FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 1

“Experimental time complexity analysis ”

Performed by

Cheng-Yuan and Ma

Academic group

J4133c

Petr Chunaev *St.*

Petersburg 2022

Language : python

Goal : Experimental study of the time complexity of different algorithms

Problem : Generate an n-dimensional random vector $\boldsymbol{v} = [v, v, \dots , v]$ with non-negative elements. For $\boldsymbol{v}$, implement the following calculations and algorithms:

I.

Theory :

1. $f(\boldsymbol{v}) = const$ *(constant function);*
   Theory : when we create matrix , it cost O(Row * Column) complexity

2. $f(\boldsymbol{v}) = \sum v$ (the sum of elements);
Theory : when we create matrix , it cost O(Row * Columns) complexity

and it cost anohter Row * Columns to make calculation of sum

complexity will be 2(Row * Columns) = O(Row * Columns)

3. $f(\boldsymbol{v}) = \prod v$ (the product of elements)
theory : when we create matrix , it cost O(Row * Columns) complexity

and it cost anotheser Row * Columns to make calculation of product

complexity will be 2(Row * Columns) = O(Row * Columns

4. supposing that the elements of $\boldsymbol{v}$ are the coefficients of a polynomial P of

degree $n - 1$, calculate the value $P(1.5)$ by a direct calculation of $P(x) = \sum v\ x^{\wedge(k-1)}$(i.e. evaluating each term one by one) and by Horner's method

Theory : To understand the method, let us consider the example of $2x3 – 6x2 + 2x – 1$. The polynomial can be evaluated as $((2x – 6)x + 2)x – 1$. The idea is to initialize result as coefficient of xn which is 2 in this case, repeatedly multiply result with x and add next coefficient to result. Finally return result.

5. Bubble Sort of the elements of ***v***

Theory : Bubble sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not efficient for large data sets due to its average and worst-case time complexity is quite high.

Steps :

Run a nested for loop to traverse the input array using two variables **i** and **j**, such that $0 \le i < n-1$ and $0 \le j < n-i-1$

If arr[j] is greater than arr[j+1] then swap these adjacent elements, else move on

Print the sorted array

Time Complexity: O(N^2)

Auxiliary Space**:** O(1)

6. Quick Sort of the elements of ***v***;

Theory : quick sort is one of sorting algorithms as well

Step :

The idea of Quick Sort is to first find a reference point

Then send two agents to search from both sides of the data to the middle. If a value is found on the right that is smaller than the reference point, and a value is found on the left that is smaller than the reference point big, let them swap.

Repeatedly find and exchange until the two meet. Then swap the meeting point with the reference point. The first round is over.

Time Complexity: average : O(nLogn) , best : O(nLogn) , worst : O(n^2)

Auxiliary Space**:** O(x)

7. Timsort of the elements of ***v***.

Theory : Timsort is derived from merge sort and insertion sort , designed to perform well on many kinds of real-world data

Step :

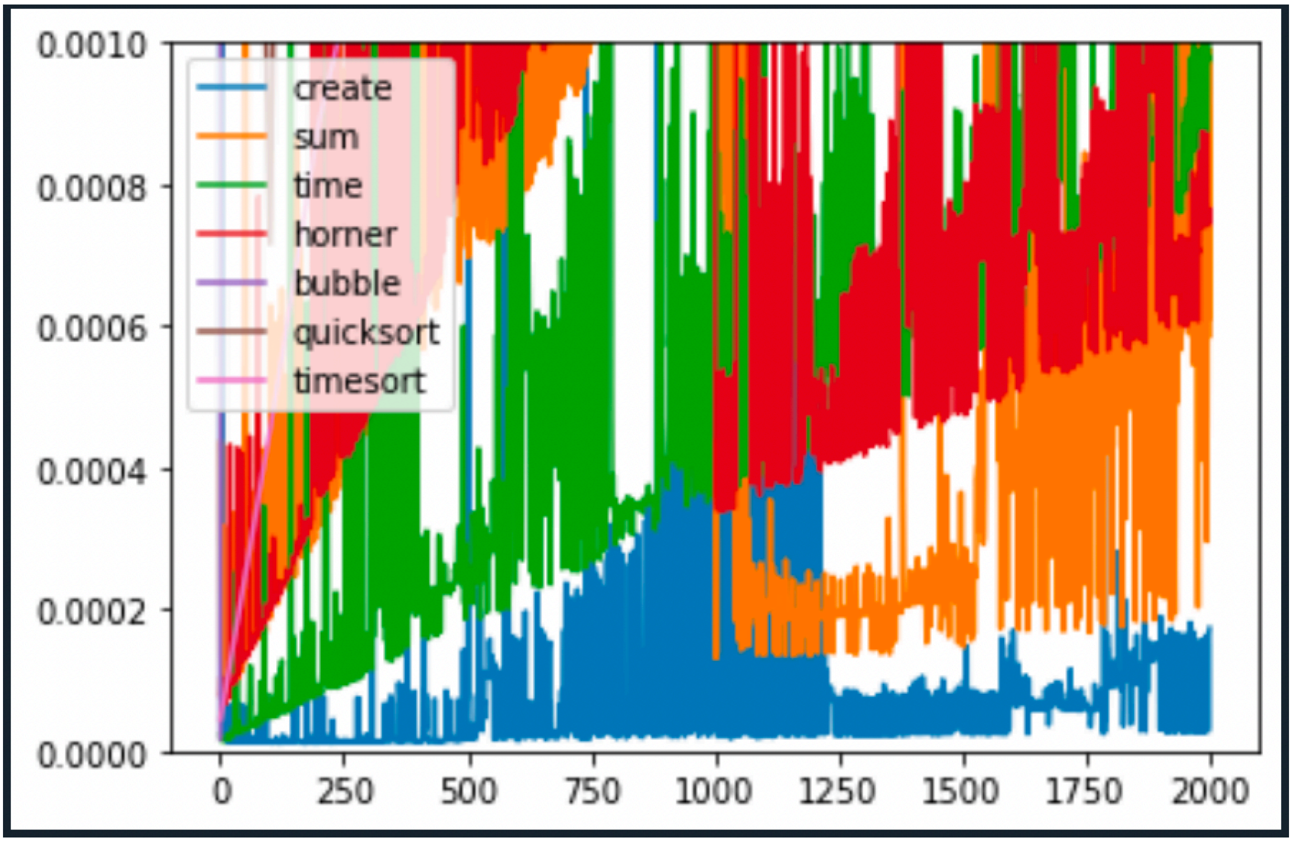
We divide the Array into chunks called Run . We sort these runs one by one using insertion sort, and then combine the runs using the combine function used in merge sort. Only use Insertion Sort to sort the Array if its size is less than run.

Depending on the size of the array, the run size may vary from 32 to 64. Note that the merge function performs well when the large and small subarrays are powers of 2. The idea is based on the fact that insertion sort performs well for small arrays.

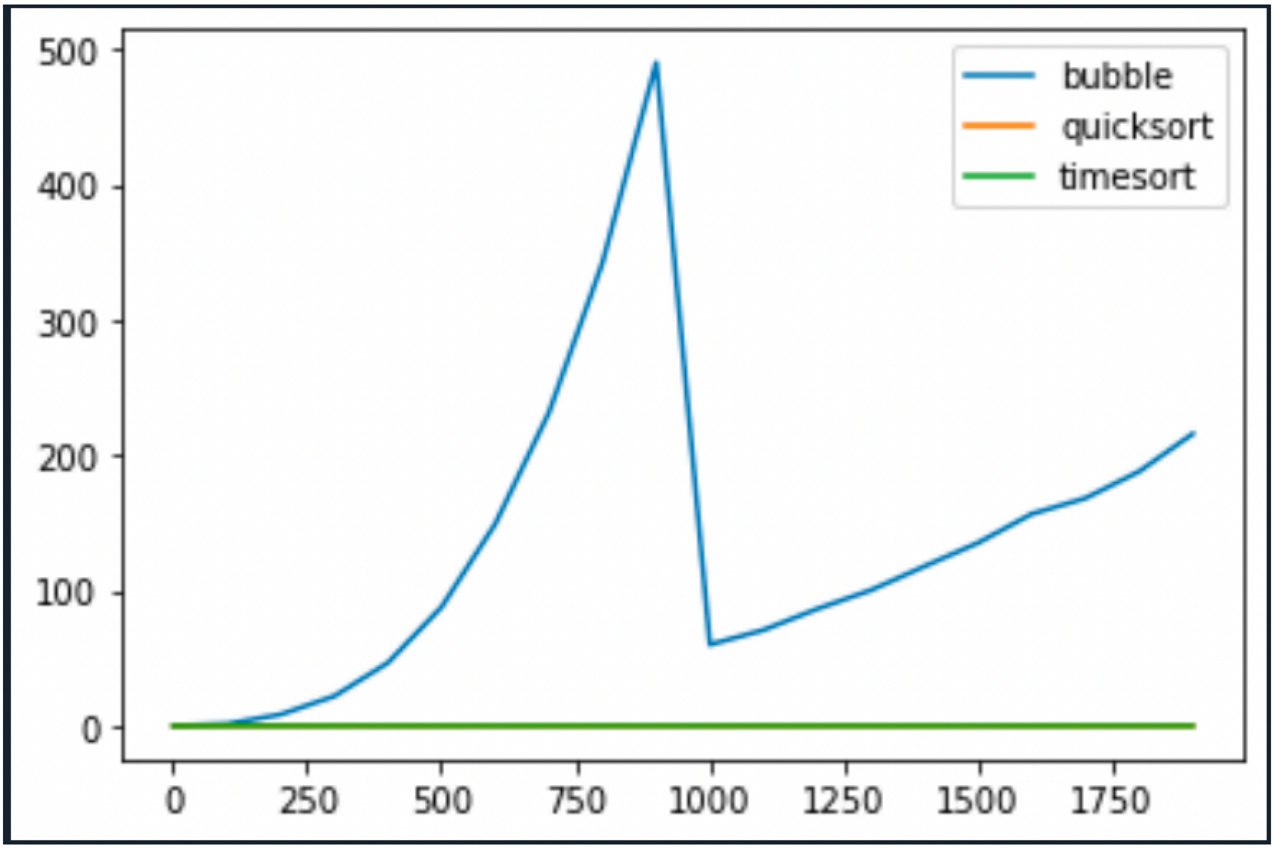
Time Complexity: O(nLogn)

Auxiliary Space**:** O(x)

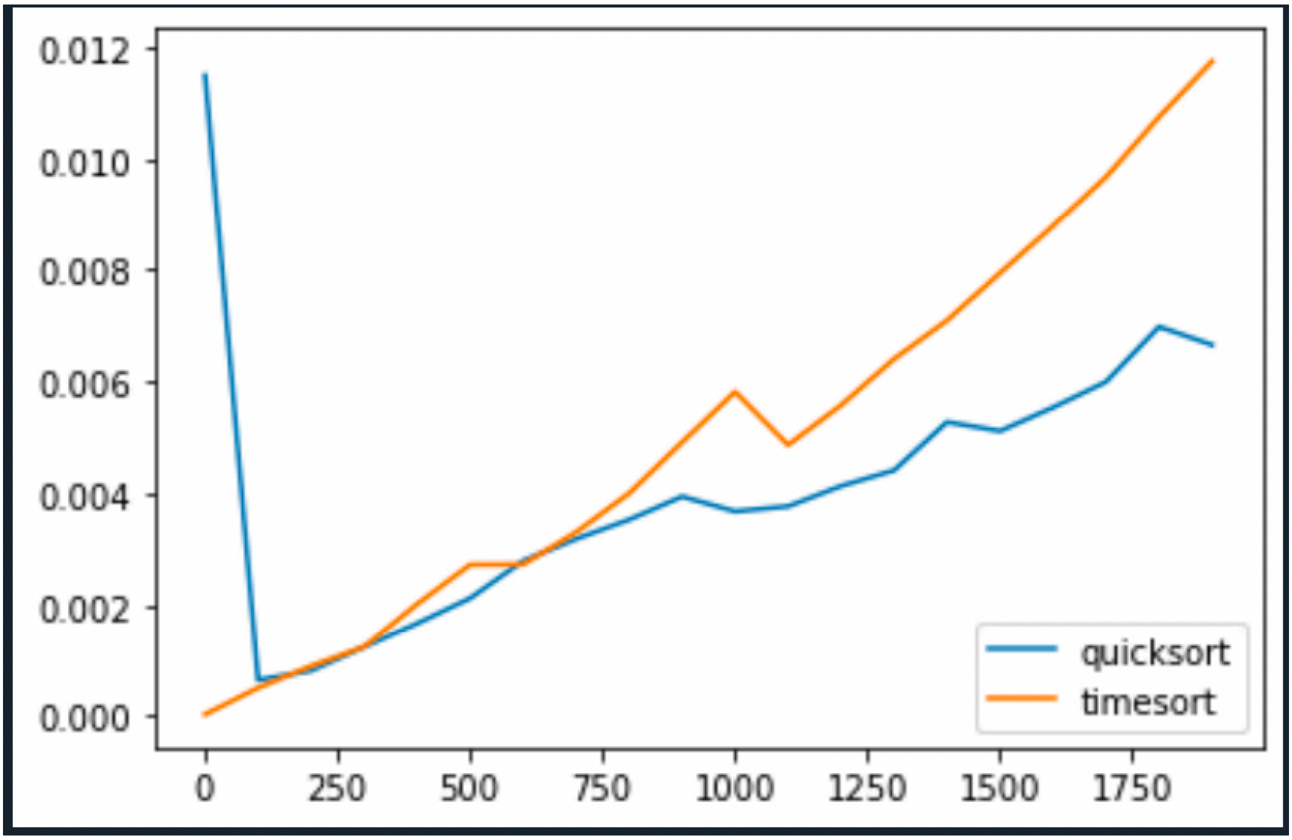
Result :



This is graph only with “bubble sort” , “quick sort ” and “time sort ”



This is only with “quick sort ” and “time sort ”



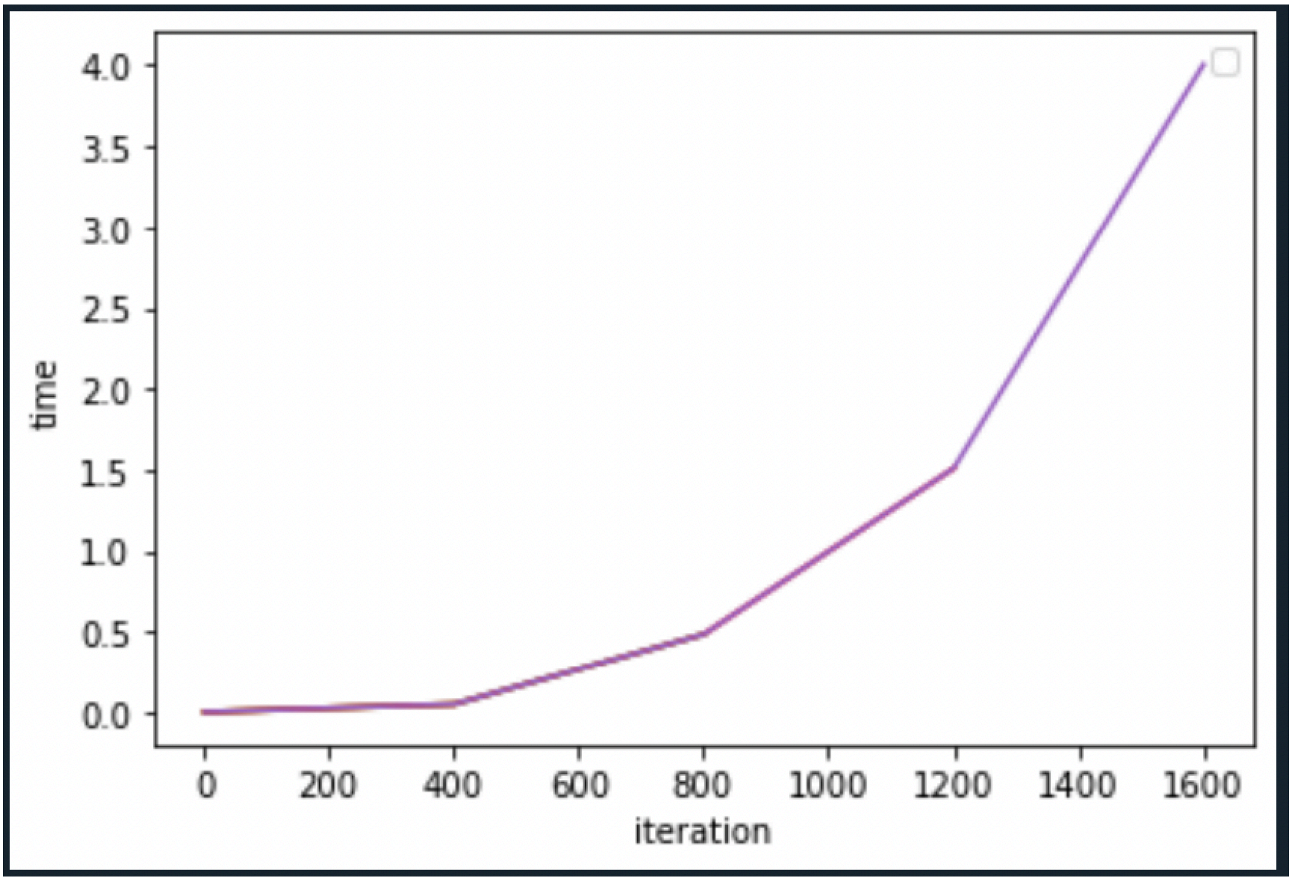
II. Generate random matrices A and B of size $n \times n$ with non-negative elements. Find the usual matrix product for A and B.

Theory :

Here I implemented numpy to use np.dot to do multiplication

Usually without implementing any API , the naive matrix multiplication and the Solvay Strassen algorithm are two most common way . There exist algorithms that achieve better complexity than the naive $O(n^3)$. Strassen algorithm achieves a complexity of $O(n^{2.807})$.

Result



III. Describe the data structures and design techniques used within the algorithms.

I used python to complete this graph by applying numpy which is considered one of most useful implementations . I used numpy to create two matrix withe the same size . Arrays from the NumPy library, called N-dimensioanl arrays or ndarray , are used as the primary data structure for representing data.

Due to property of ndarray , I can make multiplication with mathematical way . Therefore I use numpy.dot to make calculation to get product of two matrix

conclusion:

In the first section , through graphs that I created , We can see that when elements are getting large , “creating element” cost least time , and then it is “sum of element” , and then it is “product of elements” , “Horner method” , they take similar time to complete calculation . Sort function cost much more than first 4 functions , comparing only among sort algorithm , we can see that “bubble sort” cost much more ,and then it is “time sort” and “quick sort” .

In the second section , I used numpy instead of common algorithms such as naive matrix multiplication and the Solvay Strassen algorithm , it show polynomial execute time .which can be tested with theoretical complexity .

Appendix

https://github.com/MaChengYuan/task1.git