UNIVERSITI SAINS MALAYSIA

CPT212 Design & Analysis of Algorithms
Semester II, Academic Session 2022/2023

ASSIGNMENT I: Principles of Analysis of Algorithms and Sorting Methods

Name	Matric No.	USM Email
OOI YUE SHENG	158494	shengtheodore@student.usm.my
YEO YING SHENG	157627	yeousm@student.usm.my
DEW KHAI YEK	158795	dewkhaiyek@student.usm.my

Task Deadline
17th May 2023 11.59 pm(Wednesday)

Contents

Question 1

The full program for question 1 is in the **CPT212 Question 1** folder, src/**RadixSort**.java file. The following sample inputs are run in IntelliJ compiler. For easier testing and analysis, the program will randomly generate an array based on the user's input.

Input:

The number of elements to be sorted. (Ex. 4)

Minimum value of element in the randomly generated array.

Maximum value of element in the randomly generated array.

Output:

The unsorted list of integers. (Ex. 5 123123 7 1)

Output between each pass.

The sorted list of integers. (Ex. 1 5 7 123123)

Constraint:

$0 \leq x < 2147483647$

$x\ is\ the\ value\ of element\ inside\ the\ array$

Sample output

(See Appendix: Question 1)

Question 2

The full program for question 2 is in the **CPT212 Question 2** folder, src/**RadixSort**.java file. The following sample inputs are run in IntelliJ compiler. For easier testing and analysis, the program will randomly generate an array based on the user's input.

Input:

The number of elements to be sorted. (Ex. 3)

The number of digit(s) for the integer part. (Ex. 0)

The number of digit(s) for the decimal part. (Ex. 2)

Output:

The unsorted list of integers. (Ex. 0.31 0.11 0.12)

Output between each pass.

The sorted list of integers. (Ex. 0.11 0.12 0.31)

Constraint:

$0 \leq x < 2147483647$

x *is the value of element inside the array without considering the decimal point.*

For example. 231.32 *is regarded as* 23132

The modification made to the code in question 1 is the use of power of ten multiplication to convert the floating point number into a whole number, and then perform the radix sort as usual.

However, due to the java's limitation in precision (java's double/float are represented internally using 32-bit single-precision and 64-bit double-precision format IEEE 754 values (Chapter 2. The Structure of the Java Virtual Machine, n.d.)), the arrays may lose precision during the sorting process. The digits of each element are particularly important for radix sort, and the loss of precision will affect the bucket assignment during the passes. To solve this issue, we use BigDecimal to maintain the precision of the elements in the array.

Sample output

(See Appendix: Question 2)

Question 3

General Time Complexity Analysis

For complexity analysis, we will plot 2 graphs each for both algorithms in Q1 and Q2. From the algorithm, we observe that the number of primitive operations is mainly affected by:

i) **The highest number of digits of elements in the array, d** (Ex. 245 has 3 digits, 231.32 has 5 digits)

The number of digits influence the number of passes, since radix sort essentially performs a bucket sort for each digit of the element. If the highest number of digits in the array is 6, then 6 passes will be done to sort the array from the rightmost (least significant) to the leftmost digit (most significant).

ii) **The number of elements in the array, n**

The number of elements affects the number of primitive operations that needs to be done to put all the elements into the bucket. If there are 100 elements, then the "put element into the bucket" operation will be repeated 100 times.

Due to this reason, one 2D graph is not sufficient to perform the complexity analysis on both of those algorithms in Q1 and Q2. Hence, we will plot two 2D graphs, with one of the variables, n, d, being kept constant in each graph.

For the first graph, we will plot the graph for the number of operations against the number of elements in the array, n. The number of digits, d is kept constant. Meanwhile for the second graph, we will plot the graph for the number of operations against the number of digits, d. The number of elements, n, in the array is kept constant.

The number of primitive functions is kept track in both programs by using the global opCounter variable. For each value of n or d, we repeat the calculation 5 times and plot the average number of primitive operations.

The outputs used for the analysis can be viewed in Appendix: Question 3.

Graph-based Time Complexity Analysis

Question 1 (Radix sort for integer)

Table 1: (number of primitive operations against n)

n	Number of primitive operations
0	0
10	2228
100	16364
1000	156764
10000	1560800
100000	15600752
200000	31200812
300000	46800800
400000	62400776
500000	78000752
600000	93600788
700000	109200740
800000	124800836
900000	140400812
1000000	156000824

Graph 1: (number of primitive operations against n)

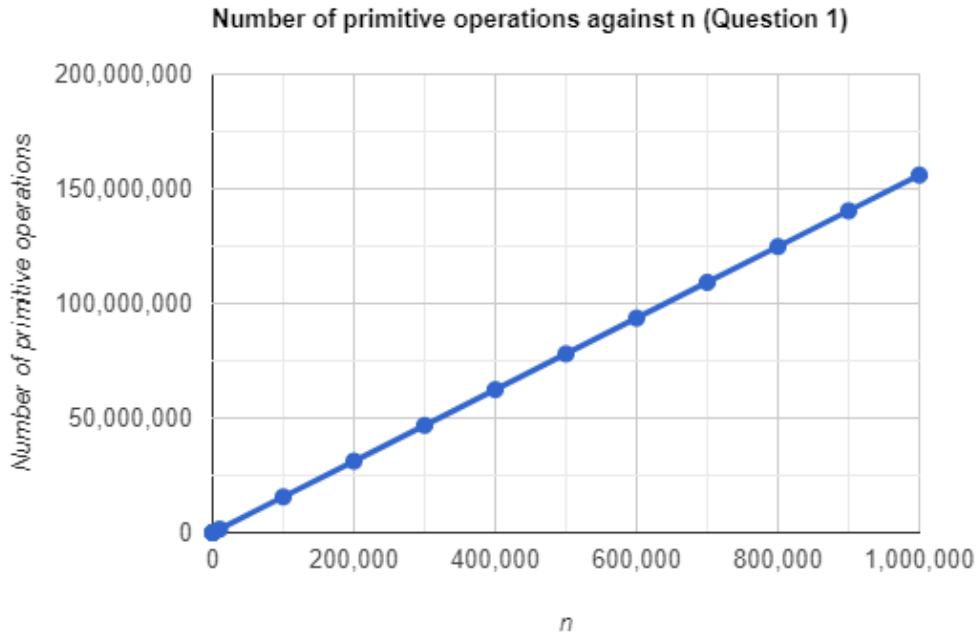
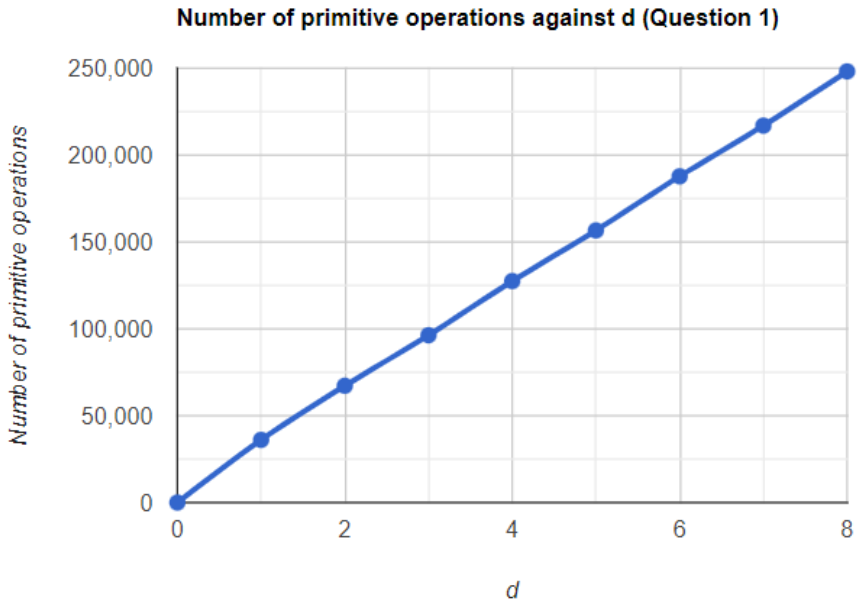


Table 2: (number of primitive operations against d)

d	Number of primitive operations
0	0
1	36128
2	67262
3	96404
4	127568
5	156752
6	187924
7	217108
8	248372

Graph 2: (number of primitive operations against d)



Question 2 (Radix sort for floating point number)

Table 1: (number of primitive operations against n)

n	Number of primitive operations
0	0
10	4334
100	36828
1000	361742
10000	3610732
100000	36100744
200000	72200746
300000	108300744
400000	144400738
500000	180500738
600000	216600748
700000	252700742
800000	288800752
900000	324900744
1000000	361000740

Graph 1: (number of primitive operations against n)

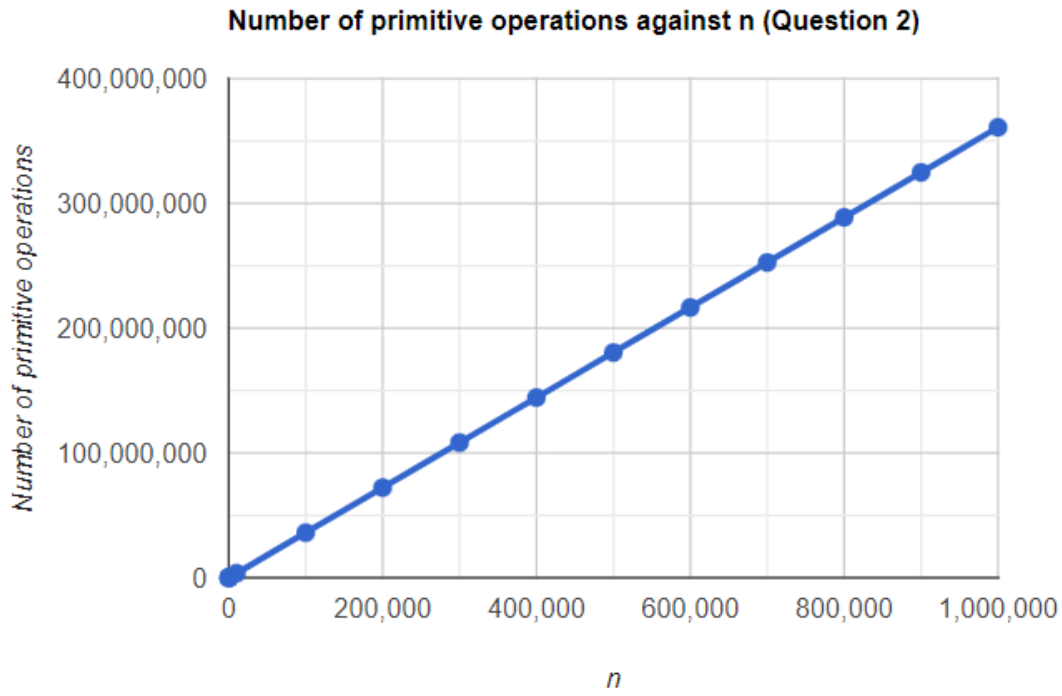
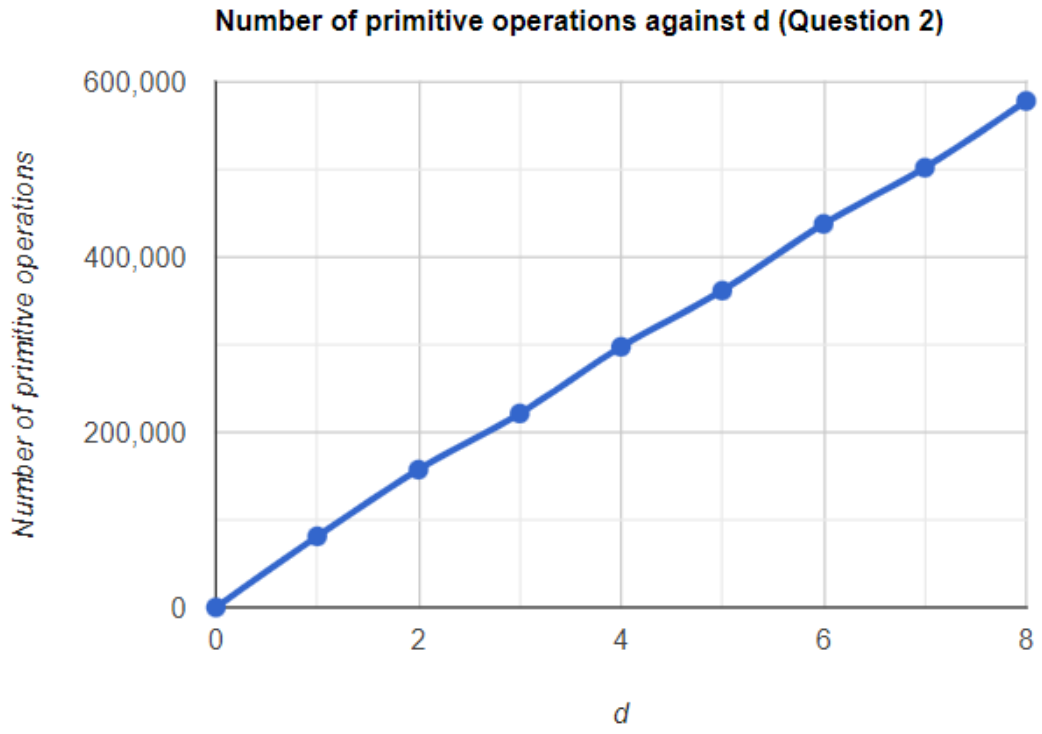


Table 2: (number of primitive operations against d)

d	Number of primitive operations
0	0
1	81134
2	157269
3	221422
4	297567
5	361732
6	437919
7	502114
8	578331

Graph 2: (number of primitive operations against d)



From the graphs of radix sort, for both integer and floating-point sorting, we can observe that the time complexity for radix sort depends on both the maximum number of digits in the array, and the number of elements inside the array. n is the number of elements in the input data, and d is the number of digits required to represent the maximum value.

We can infer from the code that in each pass, radix sort performs n operations to distribute the elements from the source array into the destination array, also known as buckets. The linear graph shown in graph 1 of both questions also supports our inference by indicating that the

number of primitive operations increases linearly with the number of elements in the initial array, n. Therefore, radix sort takes O(n) time per pass.

Also, in total, radix sort for integer performs d passes, one for each digit in the highest number of digits in the array. The linear graph shown in graph 2 of both questions proves that the number of primitive operations scales linearly with the highest number of digits in the initial array, d.

Therefore, the overall time complexity of radix sort for both integer and floating-point sorting is O(n) * O(d) = O(nd).

Despite taking more primitive operations to complete the sorting, the radix sort for floating-point sorting has the same time complexity as the radix sort for integer, which is O(nd), as seen by the linear natures of the graphs for both algorithms.

Best Case, Average Case and Worst Case Time Complexity

Since radix sort is a distribution sort, the time complexity of radix sort remains the same in all cases, best, average and worst. This is because no matter what the initial arrangement of array is, radix sort will still attempt to sort the array without skipping any of the procedures, which is by repeatedly passing the numbers into buckets from the rightmost (least significant) to the leftmost digit (most significant).

Radix sort's number of primitive operation is completely independent of the initial arrangement of the array (unlike insertion sort or quick sort) and is purely dependent on n and d only. This allows radix sort to be relatively consistent in terms of sorting time as compared to some other sorting methods.

Hence, here's the complete time complexity for the radix sort.

Case	Time complexity
Best-case	O(nd)
Average-case	O(nd)
Worst-case	O(nd)

And that concludes our analysis.

References

Baeldung. (2023, January 26). *Number of Digits in an Integer in Java*. Baeldung. https://www.baeldung.com/java-number-of-digits-in-int

Chapter 2. The Structure of the Java Virtual Machine. (n.d.). Docs.oracle.com. Retrieved May 12, 2023, from https://docs.oracle.com/javase/specs/jvms/se9/html/jvms-2.html

Moore, K. (n.d.). *Radix Sort | Brilliant Math & Science Wiki*. Brilliant.org. Retrieved May 6, 2023, from https://brilliant.org/wiki/radix-sort/#:~:text=Radix%20sort%20uses%20counting%20sort

Radix Sort. (2023, April 20). GeeksforGeeks. https://www.geeksforgeeks.org/radix-sort/

String (Java Platform SE 8). (n.d.). Docs.oracle.com. Retrieved May 6, 2023, from https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#format-java.lang.String-java.lang.Object...-

Appendix

Question 1

```
Enter the number of elements to be sorted: 10
Enter the maximum integer number: 50
Enter the minimum integer number: 0
Unsorted list: 37 1 43 21 6 37 47 14 37 46

(Pass 1)
Array 1:
 0->
 1->  1 21
 2->
 3-> 43
 4-> 14
 5->
 6->  6 46
 7-> 37 37 47 37
 8->
 9->

(Pass 2)
Array 2:
 0->  1  6
 1-> 14
 2-> 21
 3-> 37 37 37
 4-> 43 46 47
 5->
 6->
 7->
 8->
 9->
Sorted list: 1 6 14 21 37 37 37 43 46 47

Number of primitive operations: 914
```

Question 2

```
Enter the number of elements to be sorted: 10
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 1
Unsorted List: 11.3 11.6 99.0 20.1 58.4 81.3 98.5 28.8 81.3 62.7
(Pass 1)
Array 1:
 0->   99
 1-> 20.1
 2->
 3-> 11.3 81.3 81.3
 4-> 58.4
 5-> 98.5
 6-> 11.6
 7-> 62.7
 8-> 28.8
 9->

(Pass 2)
Array 2:
 0-> 20.1
 1-> 11.3 81.3 81.3 11.6
 2-> 62.7
 3->
 4->
 5->
 6->
 7->
 8-> 58.4 98.5 28.8
 9->   99

(Pass 3)
Array 1:
 0->
 1-> 11.3 11.6
 2-> 20.1 28.8
 3->
 4->
 5-> 58.4
 6-> 62.7
 7->
 8-> 81.3 81.3
 9-> 98.5   99
Sorted list: 11.3 11.6 20.1 28.8 58.4 62.7 81.3 81.3 98.5 99

Number of primitive operations: 2760
```

Question 3

Question 1

n is the variable, d is kept constant

n = 10

```
Enter the number of elements to be sorted: 10
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Unsorted list: 95092 31552 57999 85467 33318 30118 23817 11694 19192 78110
Sorted list: 11694 19192 23817 30118 31552 33318 57999 78110 85467 95092

Number of primitive operations: 2228
```

n = 100

```
Enter the number of elements to be sorted: 100
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 16364
```

n = 1000

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 156764
```

n = 10000

```
Enter the number of elements to be sorted: 10000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 1560800
```

n = 100000

```
Enter the number of elements to be sorted: 100000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 15600752
```

n = 200000

```
Enter the number of elements to be sorted: 200000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 31200812
```

n = 300000

```
Enter the number of elements to be sorted: 300000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 46800800
```

n = 400000

```
Enter the number of elements to be sorted: 400000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 62400776
```

n = 500000

```
Enter the number of elements to be sorted: 500000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 78000752
```

n = 600000

```
Enter the number of elements to be sorted: 600000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 93600788
```

n = 700000

```
Enter the number of elements to be sorted: 700000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 109200740
```

n = 800000

```
Enter the number of elements to be sorted: 800000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 124800836
```

n = 900000

```
Enter the number of elements to be sorted: 900000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 140400812
```

n = 1000000

```
Enter the number of elements to be sorted: 1000000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 156000824
```

n is kept constant, d is the variable

d = 1

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 9
Enter the minimum integer number: 0
Number of primitive operations: 36128
```

d = 2

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 99
Enter the minimum integer number: 0
Number of primitive operations: 67262
```

d = 3

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 999
Enter the minimum integer number: 0
Number of primitive operations: 96404
```

d = 4

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 9999
Enter the minimum integer number: 0
Number of primitive operations: 127568
```

d = 5

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 99999
Enter the minimum integer number: 0
Number of primitive operations: 156752
```

d = 6

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 999999
Enter the minimum integer number: 0
Number of primitive operations: 187924
```

d = 7

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 9999999
Enter the minimum integer number: 0
Number of primitive operations: 217108
```

d = 8

```
Enter the number of elements to be sorted: 1000
Enter the maximum integer number: 99999999
Enter the minimum integer number: 0
Number of primitive operations: 248372
```

Question 2

n is the variable, d is kept constant

n = 10

```
Enter the number of elements to be sorted: 10
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Unsorted List: 669.66 968.39 911.89 420.34 341.03 853.37 825.58 235.88 772.70 951.65
(Pass 1)
Array 1:
 0->  772.7
 1->
 2->
 3-> 341.03
 4-> 420.34
 5-> 951.65
 6-> 669.66
 7-> 853.37
 8-> 825.58 235.88
 9-> 968.39 911.89

(Pass 2)
Array 2:
 0-> 341.03
 1->
 2->
 3-> 420.34 853.37 968.39
 4->
 5-> 825.58
 6-> 951.65 669.66
 7->  772.7
 8-> 235.88 911.89
 9->

(Pass 3)
Array 1:
 0-> 420.34
 1-> 341.03 951.65 911.89
 2->  772.7
 3-> 853.37
 4->
 5-> 825.58 235.88
 6->
 7->
 8-> 968.39
 9-> 669.66
```

```
(Pass 4)
Array 2:
 0->
 1-> 911.89
 2-> 420.34 825.58
 3-> 235.88
 4-> 341.03
 5-> 951.65 853.37
 6-> 968.39 669.66
 7->  772.7
 8->
 9->

(Pass 5)
Array 1:
 0->
 1->
 2-> 235.88
 3-> 341.03
 4-> 420.34
 5->
 6-> 669.66
 7->  772.7
 8-> 825.58 853.37
 9-> 911.89 951.65 968.39
Sorted list: 235.88 341.03 420.34 669.66 772.7 825.58 853.37 911.89 951.65 968.39

Number of primitive operations: 4334
```

n = 100

```
Enter the number of elements to be sorted: 100
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 36828
```

n = 1000

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 361742
```

n = 10000

```
Enter the number of elements to be sorted: 10000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 3610732
```

n = 100000

```
Enter the number of elements to be sorted: 100000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 36100744
```

n = 200000

```
Enter the number of elements to be sorted: 200000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 72200746
```

n = 300000

```
Enter the number of elements to be sorted: 300000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 108300744
```

n = 400000

```
Enter the number of elements to be sorted: 400000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 144400738
```

n = 500000

```
Enter the number of elements to be sorted: 500000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 180500738
```

n = 600000

```
Enter the number of elements to be sorted: 600000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 216600748
```

n = 700000

```
Enter the number of elements to be sorted: 700000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 252700742
```

n = 800000

```
Enter the number of elements to be sorted: 800000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 288800752
```

n = 900000

```
Enter the number of elements to be sorted: 900000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 324900744
```

n = 1000000

```
Enter the number of elements to be sorted: 1000000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 361000740
```

n is kept constant, d is the variable

d = 1

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 1
Enter the number of digit(s) in the decimal part: 0
Number of primitive operations: 81134
```

d = 2

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 1
Enter the number of digit(s) in the decimal part: 1
Number of primitive operations: 157269
```

d = 3

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 1
Number of primitive operations: 221422
```

d = 4

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 2
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 297567
```

d = 5

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 2
Number of primitive operations: 361732
```

d = 6

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 3
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 437919
```

d = 7

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 4
Enter the number of digit(s) in the decimal part: 3
Number of primitive operations: 502114
```

d = 8

```
Enter the number of elements to be sorted: 1000
Enter the number of digit(s) in the integer part: 4
Enter the number of digit(s) in the decimal part: 4
Number of primitive operations: 578331
```