浙江工业大学

# BTH004
# Assignment2
# Experiment Report

（原创）

| | |
|---|---|
| Student ID | 201619630101 |
| Class | 1601 |
| Name | Chen  Yihang |
| Teacher | Cheng Zhenbo, John |
| Date | 2018/11/2 |

## 1.Problem description

Implementing merge sort and bubble sort. Analyse the running time of the two algorithms and compare the two algorithms.

## 2.Algorithm analysis

### 2.1 Merge sort:

Step 1: Divide the entire list to half, keep dividing until the length of each sub list is 1. Set the left sub list as “S_AL”, the right sub list as “S_AR”

Step 2: Merge every two sub lists. Compare the element of the two sub lists, append the smaller one to the list “result”

### 2.2 Bubble sort:

Using a nested loop to compare each two adjacent elements, if the previous one is larger than the rear one, then exchange the position. Keep doing this operation the whole list is sorted. In the worst case the elements are sorted in a reverse order.

## 3.Source code

### 3.1 Merge sort

```
import random
import time
start = time.clock()

def merge_Sort(lists):
    if len(lists)<=1:
        return lists
    else:
        mid=len(lists)//2
        AL=lists[0:mid]
        AR=lists[mid:]
        S_AL=merge_Sort(AL)
        S_AR=merge_Sort(AR)
        return merge(S_AL,S_AR)

def merge(S_AL,S_AR):
    i,j=0,0
    result=[]
    while i<len(S_AL) and j<len(S_AR):
        if S_AL[i]<=S_AR[j]:
            result.append(S_AL[i])
            i+=1
        else:
            result.append(S_AR[j])
            j+=1
    result+=S_AL[i:]
    result+=S_AR[j:]
    return result

if __name__=="__main__" :
    A = [random.randint(0, 10) for _ in range(1000)]
    iniTime = time.clock()
    print(A)
    print(merge_Sort(A))
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime -
start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
    print('Total running time: %s seconds' % (end - start))
```

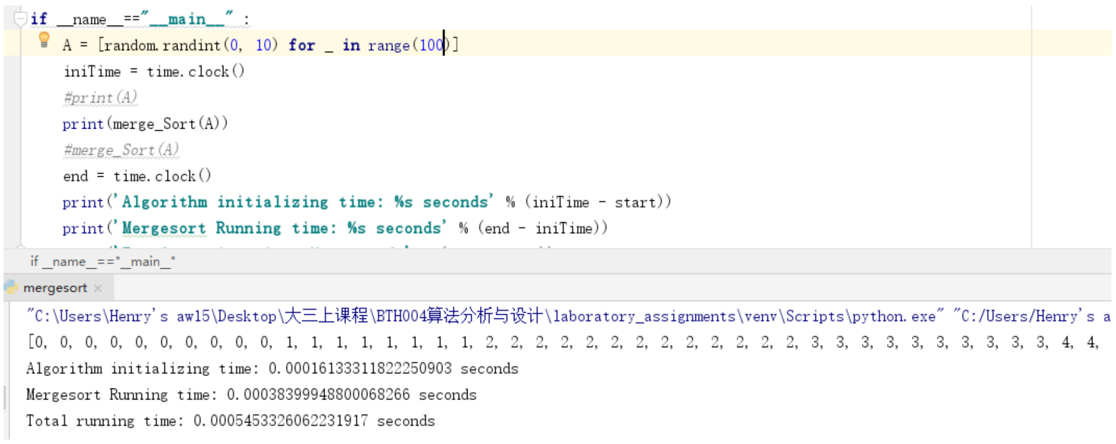
### 3.2 Bubble sort

```
import random
import time
start =time.clock()
def bubbleSort(bubbleList):
    n=len(bubbleList)
    while n>0:
        for i in range(n-1):
            if bubbleList[i] > bubbleList[i+1]:
                temp=bubbleList[i+1]
                bubbleList[i+1]=bubbleList[i]
                bubbleList[i]=temp
        n -= 1
    return bubbleList

if __name__ == '__main__':
    bubbleList = [random.randint(0, 10) for _ in range(1000)]
    iniTime = time.clock()
    print(bubbleList)
    print(bubbleSort(bubbleList))
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime -
start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
    print('Total running time: %s seconds' % (end - start))
```

## 4.Debugging

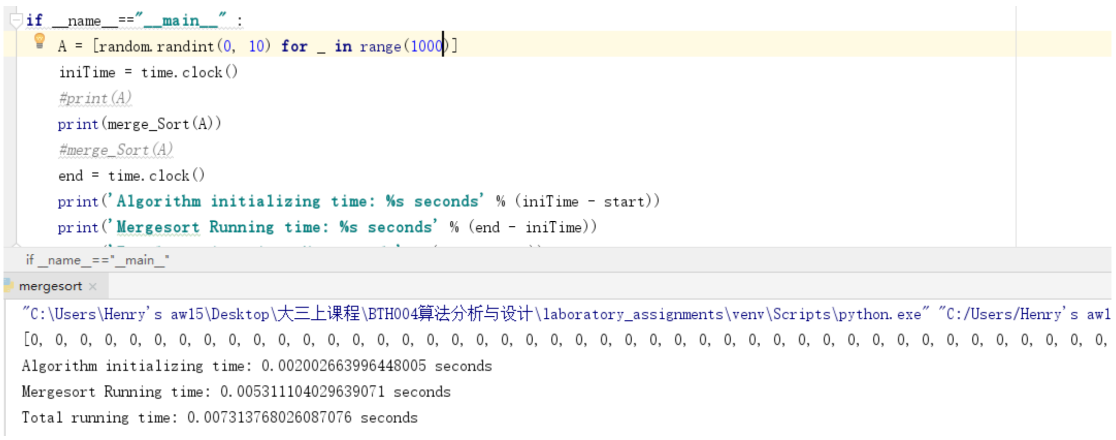
### 4.1.Debugging of Mergesort:



```
if __name__=="__main__" :
    A = [random.randint(0, 10) for _ in range(100)]
    iniTime = time.clock()
    #print(A)
    print(merge_Sort(A))
    #merge_Sort(A)
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime - start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
```

```
"C:\Users\Henry's aw15\Desktop\大三上课程\BTH004算法分析与设计\laboratory_assignments\venv\Scripts\python.exe" "C:/Users/Henry's a
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4,
Algorithm initializing time: 0.00016133311822250903 seconds
Mergesort Running time: 0.00038399948800068266 seconds
Total running time: 0.0005453326062231917 seconds
```

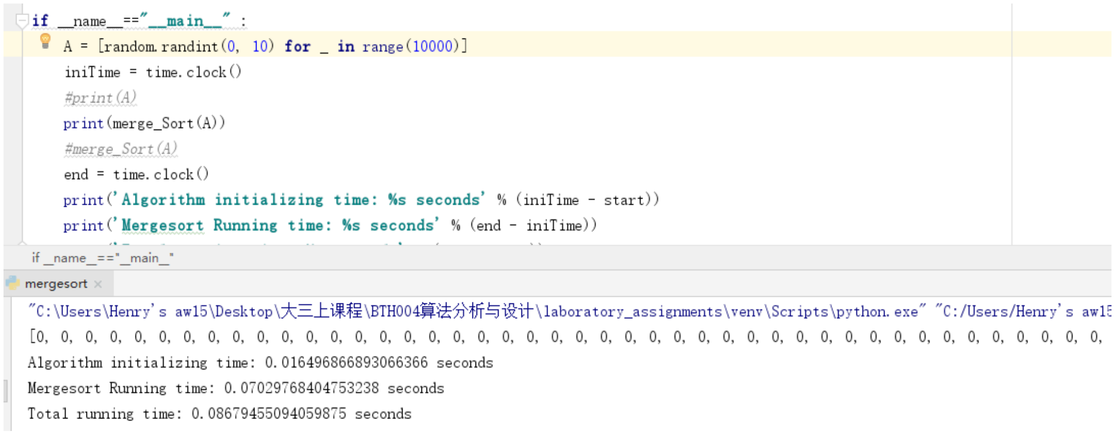
**(100 random numbers in the list)**



```
if __name__=="__main__":
    A = [random.randint(0, 10) for _ in range(1000)]
    iniTime = time.clock()
    #print(A)
    print(merge_Sort(A))
    #merge_Sort(A)
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime - start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
```

```
if _name_=="_main_"
mergesort
"C:\Users\Henry's aw15\Desktop\大三上课程\BTH004算法分析与设计\laboratory_assignments\venv\Scripts\python.exe" "C:/Users/Henry's aw1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Algorithm initializing time: 0.002002663996448005 seconds
Mergesort Running time: 0.005311104029639071 seconds
Total running time: 0.007313768026087076 seconds
```

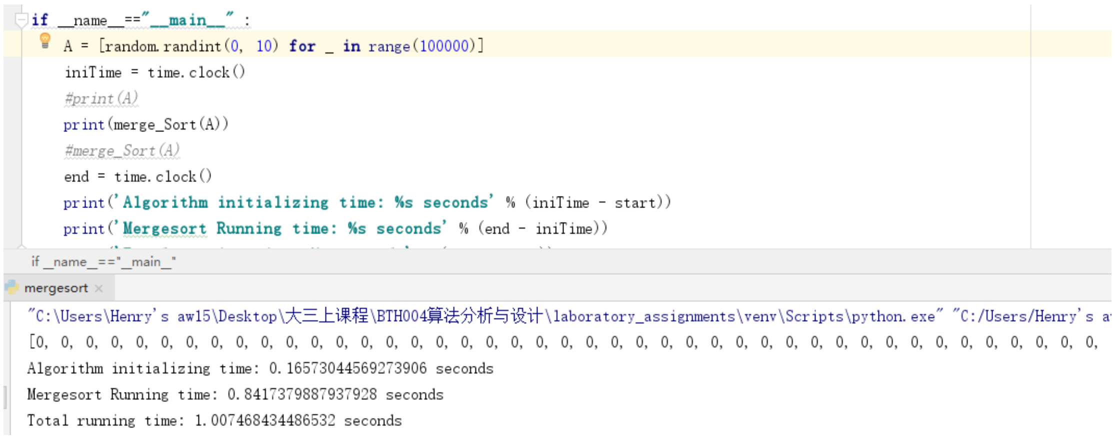
**(1000 random numbers in the list)**



```
if __name__=="__main__":
    A = [random.randint(0, 10) for _ in range(10000)]
    iniTime = time.clock()
    #print(A)
    print(merge_Sort(A))
    #merge_Sort(A)
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime - start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
```

```
if _name_=="_main_"
mergesort
"C:\Users\Henry's aw15\Desktop\大三上课程\BTH004算法分析与设计\laboratory_assignments\venv\Scripts\python.exe" "C:/Users/Henry's aw15
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Algorithm initializing time: 0.016496866893066366 seconds
Mergesort Running time: 0.07029768404753238 seconds
Total running time: 0.08679455094059875 seconds
```

**(10000 random numbers in the list)**

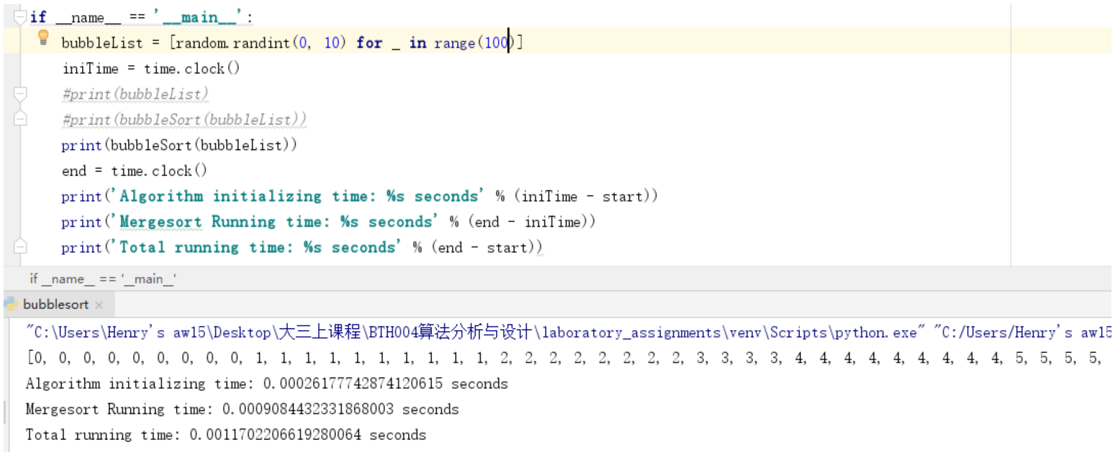


```
if __name__=="__main__":
    A = [random.randint(0, 10) for _ in range(100000)]
    iniTime = time.clock()
    #print(A)
    print(merge_Sort(A))
    #merge_Sort(A)
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime - start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
```

```
if _name_=="_main_"
mergesort
"C:\Users\Henry's aw15\Desktop\大三上课程\BTH004算法分析与设计\laboratory_assignments\venv\Scripts\python.exe" "C:/Users/Henry's a
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Algorithm initializing time: 0.16573044569273906 seconds
Mergesort Running time: 0.8417379887937928 seconds
Total running time: 1.007468434486532 seconds
```

**(100000 random numbers in the list)**

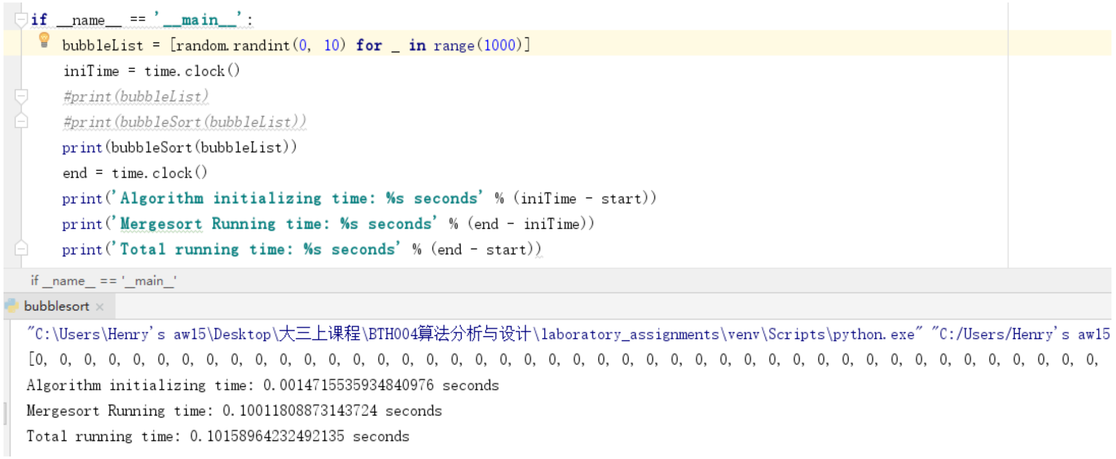
## 4.2 Debugging of Bubblesort



```
if __name__ == '__main__':
    bubbleList = [random.randint(0, 10) for _ in range(100)]
    iniTime = time.clock()
    #print(bubbleList)
    #print(bubbleSort(bubbleList))
    print(bubbleSort(bubbleList))
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime - start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
    print('Total running time: %s seconds' % (end - start))
```

```
"C:\Users\Henry's aw15\Desktop\大三上课程\BTH004算法分析与设计\laboratory_assignments\venv\Scripts\python.exe" "C:/Users/Henry's aw15
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5,
Algorithm initializing time: 0.00026177742874120615 seconds
Mergesort Running time: 0.0009084432331868003 seconds
Total running time: 0.0011702206619280064 seconds
```

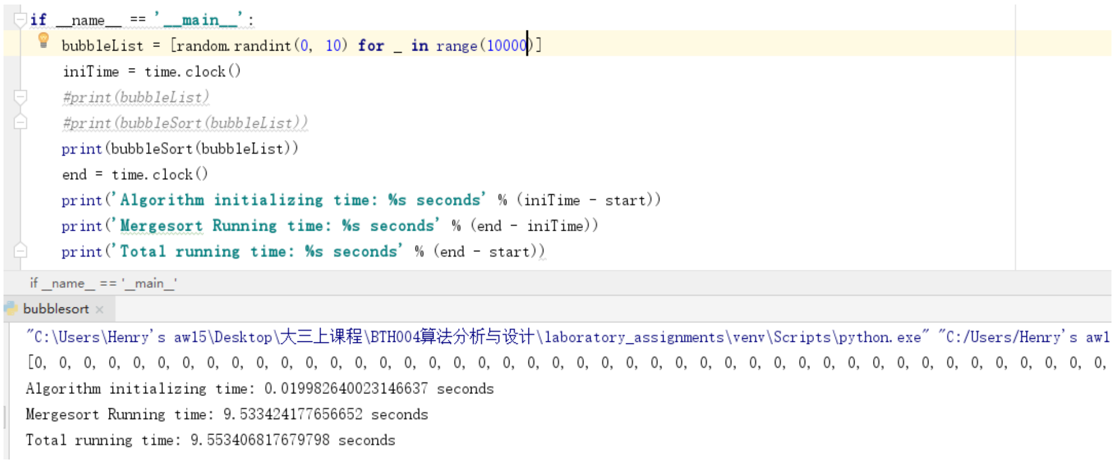
(100 random numbers in the list)



```
if __name__ == '__main__':
    bubbleList = [random.randint(0, 10) for _ in range(1000)]
    iniTime = time.clock()
    #print(bubbleList)
    #print(bubbleSort(bubbleList))
    print(bubbleSort(bubbleList))
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime - start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
    print('Total running time: %s seconds' % (end - start))
```

```
"C:\Users\Henry's aw15\Desktop\大三上课程\BTH004算法分析与设计\laboratory_assignments\venv\Scripts\python.exe" "C:/Users/Henry's aw15
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Algorithm initializing time: 0.0014715535934840976 seconds
Mergesort Running time: 0.10011808873143724 seconds
Total running time: 0.10158964232492135 seconds
```

(1000 random numbers in the list)



```
if __name__ == '__main__':
    bubbleList = [random.randint(0, 10) for _ in range(10000)]
    iniTime = time.clock()
    #print(bubbleList)
    #print(bubbleSort(bubbleList))
    print(bubbleSort(bubbleList))
    end = time.clock()
    print('Algorithm initializing time: %s seconds' % (iniTime - start))
    print('Mergesort Running time: %s seconds' % (end - iniTime))
    print('Total running time: %s seconds' % (end - start))
```

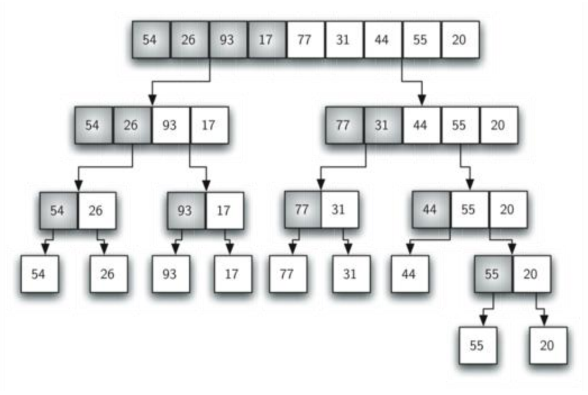
```
"C:\Users\Henry's aw15\Desktop\大三上课程\BTH004算法分析与设计\laboratory_assignments\venv\Scripts\python.exe" "C:/Users/Henry's aw1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Algorithm initializing time: 0.019982640023146637 seconds
Mergesort Running time: 9.533424177656652 seconds
Total running time: 9.553406817679798 seconds
```

(10000 random numbers in the list)

## 5. Theoretical analysis:

5.1 Merge sort:

First, the list is split into halves. We can divide a list in half ***logn*** times where n is the length of the list. The second process is the merge. Each item in the list will eventually be processed and placed on the sorted list. So the merge operation which results in a list of size n requires n operations. The result of this analysis is that ***logn*** splits, each of which costs n for a total of ***nlogn*** operations. A merge sort is an O(***nlogn***) algorithm.



5.2 Bubble sort:

In the first pass the shaded items are being compared to see if they are out of order. If there are n items in the list, then there are n–1 pairs of items that need to be compared on the first pass. It is important to note that once the largest value in the list is part of a pair, it will continually be moved along until the pass is complete.

At the start of the second pass, the largest value is now in place. There are n–1 items left to sort, meaning that there will be n–2 pairs. Since each pass places the next largest value in place, the total number of passes necessary will be n–1. After completing the n–1 passes, the smallest item must be in the correct position with no further processing required.

To analyze the bubble sort, we should note that regardless of how the items are arranged in the initial list, n–1 passes will be made to sort a list of size n. The total number of comparisons is the sum of the first n–1 integers. Recall that the sum of the first n integers is 1/2*n^2+1/2*n. The sum of the first n–1 integers is 1/2*n^2+1/2*n–n, which is 1/2*n^2–1/2*n. This is still O(***n2***) comparisons.

# 6. Practical analysis

## 6.1 Practical analysis:

### Merge sort:

| test | 1 | 2 | 3 | 4 | average |
|---|---|---|---|---|---|
| **Merge sort 10^1** | 0.048 | 0.047 | 0.053 | 0.050 | 0.050 |
| **Merge sort 10^2** | 0.557 | 0.521 | 0.385 | 0.365 | 0.457 |
| **Merge sort 10^3** | 5.029 | 5.006 | 5.523 | 5.248 | 5.202 |
| **Merge sort 10^4** | 71.110 | 70.129 | 85.474 | 78.033 | 76.187 |
| **Merge sort 10^5** | 861.612 | 790.742 | 801.826 | 814.862 | 817.261 |

Standard unit: ms

### Bubble sort:

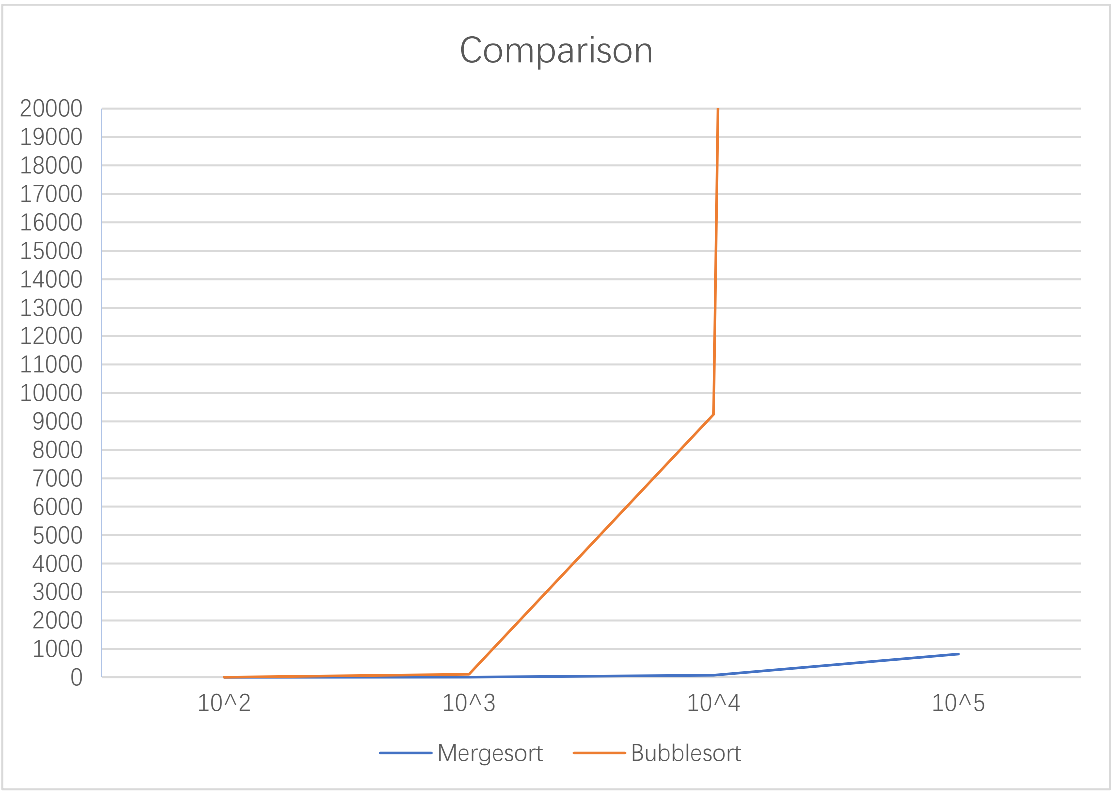
| test | 1 | 2 | 3 | 4 | average |
|---|---|---|---|---|---|
| **Bubblesort 10^1** | 0.043 | 0.036 | 0.036 | 0.041 | 0.039 |
| **Bubblesort 10^2** | 0.856 | 0.852 | 0.871 | 0.891 | 0.868 |
| **Bubblesort 10^3** | 95.388 | 112.002 | 90.372 | 95.015 | 98.194 |
| **Bubblesort 10^4** | 10223.491 | 9642.350 | 9922.532 | 9596.345 | 9164.430 |
| **Bubblesort 10^5** | …… | …… | …… | …… | …… |

Standard unit: ms

**Comparison of the two algorithms:**

| | 10^1 | 10^2 | 10^3 | 10^4 | 10^5 |
|---|---|---|---|---|---|
| **Merge sort** | 0.050 | 0.457 | 5.202 | 76.187 | 817.261 |
| **Bubble sort** | 0.039 | 0.868 | 98.194 | 9164.430 | …… |

Standard unit: ms



We can see that when the input number is higher than 10^3, there will bee a great difference between the running time of the two algorithms.

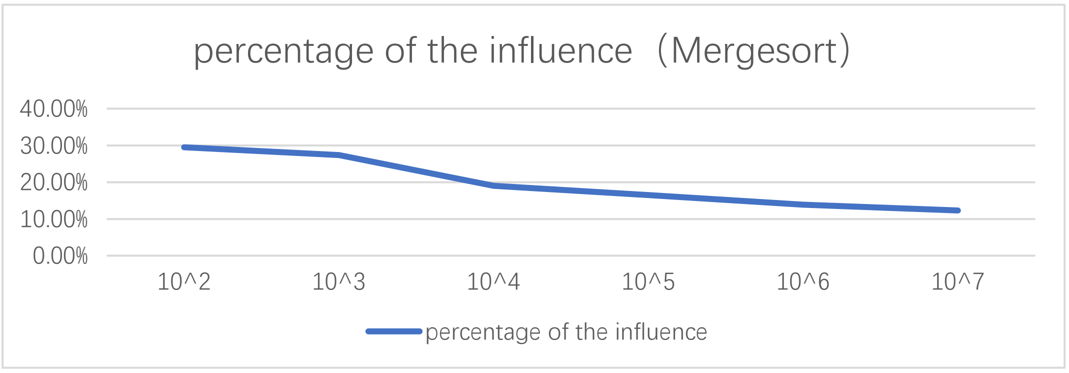
## 7. Influence of the initializing time:

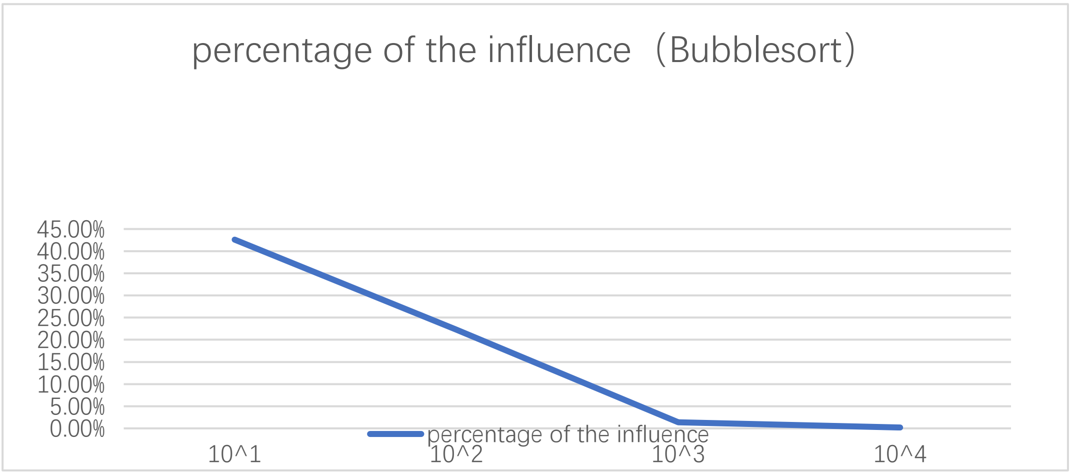
### Merge sort:

| test | 10^2 | 10^3 | 10^4 | 10^5 | 10^6 | 10^7 |
|---|---|---|---|---|---|---|
| Initializing time | 0.161 | 2.002 | 16.497 | 165.730 | 1597.424 | 15602.731 |
| Total running time | 0.545 | 7.313 | 86.795 | 1004.768 | 11395.293 | 127239.044 |
| Percentage of the influence | 29.5% | 27.4% | 19.0% | 16.5% | 13.9% | 12.3% |

### Bubble sort:

| test | 10^1 | 10^2 | 10^3 | 10^4 |
|---|---|---|---|---|
| Initializing time | 0.035 | 0.261 | 1.471 | 19.982 |
| Total running time | 0.082 | 1.170 | 101.589 | 9553.406 |
| Percentage of the influence | 42.6% | 22.3% | 1.4% | 0.2% |

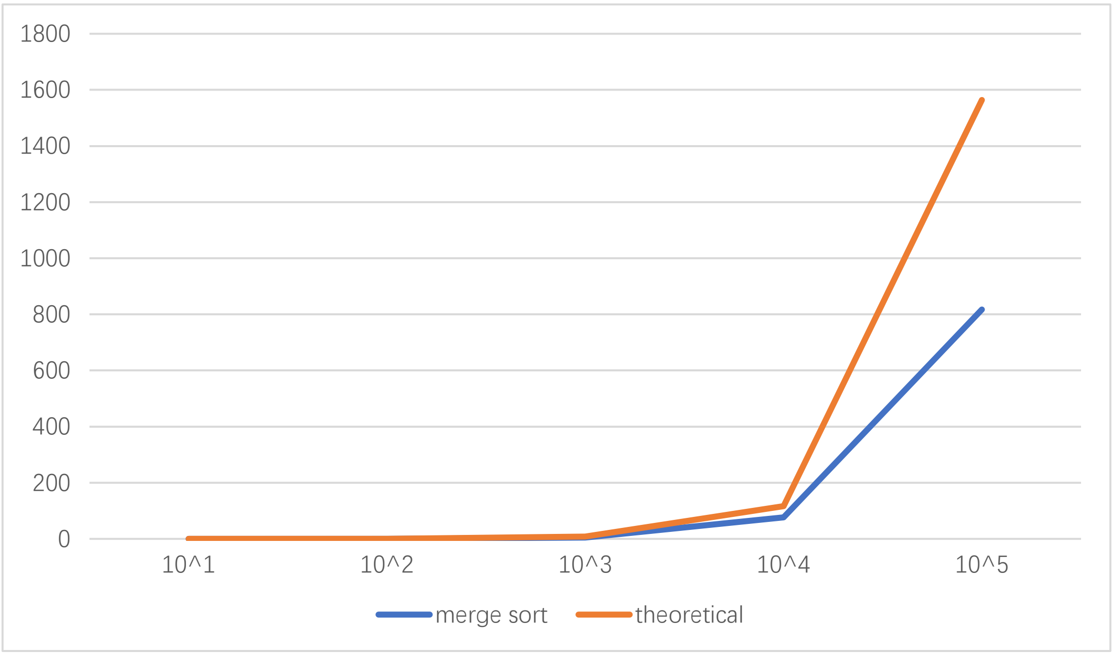




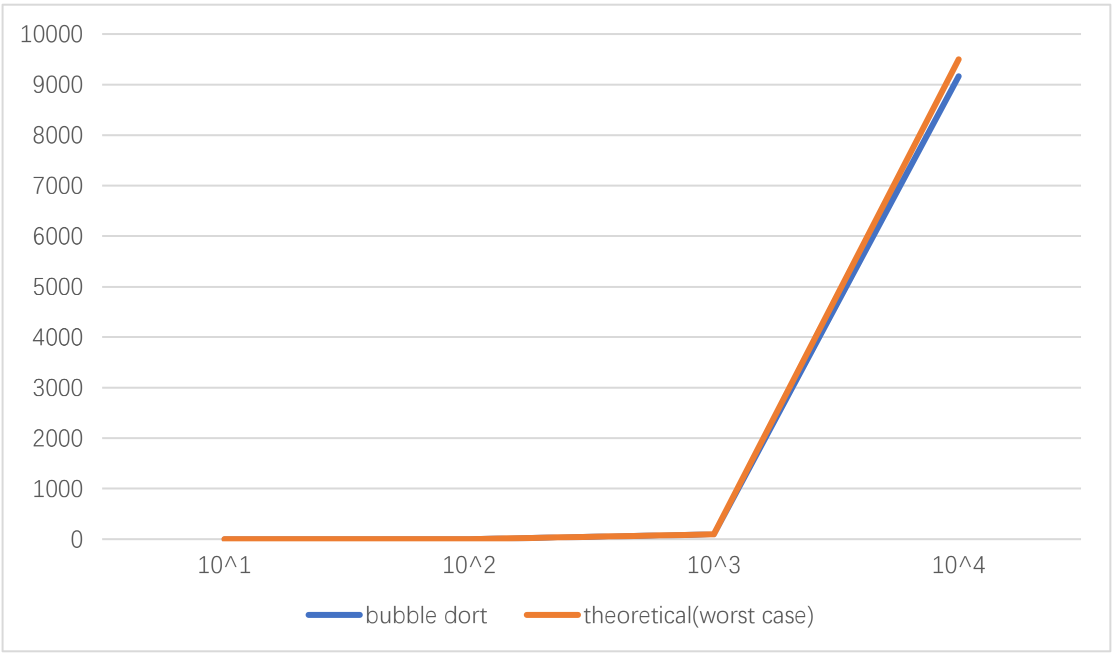
**Answer:** According to the results I consider that in bubble sort algorithm, if the input size is larger than **10^3**, we can neglect the influence of the initializing time. But in merge sort, even the input number is larger than 10^7 we still get a more than 10% of the influence of the initializing time, so I consider the time of initializing in merge sort could not be ignored.

## 8. comparison of my practical and theoretical analyses.

8.1 Comparison of merge sort:



8.2 Comparison of bubble sort:


10^1
10^2
10^3
10^4
bubble dort
theoretical(worst case)