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

COMP122/22-08 Data Structures and Algorithms	0 – 36 points
Fundamentals of Algorithm Analysis	2022-02-17 *Due Date* — 2022-02-21
Class Code	
Student No.	DO NOT WRITE YOUR NAME

1. For the f_1 function shown below:

```
def f1(a):  # The sum of the elements in list a
    i = len(a) % 2
    s = a[0] if i == 1 else 0
    while i < len(a):
        s += a[i]+a[i+1]
        i += 2
    return s
```

   a) How many times does the loop repeat, in terms of len(a) ? $\left\lfloor \frac{\texttt{len}(a)}{2} \right\rfloor$ ⁽¹⁾. 3 points

   b) Give a big-Oh characterization of the running time of f_1: $\mathcal{O}(\texttt{len}(a))$ ⁽²⁾. 2 points

2. For the f_2 function shown below:

```
def f2(a):  # The sum of the elements at every four cells in list a
    s = a[0]
    for i in range(4, len(a), 4):
        s += a[i]
    return s
```

   a) How many times does the loop repeat, in terms of len(a) ? $\left\lfloor \frac{\texttt{len}(a)-1}{4} \right\rfloor$ ⁽³⁾. 3 points

   b) Give a big-Oh characterization of the running time of f_2: $\mathcal{O}(\texttt{len}(a))$ ⁽⁴⁾. 2 points

3. For the f_3 function shown below:

```
def f3(a):  # The sum of the elements at each one-eighth of list a
    s = 0
    m = (len(a)+7)//8
    for i in range(0, len(a), m):
        s += a[i]
    return s
```

   Give a big-Oh characterization of the running time of f_3, in terms of len(a): $\mathcal{O}(1)$ ⁽⁵⁾. 3 points

4. Suppose stack s has n elements, for the f_4 function shown below:

```
def f4(s, t, x):
    while s:
        y = s.pop()
        if y != x:
            t.push(y)
```

   Give a big-Oh characterization of the running time of f_4, in terms of n: $\mathcal{O}(n)$ ⁽⁶⁾. 3 points

5. Suppose $n > 1$. For the f_5 function shown below:

```
def f5(n):
    t = 0
    i = 1
    while i < n**3:
        t += 1
        i *= 2
    return t
```

a) What is returned from the function, in terms of n? $\lfloor \log(n^3 - 1) \rfloor + 1$ (7). 2 points

b) Give a big-Oh characterization of the running time of f_5: $\mathcal{O}(\log n)$ (8). 3 points

6. To add the support of indexing to the linked list *LnLs* defined in Lesson 5, we need to locate the node at a given index, both forward and backward. If we are able to locate the node, we return its reference, otherwise we return None.

```
def fore_node(self, i): # assume i ⩾ 0.
    p = self.head
    for j in range(i):
        if p is None:
            return None
        p = p.nxt
    return p
```

```
def back_node(self, i): # assume i ⩾ 1.
    p = self.head
    while True:
        q = p
        for j in range(i):
            if q is None:
                return None
            q = q.nxt
        if q is None:
            return p
        p = p.nxt
```

Suppose a linked list has n nodes.

a) The *fore_node* method looks for the node at index i. Give a big-Oh characterization of the *worst case* running time of *fore_node*, in terms of n:

$\mathcal{O}(n)$ (9). 3 points

b) Give an example to describe the worse case of the *fore_node* method:

When $i \geqslant n$, obviously we need to repeat exactly n times (10). 2 points

c) The *back_node* method looks for the node at index $-i$. Give a big-Oh characterization of the *worst case* running time of *back_node*, in terms of n:

$\mathcal{O}(n^2)$ (11). 3 points

d) Give an example to describe the worse case of the *back_node* method:

When $i = n/2$, each p of the first $n/2$, must go through the inner loop of $n/2$ repetitions, that is $n^2/4$ (12). 2 points

e) Give a big-Oh characterization of the *best case* running time of *back_node*, in terms of n:

$\mathcal{O}(n)$ (13). 3 points

f) Give an example to describe the best case of the *back_node* method:

When $i = n$, only the first p needs to go through the inner loop of n repetitions, then the outer loop stops because q hits the end. (14). 2 points