HW1

20989977 Zhang Mingtao

2023/11/11

0.

```
library(reticulate)
```

1.

```
import csv
#
# #
# with open('C:/Users/mzhangdb/Desktop/data/TDCS_M06A_20190830_080000.csv', newline='') as csvfile:
#   reader = csv.reader(csvfile)
#   VT = []
#   DT = []
#   GID = []
#   for row in reader:
#     c1 = row[:1]
#     VT.append(c1)
#     c2 = row[:2]
#     DT.append(c2)
#     c3 = row[:3]
#     GID.append(c3)
#
#
# VT[:10]
# DT[:10]
# GID[:10]

import time
import timeit
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

def check_sort_result(func, data):
    start = time.time()
    data_copy = data[:] # important -- copy the list, instead of copy the reference.
    result = func(data_copy)
    time_used = time.time() - start
    for i in range(len(result)-1):
        if result[i] > result[i+1]:
            print("Check failed: func(data) is not sorted properly.")
            return
    print("The method {0} returned successfully for data size {1} with {2} seconds.".format(func.__name__.center(15), len(resul
```

```
t), time_used))

def check_performance(func, data, size_samples = [10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]):
    timing_array = []
    for size in size_samples:
        n_repeated = size_samples[-1] // size
        start = time.time()
        for run in range(n_repeated):
            data_truncated = data[:size]
            func(data_truncated)
        timing_array.append((time.time() - start) / n_repeated)
    ref_linear = [i * timing_array[0] / size_samples[0] for i in size_samples]
    ref_quadratic = [i**2 * timing_array[0] / size_samples[0]**2 for i in size_samples]
    plt.figure(figsize=(8,6),dpi=300)
    plt.loglog(size_samples, ref_linear, size_samples, ref_quadratic, size_samples, timing_array)
    plt.show()


traffic_data = pd.read_csv("C:\\Users\\张铭韬\\Desktop\\学业\\港科大\\MSDM5051面向对象python数据结构\\作业\\hw1\\data\\TDCS_M06
A_20190830_080000.csv",header=None)
traffic_data = traffic_data.iloc[:20000,]

# traffic_data.iloc[1,1]  #索引
# traffic_data.loc[1,1]   #标签

VT = traffic_data.iloc[:,0]
DT = traffic_data.iloc[:,1]
GID = traffic_data.iloc[:,2]

distances = [row[5] for row in traffic_data.values.tolist()][:10000]

import sys
module_path = 'C:\\Users\\张铭韬\\Desktop\\学业\\港科大\\MSDM5051面向对象python数据结构\\作业\\hw1'
sys.path.append(module_path)

from sorting import *
from tree_graph import *
from bintrees import *
```

```
from sortedcontainers import *

def AVL_sort(data):
    tree = AVLTree()

    for val in data:
        if val in tree:
            tree[val] += 1  # 增加计数器
        else:
            tree.insert(val, 1)

    sorted_data = []
    for val, count in tree.iter_items():  # 使用iter_items()迭代器按顺序获取节点值和计数器
        sorted_data.extend([val] * count)  # 根据计数器数量添加值到结果列表

    return sorted_data


def BST_sort(data):
  bst = SortedDict()

  for val in data:
    if val in bst:
      bst[val] += 1
    else:
      bst[val] = 1

  sorted_data = []
  for key, count in bst.items():
    sorted_data.extend([key] * count)

  return sorted_data
```

```
# column 0

check_sort_result(bubble_sort,VT.tolist())
```

```
## The method   bubble_sort   returned successfully for data size 20000 with 15.676480531692505 seconds.
```

```
check_sort_result(insertion_sort,VT.tolist())
```

```
## The method  insertion_sort returned successfully for data size 20000 with 9.150026559829712 seconds.
```

```
sys.setrecursionlimit(50000)
check_sort_result(quicksort,VT.tolist())
```

```
## The method    quicksort    returned successfully for data size 20000 with 6.516992807388306 seconds.
```

```
check_sort_result(heapsort,VT.tolist())
```

```
## The method     heapsort    returned successfully for data size 2 with 0.5833611488342285 seconds.
```

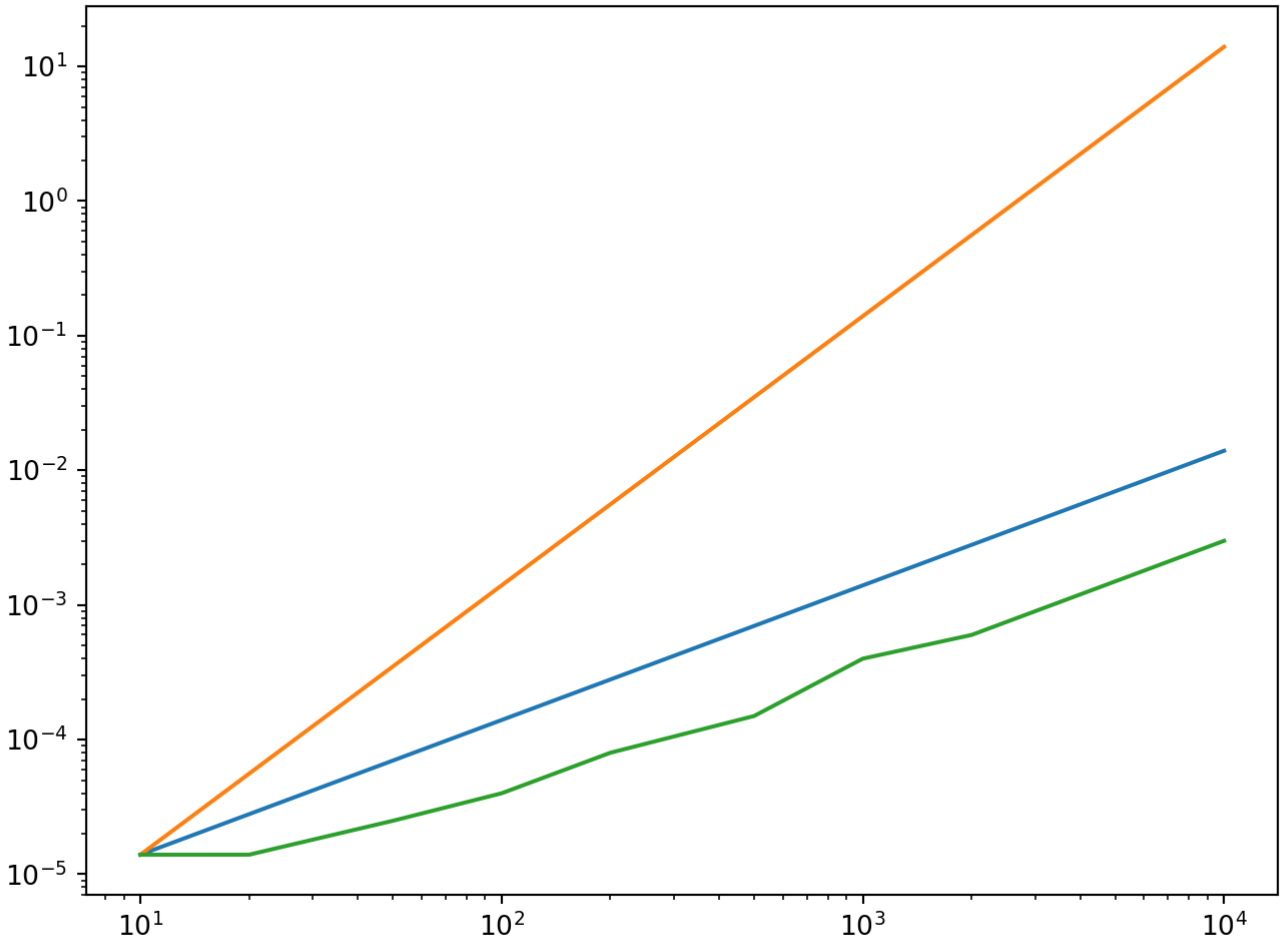
```
check_sort_result(mergesort,VT.tolist())
```

```
## The method    mergesort    returned successfully for data size 20000 with 0.047872304916381836 seconds.
```

```
check_sort_result(AVL_sort,VT.tolist())
# check_sort_result(BST_sort,VT.tolist())
```

```
## The method     AVL_sort    returned successfully for data size 20000 with 0.02190995216369629 seconds.
```

```
check_performance(BST_sort,VT.tolist())
```




```
# column 1

check_sort_result(bubble_sort,DT.tolist())
```

```
## The method   bubble_sort   returned successfully for data size 20000 with 25.23228883743286 seconds.
```

```
check_sort_result(insertion_sort,DT.tolist())
```

```
## The method  insertion_sort returned successfully for data size 20000 with 12.978593111038208 seconds.
```

```
sys.setrecursionlimit(50000)
check_sort_result(quicksort,DT.tolist())
# check_sort_result(heapsort,DT.tolist())
```

```
## The method    quicksort    returned successfully for data size 20000 with 0.04886960983276367 seconds.
```

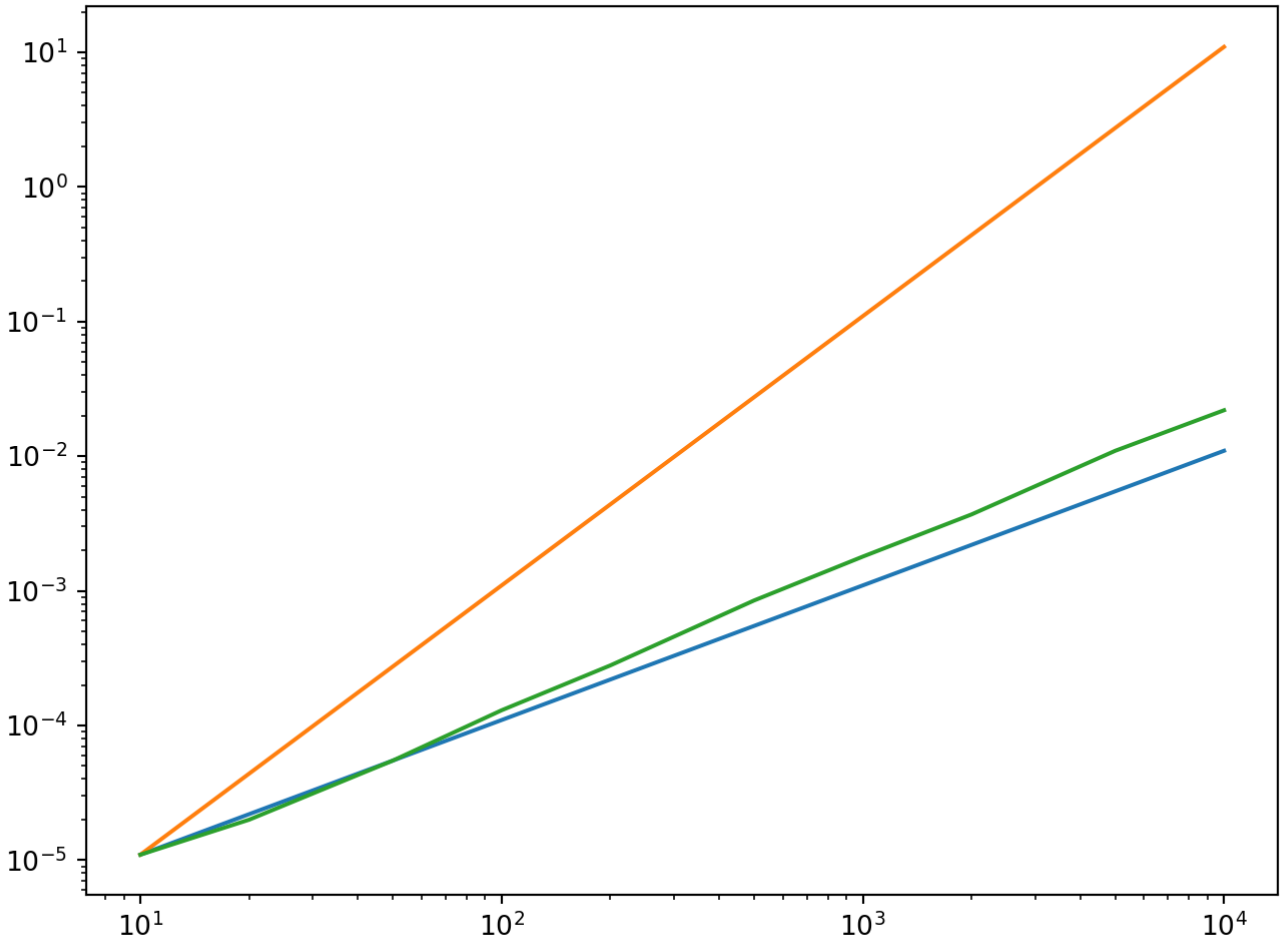
```
check_sort_result(mergesort,DT.tolist())
```

```
## The method    mergesort    returned successfully for data size 20000 with 0.06482577323913574 seconds.
```

```
check_sort_result(AVL_sort,DT.tolist())
# check_sort_result(BST_sort,DT.tolist())
```

```
## The method     AVL_sort    returned successfully for data size 20000 with 0.13223910331726074 seconds.
```

```
check_performance(quicksort,DT.tolist())
```




```
# column 2

check_sort_result(bubble_sort,GID.tolist())
```

```
## The method   bubble_sort   returned successfully for data size 20000 with 25.013323068618774 seconds.
```

```
check_sort_result(insertion_sort,GID.tolist())
```

```
## The method  insertion_sort returned successfully for data size 20000 with 11.687329053878784 seconds.
```

```
sys.setrecursionlimit(50000)
check_sort_result(quicksort,GID.tolist())
# check_sort_result(heapsort,GID.tolist())
```

```
## The method    quicksort    returned successfully for data size 20000 with 0.13665056228637695 seconds.
```

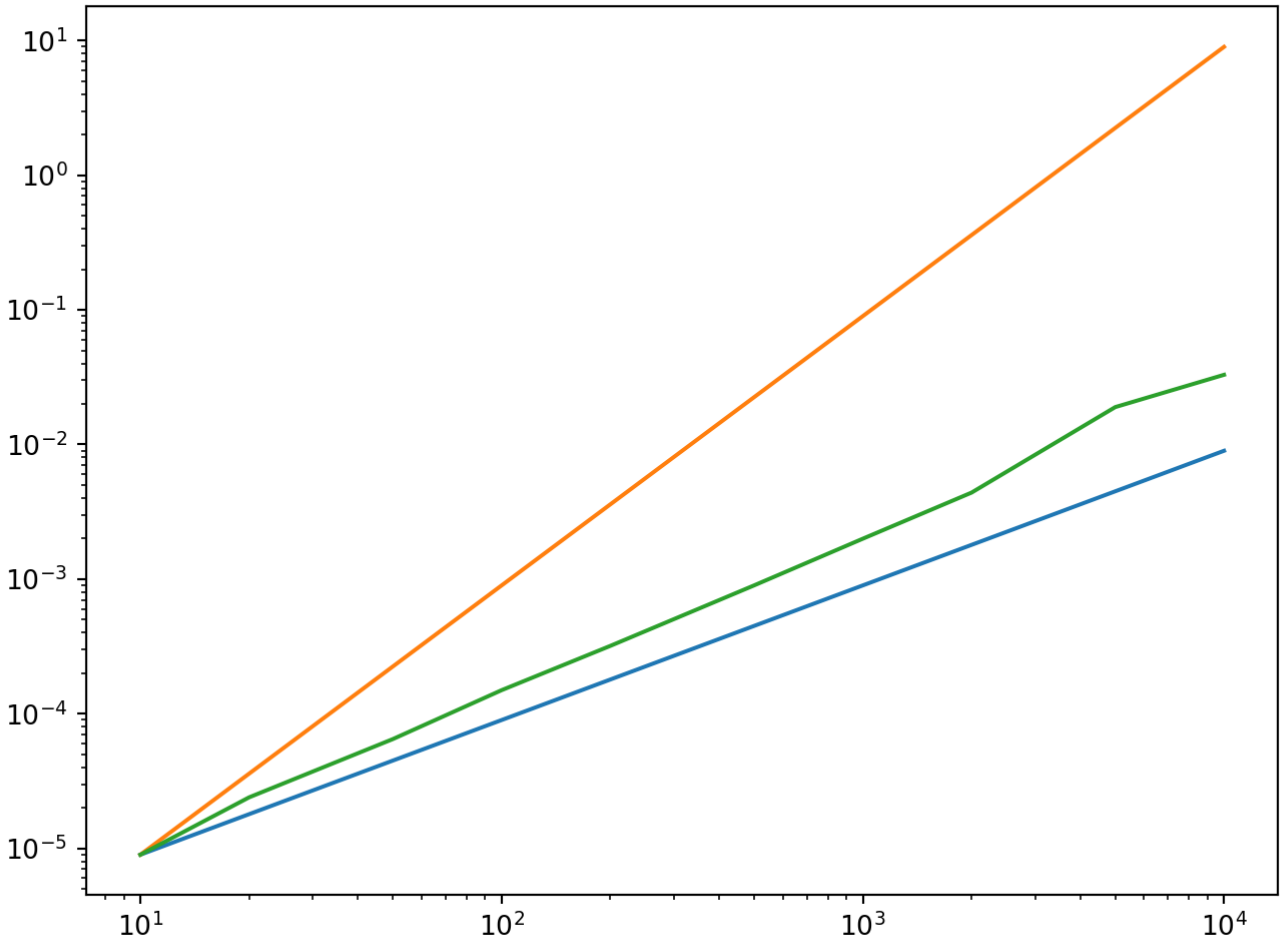
```
check_sort_result(mergesort,GID.tolist())
```

```
## The method    mergesort    returned successfully for data size 20000 with 0.05385756492614746 seconds.
```

```
check_sort_result(AVL_sort,GID.tolist())
# check_sort_result(BST_sort,GID.tolist())
```

```
## The method     AVL_sort    returned successfully for data size 20000 with 0.06581926345825195 seconds.
```

```
check_performance(mergesort,GID.tolist())
```


10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{1}
10^{2}
10^{3}
10^{4}