Beihang University Beijing, Dec. 12th, 2018

Computer Science and Programming
Lab Class 10

Task 1. ***Time complexity*** *(15 minutes)*

(1) Write the recursive equation $T(n)$ for the merge sort. Then, estimate the time complexity of merge sort algorithm.

```
def merge_sort(L):
    if len(L)==1:
        return (L)
    else:
        n=len(L)
        L_left=merge_sort(L[:int(n/2)])
        L_right=merge_sort(L[int(n/2):])
        L_new=[]
        i=0
        j=0
        while i<len(L_left) and j<len(L_right):
            if L_left[i]<L_right[j]:
                L_new.append(L_left[i])
                i+=1
            else:
                L_new.append(L_right[j])
                j+=1
        L_new+=L_left[i:]
        L_new+=L_right[j:]
        return (L_new)
```

(2) Write the recursive equation $T(n)$ for Karatsubas Algorithm. Then, estimate the time complexity of Karatsubas Algorithm.

```
def multiply(x,y):
    #base case
    if len(str(x))==1 or len(str(y))==1:
        return x*y
    #recursive case
    else:
        n=max(len(str(x)),len(str(y)))
        nby2=int(n/2)
        #Compute x=ab, y=cd
        a=int(x/(10**nby2))
```

```
        b=int(x%(10**nby2))
        c=int(y/(10**nby2))
        d=int(y%(10**nby2))

        #Multiply recursively Karatsuba-style
        ac=multiply(a,c)
        bd=multiply(b,d)
        ad_plus_bc=multiply(a+b,c+d)-ac-bd
        prod=ac*10**(2*nby2)+(ad_plus_bc*10**nby2) + bd
        return int(prod)
```

Task 2. ***Search and dictionary*** *(15 minutes)*

(1) Build a dictionary *students* in Python to store basic information for students. The keys of students are their ID. The value of each student is a sub-dictionary that records the **Name**, **Age** and **Score** of each student. For instance, we have *student[1]['Name']='Lisa'*, *student[3]['Age']=19*, and *student[4]['Score']=68*. The full information for all students is shown in Table 1.

Table 1

ID	Name	Age	Score
1	Lisa	18	90
2	Amy	18	85
3	John	19	59
4	Mark	17	68
5	Jack	16	75

(2) Find all students whose scores are larger than 60 and record their names in a list. Print the list.

(3) Find all students whose names contain the character 'a' and record their names and ages in a list. Print the list with your program (no need for formatting).

Task 3. ***Counting sort*** *(15 minutes)*

Write a function *counting_sort(L,k)* to sort a list L within linear time. Here, all elements in L are integers in the range 0 to k. Try your algorithm on $L = [2, 5, 3, 0, 2, 3, 0, 3]$, with $k = 6$.

Task 4. ***Full permutation*** *(15 minutes)*

Write a function *full_permutation(L)* to output the the list of full permutation for a given list *L*. For instance, the result for *L=[0, 1, 2]* is *[[0, 1, 2], [0, 2, 1], [1, 0, 2], [1, 2, 0], [2, 0, 1], [2, 1, 0]]*.

Hint: You can let each element in the list be the first number of the permutation. Then, the full permutation of the remaining elements can be used to construct the full permutation of original list.

Task 5. ***Implementing k-selection*** *(25 minutes)*

Write a function to select k -th smallest element in an unsorted list. Use the naive, random selection strategy introduced in the lecture.

Once you get the random selection strategy working, you can try to implement the median-of-median selection strategy and compare its run time to the random selection.