Beihang University Beijing, Nov. 28th, 2018

Computer Science and Programming
Lab Class 8

Task 1. ***Sort algorithm – Merge sort*** *(25 minutes)*

Write a function *merge_sort*() to sort an integer list in ascending order with merge sort algorithm.

Task 2. ***Sort algorithm – Comparison*** *(20 minutes)*

Use the following function to generate a random integer series. Parameter "start" and "end" represents the range of integers while "length" marks the length of integer series.

```
import random
def random_int_list(start, stop, length):
    random_list = []
    for i in range(length):
        random_list.append(random.randint(start, stop))
    return random_list
```

Compare the execution time of insertion sort, bubble sort, random sort and merge sort with the **same** integer list generated by the above function. See how the execution time of these sorting algorithm rises as the length of integer list increases (length = 10, 20, 50, 100, 1000, etc.).

Task 3. ***Comparison – Visualization*** *(10 minutes)*

Use the following function to visualize the running time for the above sort algorithms for sorting different sizes of lists. Here, *size* is the list of sizes (*size[j]* is the size of the *j*-th lists) and *T* is a matrix of running time for different algorithms for sorting different sizes of lists (*T[i][j]* is the running time for the *i*-th algorithm for sorting the *j*-th lists).

```
import matplotlib.pyplot as plt
def vis(size,T):
    colors=["bo-","r*--","gs-."]
    labels=["Insertion_sort","Bubble_sort","Random_sort"]
    for i in range(len(T)):
```

```
        plt.plot(size,T[i],colors[i],label=labels[i])
    plt.xscale("log")
    plt.yscale("log")
    plt.xlim(size[0]/5,size[-1]*5)
    plt.legend()
    plt.show()
```

Task 4. ***Sort algorithm for classes*** *(20 minutes)*

Build a **Student** class which stores the **name**s and the **ID**s of students. Implement the a sort algorithm which sorts a list $[s1, s2, s3, ..., sn]$ **Student** objects by their names and IDs (The **Student**s are first sorted by their names. If several students have the same names, then sort them by their IDs.). Try your code on the students $[(jim, 3), (jane, 2), (jim, 1)]$, where the first entry is the name and the second entry is the ID, respectively.

Task 5. ***Computational complexity*** *(25 minutes)*

(1) Please determine the overall complexity of the following two functions:
$f(n) = n^{10} + 2^n + lg(n) = O(\_\_\_\_)$.
$f(n) = \frac{1}{n} + \frac{n}{10} = O(\_\_\_\_)$
(2) The recursive code for Pascal's triangle is shown as follows. Please compute the computational complexity and use $O(f(n))$ to denote the result.

```
def Pas(n):
    if n==1:
        L=[1]
        print (L)
        return L
    else:
        lastL=Pas(n-1)
        newL=[1]
        for j in range(1,len(lastL)):
            newL.append(lastL[j-1]+lastL[j])
        newL.append(1)
        print (newL)
        return newL
```

(3) Assume that a divide-and-conquer algorithm has a time complexity of $T(n)$. Please write the Big O notation of its time complexity within each of the following recurrence relations. You can use the Master method from the lecture:
a) $T(n) = 2T(n/2) + 3n$
b) $T(n) = 2T(n/2) + n^2$

Task 6. ***Implementing naive divide-and-conquer based multiplication*** *(15 minutes)*

Implement naive the multiplication algorithm (naive recursive multiply as introduced in the lecture) which needs quadratic run time. If you are finished early, you can also try to implement Karatsubas algorithm in addition.

Task 7. ***Debug*** *(25 minutes)*

There are 13 python files with some bugs that you have met in folder "bugs". Please find out the bugs and correct the code. The correct result is stated at the end of each file.