Homework 1

Exercise 1

3-sum问题的时间复杂度为$O(n^2 \log n)$的算法

题目：

实现一个时间复杂度为$O(n^2 \log n)$的算法，解决3-sum问题。

源代码：

```
print("This is a python code for homework8-1: 3-sum Problem")
s = [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]
s.sort()


def binarySearch(arr, l, r, x):
    if r >= l:
        mid = int(l + (r - l) / 2)
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binarySearch(arr, l, mid - 1, x)
        else:
            return binarySearch(arr, mid + 1, r, x)
    else:
        return -1


def sum_3_problem(s, x):
    n = len(s)
    postion_temp = 0
    for i in range(n-1):
        for j in range(i+1, n):
            b = s[i]
            c = s[j]
            temp = x - b - c
            if binarySearch(s, 0, len(s)-1, temp) != -1:
                temp1 = i
                temp2 = j
                postion_temp = binarySearch(s, 0, len(s)-1, temp)
    if postion_temp == -1 or postion_temp == 0:
        return -1
    else:
        if postion_temp == i or postion_temp == j:
            return -1
        else:
```

```
            return s[temp1], s[temp2], s[postion_temp]


x = 152
ans = sum_3_problem(s, x)

if ans == -1:
    print("There are no 3 numbers that sum of them is", x)
else:
    print(x, '= sum of ', sum_3_problem(s, x))
```

运行结果：

```
/usr/local/bin/python3.9 "/Users/wangyijie/Library/Mobile
Documents/com~apple~CloudDocs/Study_in_USTC/杂事/python科学计算/hw8/hw8_1.py"

This is a python code for homework8-1: 3-sum Problem
152 = sum of  (32, 99, 21)

Process finished with exit code 0
```

可见运行结果正确！

Exercise 2

排序算法计算耗时的测量

题目：

生成长度为$100, 200, \ldots, 900, 1000$的由随机数构成的列表，分别测量插入排序、归并排序和快速排序这三种排序算法的运行时间。对于每个长度测量10次计算平均值。

源代码：

```
import timeit
print("This is a python code for homework8-2: Time of sorting methods")


def insertion_sort(s):
    n = len(s)
    for i in range(1, n):
        value = s[i]
        pos = i
        while pos > 0 and value < s[pos - 1] :
            s[pos] = s[pos - 1]
            pos -= 1
        s[pos] = value
    return s
```

```
def merge_ordered_lists(s1, s2):
    t = []
    i = j = 0
    while i < len(s1) and j < len(s2):
        if s1[i] < s2[j]:
            t.append(s1[i]); i += 1
        else:
            t.append(s2[j]); j += 1
    t += s1[i:]
    t += s2[j:]
    return t


def merge_sort(s):
    if len(s) <= 1:
        return s
    mid = len(s) // 2
    left = merge_sort(s[:mid])
    right = merge_sort(s[mid:])
    return merge_ordered_lists(left, right)


def qsort(s):
    if len(s) <= 1: return s
    s1 = [i for i in s if i < s[0]]
    s2 = [i for i in s if i > s[0]]
    s0 = [i for i in s if i == s[0]]
    return qsort(s1) + s0 + qsort(s2)


import random
def random_int_list(start, stop, length):
    start, stop = (int(start), int(stop)) if start <= stop else (int(stop), int(start))
    length = int(abs(length)) if length else 0
    random_list = []
    for i in range(length):
        random_list.append(random.randint(start, stop))
    return random_list


s1 = random_int_list(1, 100, 100)
s2 = random_int_list(1, 100, 200)
s3 = random_int_list(1, 100, 300)
s4 = random_int_list(1, 100, 400)
s5 = random_int_list(1, 100, 500)
s6 = random_int_list(1, 100, 600)
s7 = random_int_list(1, 100, 700)
```

```
s8 = random_int_list(1, 100, 800)
s9 = random_int_list(1, 100, 900)
s10 = random_int_list(1, 100, 1000)

set_up = """
import random
def random_int_list(start, stop, length):
    start, stop = (int(start), int(stop)) if start <= stop else (int(stop), int(start))
    length = int(abs(length)) if length else 0
    random_list = []
    for i in range(length):
        random_list.append(random.randint(start, stop))
    return random_list


s1 = random_int_list(1, 100, 100)
s2 = random_int_list(1, 100, 200)
s3 = random_int_list(1, 100, 300)
s4 = random_int_list(1, 100, 400)
s5 = random_int_list(1, 100, 500)
s6 = random_int_list(1, 100, 600)
s7 = random_int_list(1, 100, 700)
s8 = random_int_list(1, 100, 800)
s9 = random_int_list(1, 100, 900)
s10 = random_int_list(1, 100, 1000)


def insertion_sort(s):
    n = len(s)
    for i in range(1, n):
        value = s[i]
        pos = i
        while pos > 0 and value < s[pos - 1] :
            s[pos] = s[pos - 1]
            pos -= 1
        s[pos] = value
    return s


def merge_ordered_lists(s1, s2):
    t = []
    i = j = 0
    while i < len(s1) and j < len(s2):
        if s1[i] < s2[j]:
            t.append(s1[i]); i += 1
        else:
            t.append(s2[j]); j += 1
    t += s1[i:]
    t += s2[j:]
```

```
    return t


def merge_sort(s):
    if len(s) <= 1:
        return s
    mid = len(s) // 2
    left = merge_sort(s[:mid])
    right = merge_sort(s[mid:])
    return merge_ordered_lists(left, right)


def qsort(s):
    if len(s) <= 1: return s
    s1 = [i for i in s if i < s[0]]
    s2 = [i for i in s if i > s[0]]
    s0 = [i for i in s if i == s[0]]
    return qsort(s1) + s0 + qsort(s2)
"""

t1q = """
qsort(s1)
"""
t2q = """
qsort(s2)
"""
t3q = """
qsort(s3)
"""
t4q = """
qsort(s4)
"""
t5q = """
qsort(s5)
"""
t6q = """
qsort(s6)
"""
t7q = """
qsort(s7)
"""
t8q = """
qsort(s8)
"""
t9q = """
qsort(s9)
"""
t10q = """
qsort(s10)
```

```
"""
t1i = """
insertion_sort(s1)
"""
t2i = """
insertion_sort(s2)
"""
t3i = """
insertion_sort(s3)
"""
t4i = """
insertion_sort(s4)
"""
t5i = """
insertion_sort(s5)
"""
t6i = """
insertion_sort(s6)
"""
t7i = """
insertion_sort(s7)
"""
t8i = """
insertion_sort(s8)
"""
t9i = """
insertion_sort(s9)
"""
t10i = """
insertion_sort(s10)
"""
t1m = """
merge_sort(s1)
"""
t2m = """
merge_sort(s2)
"""
t3m = """
merge_sort(s3)
"""
t4m = """
merge_sort(s4)
"""
t5m = """
merge_sort(s5)
"""
t6m = """
merge_sort(s6)
"""
```

```
t7m = """
merge_sort(s7)
"""
t8m = """
merge_sort(s8)
"""
t9m = """
merge_sort(s9)
"""
t10m = """
merge_sort(s10)
"""
N = 10
print("**************************** Insertion sort *********************************")
print("Time for Insertion sort with array (100 elements):", timeit.timeit(stmt=t1i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (200 elements):", timeit.timeit(stmt=t2i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (300 elements):", timeit.timeit(stmt=t3i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (400 elements):", timeit.timeit(stmt=t4i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (500 elements):", timeit.timeit(stmt=t5i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (600 elements):", timeit.timeit(stmt=t6i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (700 elements):", timeit.timeit(stmt=t7i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (800 elements):", timeit.timeit(stmt=t8i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (900 elements):", timeit.timeit(stmt=t9i,
setup=set_up, number=N)/N)
print("Time for Insertion sort with array (1000 elements):", timeit.timeit(stmt=t10i,
setup=set_up, number=N)/N)
print("***************************** Merge sort **********************************")
print("Time for Merge sort with array (100 elements):", timeit.timeit(stmt=t1m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (200 elements):", timeit.timeit(stmt=t2m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (300 elements):", timeit.timeit(stmt=t3m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (400 elements):", timeit.timeit(stmt=t4m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (500 elements):", timeit.timeit(stmt=t5m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (600 elements):", timeit.timeit(stmt=t6m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (700 elements):", timeit.timeit(stmt=t7m,
setup=set_up, number=N)/N)
```

```
print("Time for Merge sort with array (800 elements):", timeit.timeit(stmt=t8m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (900 elements):", timeit.timeit(stmt=t9m,
setup=set_up, number=N)/N)
print("Time for Merge sort with array (1000 elements):", timeit.timeit(stmt=t10m,
setup=set_up, number=N)/N)
print("**************************** Quick sort ***********************************")
print("Time for quick sort with array (100 elements):", timeit.timeit(stmt=t1q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (200 elements):", timeit.timeit(stmt=t2q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (300 elements):", timeit.timeit(stmt=t3q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (400 elements):", timeit.timeit(stmt=t4q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (500 elements):", timeit.timeit(stmt=t5q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (600 elements):", timeit.timeit(stmt=t6q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (700 elements):", timeit.timeit(stmt=t7q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (800 elements):", timeit.timeit(stmt=t8q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (900 elements):", timeit.timeit(stmt=t9q,
setup=set_up, number=N)/N)
print("Time for quick sort with array (1000 elements):", timeit.timeit(stmt=t10q,
setup=set_up, number=N)/N)
```

运行结果：

```
/usr/local/bin/python3.9 "/Users/wangyijie/Library/Mobile
Documents/com~apple~CloudDocs/Study_in_USTC/杂事/python科学计算/hw8/hw8_2.py"

This is a python code for homework8-2: Time of sorting methods
**************************** Insertion sort *********************************
Time for Insertion sort with array (100 elements): 5.262820000000446e-05
Time for Insertion sort with array (200 elements): 0.00021481610000000372
Time for Insertion sort with array (300 elements): 0.000435657300000003
Time for Insertion sort with array (400 elements): 0.0008267982000000007
Time for Insertion sort with array (500 elements): 0.0014173490999999983
Time for Insertion sort with array (600 elements): 0.0018880520000000012
Time for Insertion sort with array (700 elements): 0.002716698000000001
Time for Insertion sort with array (800 elements): 0.0035952701999999947
Time for Insertion sort with array (900 elements): 0.0037587039000000046
Time for Insertion sort with array (1000 elements): 0.0055716140000000025
***************************** Merge sort ***********************************
Time for Merge sort with array (100 elements): 0.0002851727000000026
Time for Merge sort with array (200 elements): 0.0005708688999999989
Time for Merge sort with array (300 elements): 0.000888429199999996
```

```
Time for Merge sort with array (400 elements): 0.0012131353000000011
Time for Merge sort with array (500 elements): 0.0015551584999999978
Time for Merge sort with array (600 elements): 0.0018737004000000002
Time for Merge sort with array (700 elements): 0.002266307200000006
Time for Merge sort with array (800 elements): 0.0022974979000000006
Time for Merge sort with array (900 elements): 0.0030652296999999938
Time for Merge sort with array (1000 elements): 0.0027567341999999995
****************************** Quick sort ************************************
Time for quick sort with array (100 elements): 0.00013827119999999528
Time for quick sort with array (200 elements): 0.00036119879999999547
Time for quick sort with array (300 elements): 0.00047635750000000197
Time for quick sort with array (400 elements): 0.0005021141000000062
Time for quick sort with array (500 elements): 0.0006392033000000019
Time for quick sort with array (600 elements): 0.0008065388000000007
Time for quick sort with array (700 elements): 0.0008404405000000059
Time for quick sort with array (800 elements): 0.0010616049000000017
Time for quick sort with array (900 elements): 0.0010742731000000006
Time for quick sort with array (1000 elements): 0.0013463303999999953

Process finished with exit code 0
```

Exercise 3

归并排序的函数每条语句的时间性能

题目：

测量归并排序的两个函数中的每条语句的时间性能。

源代码：

```
import timeit
import random
print("This is a python code for homework8-3: Time of merge sorting")


@profile
def merge_ordered_lists(s1, s2):
    t = []
    i = j = 0
    while i < len(s1) and j < len(s2):
        if s1[i] < s2[j]:
            t.append(s1[i]); i += 1
        else:
            t.append(s2[j]); j += 1
    t += s1[i:]
    t += s2[j:]
    return t
```

```
@profile
def merge_sort(s):
    if len(s) <= 1:
        return s
    mid = len(s) // 2
    left = merge_sort(s[:mid])
    right = merge_sort(s[mid:])
    return merge_ordered_lists(left, right)


def random_int_list(start, stop, length):
    start, stop = (int(start), int(stop)) if start <= stop else (int(stop), int(start))
    length = int(abs(length)) if length else 0
    random_list = []
    for i in range(length):
        random_list.append(random.randint(start, stop))
    return random_list


s1 = random_int_list(1, 100, 100)
s1 = merge_sort(s1)
```

运行结果：

```
wangyijie@wangyijedeMBP10 ~/L/M/c/S/杂/p/hw8 [2]> kernprof -l -v hw8_3.py
This is a python code for homework8-3: Time of merge sorting
Wrote profile results to hw8_3.py.lprof
Timer unit: 1e-06 s

Total time: 0.001188 s
File: hw8_3.py
Function: merge_ordered_lists at line 6

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     6                                           @profile
     7                                           def merge_ordered_lists(s1, s2):
     8        99         38.0      0.4      3.2      t = []
     9        99         35.0      0.4      2.9      i = j = 0
    10       641        367.0      0.6     30.9      while i < len(s1) and j < len(s2):
    11       542        274.0      0.5     23.1          if s1[i] < s2[j]:
    12       268        154.0      0.6     13.0              t.append(s1[i]); i += 1
    13                                               else:
    14       274        172.0      0.6     14.5              t.append(s2[j]); j += 1
    15        99         62.0      0.6      5.2      t += s1[i:]
    16        99         54.0      0.5      4.5      t += s2[j:]
    17        99         32.0      0.3      2.7      return t
```

```
Total time: 0.002596 s
File: hw8_3.py
Function: merge_sort at line 20

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    20                                           @profile
    21                                           def merge_sort(s):
    22       199        234.0      1.2      9.0      if len(s) <= 1:
    23       100         26.0      0.3      1.0          return s
    24        99         48.0      0.5      1.8      mid = len(s) // 2
    25        99        129.0      1.3      5.0      left = merge_sort(s[:mid])
    26        99        125.0      1.3      4.8      right = merge_sort(s[mid:])
    27        99       2034.0     20.5     78.4      return merge_ordered_lists(left,
right)
```