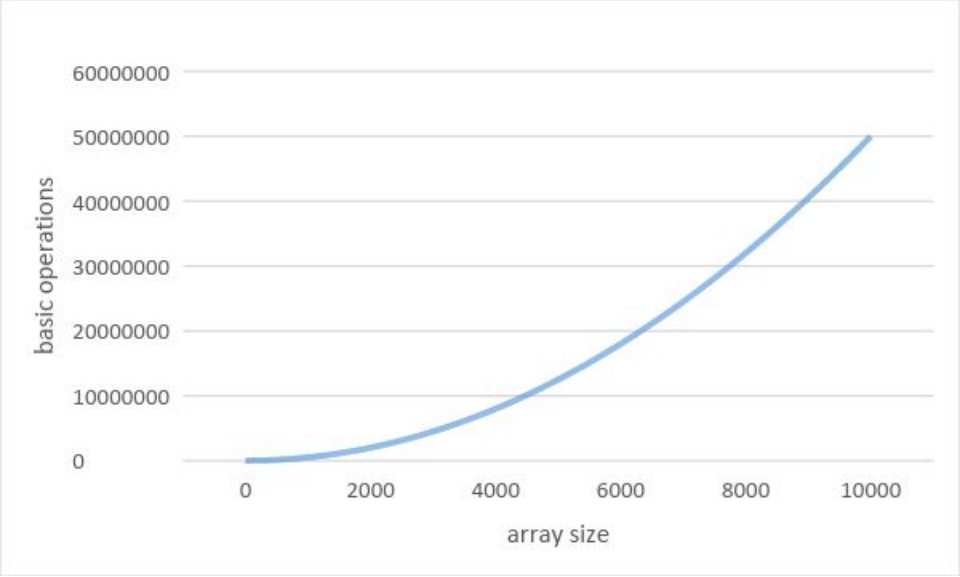
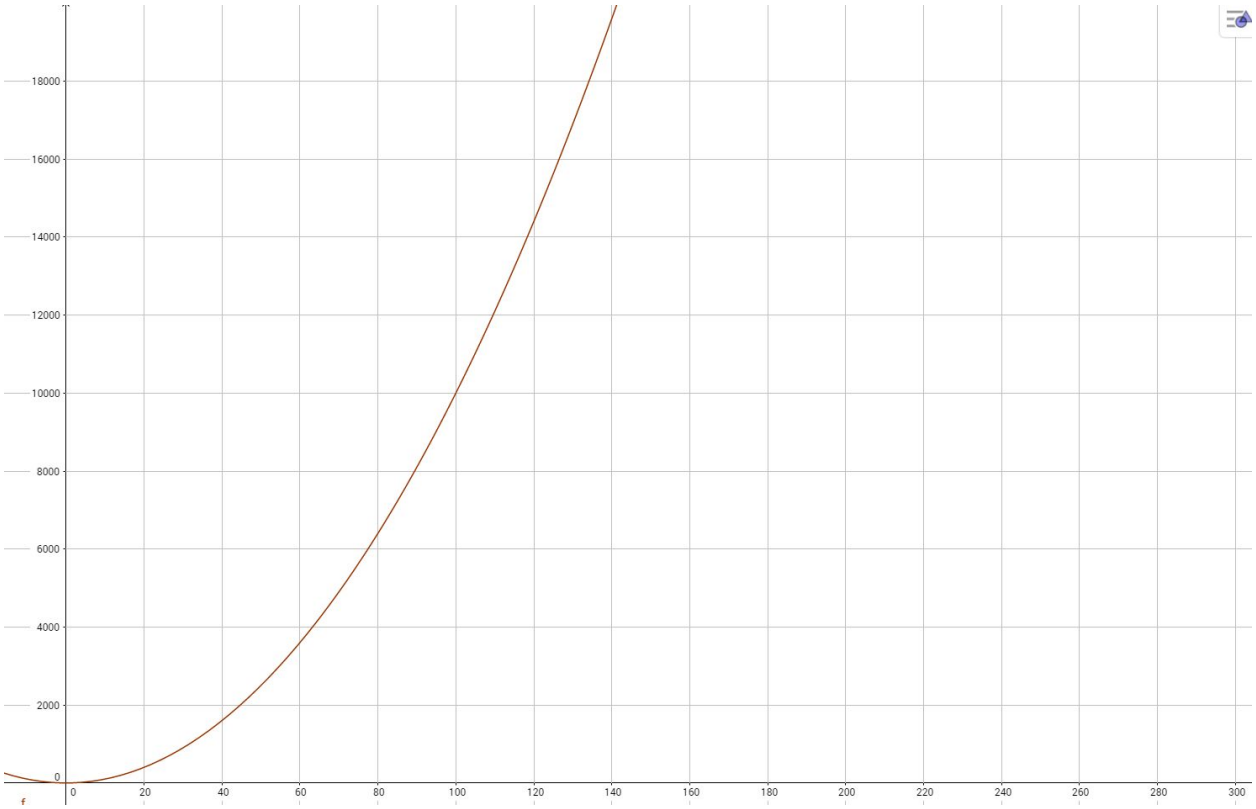


Figure 3a – Comparison execution times

Figure 3b – Comparison basic operations

Figure 5 – n^2 (f(x)= x^2)



5000	85408752	25000000	5000	47082799	12497500
5001	85700463	25010001	5001	47406092	12502500
5002	84933728	25020004	5002	47468475	12507501
5003	85199744	25030009	5003	47153642	12512503
5004	85686600	25040016	5004	47172523	12517506
5005	86000126	25050025	5005	47111373	12522510
5006	84969132	25060036	5006	47348097	12527515
5007	85138623	25070049	5007	47791546	12532521
5008	85133608	25080064	5008	47284054	12537528
5009	86094902	25090081	5009	47226176	12542536
5010	85224755	25100100	5010	47466538	12547545
5011	85171848	25110121	5011	47858413	12552555
5012	85220280	25120144	5012	49283649	12557566
5013	85729985	25130169	5013	47368160	12562578
5014	85453299	25140196	5014	47378418	12567591
5015	86025360	25150225	5015	47416468	12572605
5016	85665657	25160256	5016	48374053	12577620
5017	85380344	25170289	5017	47711488	12582636
5018	85892934	25180324	5018	47353005	12587653
5019	85986444	25190361	5019	47435154	12592671
5020	86387399	25200400	5020	47426823	12597690
5021	85620156	25210441	5021	47614259	12602710
5022	85505763	25220484	5022	48139548	12607731
5023	85747450	25230529	5023	47495972	12612753
5024	86720740	25240576	5024	47553889	12617776
5025	85750804	25250625	5025	48148749	12622800
5026	87124327	25260676	5026	49162449	12627825
5027	88180733	25270729	5027	48555104	12632851
5028	85721422	25280784	5028	47699116	12637878
5029	86221349	25290841	5029	47533958	12642906
5030	85978523	25300900	5030	47572064	12647935
5031	86582911	25310961	5031	47866128	12652965
5032	85763590	25321024	5032	47714403	12657996
5033	86115107	25331089	5033	47653117	12663028
5034	86245591	25341156	5034	47751497	12668061
5035	86827304	25351225	5035	47679872	12673095

Figure 4 – csv data file excerpt, (MinDistance/MinDistance2 array size, execution time, basic operations)

9 References