A zero mark will be given if you copy someone else's work or you let someone copy your work.

COMP122/22-19 Data Structures and Algorithms	0 – 40 points
Sorting	2022-04-04 *Due Date* — 2022-04-11
Class Code	
Student No.	DO NOT WRITE YOUR NAME

1. Illustrate the execution of the insertion-sort algorithm on the following input sequence:

$$16, 19, 10, 13, 15, 17, 18, 12, 11, 14.$$

List the intermediate sequences after each time an element is properly inserted. For each sequence, you must put a marker on the element just been inserted and put a separator between the sorted part and the unsorted part. 8 points

16	|	19	10	13	15	17	18	12	11	14
16	(19)	|	10	13	15	17	18	12	11	14
(10)	16	19	|	13	15	17	18	12	11	14
10	(13)	16	19	|	15	17	18	12	11	14
10	13	(15)	16	19	|	17	18	12	11	14
10	13	15	16	(17)	19	|	18	12	11	14
10	13	15	16	17	(18)	19	|	12	11	14
10	(12)	13	15	16	17	18	19	|	11	14
10	(11)	12	13	15	16	17	18	19	|	14
10	11	12	13	(14)	15	16	17	18	19	|

(1)

2. Based on the idea of insertion sort, we keep popping elements from stack s and inserting them orderly to stack t, in order to sort the elements of s. Write a function *ins_sort_stack*(s, t, u) to implement this sorting algorithm *stably*, where (i) s contains the elements to sort, (ii) t, initially empty, stores the sorted elements in increasing order from top to bottom, and (iii) u is an auxiliary stack, possibly having other elements initially, to help insert elements to t. 10 points

```
def ins_sort_stack(s, t, u):

    while s:                          1
        n = 0                         1
        x = s.pop()                   1
        while t and t.top() <= x:     2
            u.push(t.pop())           1
            n += 1                    1
        t.push(x)                     1
        for i in range(n):            1
            t.push(u.pop())           1
```

(2)

3. To sort n elements in an array-based list, the idea of in-place stooge sort is to split n into three parts: $n_1+n_2+n_3=n$ and $n_1, n_2, n_3 \geqslant 1$, then recursively sort (i) the first n_1+n_2 elements, (ii) the last n_2+n_3 elements, and (iii) the first n_1+n_3 elements. The base case is when $0 \leqslant n \leqslant 2$, that is, n cannot be split into three parts. Usually, n_1, n_2, n_3 are approximately $\frac{n}{3}$ to be the most efficient.

a) Explain why this stooge sort produces the sorted elements. (7 points)

Step (i) makes $a[n_1 : n_1+n_2] > a[0 : n_1]$, so, there are n_2 elements in $a[n_1 : n]$ greater than $a[0 : n_1]$.

(2)

Step (ii) moves the greatest n_2 elements to $a[n-n_2 : n]$, so $a[n-n_2 : n] > a[0 : n_1]$. Also, $a[n-n_2 : n] > a[n_1 : n-n_2]$, therefore $a[n-n_2 : n] > a[0 : n_1+n_3]$, for $n-n_2 = n_1+n_3$.

(3)

Step (iii) sorts $a[0 : n_1+n_3]$, thus completes the sorting of the whole list.

(2)

(3)

b) Obviously, for each part taking $\frac{n}{3}$, the running time of stooge sort is $T(n) = 3T\left(\frac{2n}{3}\right)$. Derive the time complexity of stooge sort in the Big-Oh notation. (5 points)

$$T(n) = 3T\left(\frac{2n}{3}\right) = 3^2 T\left(\left(\frac{2}{3}\right)^2 n\right) = \cdots = 3^{\log_{\frac{3}{2}} n} T(1) \quad (2)$$

$$= 3^{\log_3 n \times \log_{\frac{3}{2}} 3} T(1) = n^{\log_{\frac{3}{2}} 3} T(1). \quad (2)$$

Since $T(1)$ is a constant, therefore, the time complexity of stooge sort is $\mathcal{O}(n^{\log_{\frac{3}{2}} 3}) \approx \mathcal{O}(n^{2.71})$. This is a very slow sorting algorithm.

(4) (1)

c) Write the recursive function *stooge_sort*(a, i, j) to sort the elements $a[i], a[i+1], \ldots, a[j-1]$ of list a by stooge sort. Assume $0 \leqslant i \leqslant j \leqslant$ `len`(a). (10 points)

```
def stooge_sort(a, i, j):
    if j-i >= 3:                              (1)
        k = (i*2+j)//3                        (1)
        l = (i+j*2)//3                        (1)
        stooge_sort(a, i, l)                  (1)
        stooge_sort(a, k, j)                  (1)
        stooge_sort(a, i, j-(l-k))            (2)
    elif j-i == 2:                            (1)
        if a[i] > a[i+1]:                     (1)
            a[i], a[i+1] = a[i+1], a[i]       (1)
```

(5)