COMP9101 Assignment 1

Name: Xuhua Le
Student Number: z5047516

Question 1.

Solution:

Algorithm:

Firstly, use "Merge-Sort" algorithm to sort array A.
Then, for $k = 1$ to n, do:
    $index_1 \leftarrow binary - search - for - left(A, L_k)$
    $index_2 \leftarrow binary - search - for - right(A, R_k)$
    if $index_2 \geq index_1$:
        $count \leftarrow index_2 - index_1 + 1$
    else:
        $count \leftarrow 0$
    output $count$

PS:

(1). Merge-Sort(A, p, r):
    if $p < r$:
        $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
        Merge-Sort(A, p, q)
        Merge-Sort(A, $q+1$, r)
        Merge(A, p, q, r)

(2). binary-search-for-left(A, L_k):
    if $L_k > A[length(A) - 1]$:
        return $length(A)$
    $left \leftarrow 0$
    $right \leftarrow length(A) - 1$
    $index \leftarrow \lfloor \frac{left+right}{2} \rfloor$
    while $left \leq right$:
        if $A[index] < L_k$:
            $left \leftarrow index + 1$
            $index \leftarrow \lfloor \frac{left+right}{2} \rfloor$
        else:
            if $index == 0$:
                return $index$
            else:
                if $A[index - 1] < L_k$:
                    return $index$
                else:
                    $right \leftarrow index - 1$
                    $index \leftarrow \lfloor \frac{left+right}{2} \rfloor$
    return $index$

```
(3). binary-search-for-right(A, R_k):
        if R_k < A[0]:
            return -1
        left ← 0
        right ← length(A) − 1
        index ← ⌊(left+right)/2⌋
        while left ≤ right:
            if A[index] > R_k:
                right ← index − 1
                index ← ⌊(left+right)/2⌋
            else:
                if index == length(A) − 1:
                    return index
                else:
                    if A[index + 1] > R_k:
                        return index
                    else:
                        left ← index + 1
                        index ← ⌊(left+right)/2⌋
        return index
```

As Merge-Sort satisfies $O(n * \log n)$, and this two binary search processes satisfies $O(\log n)$, multiplying with the n queries, which becomes $O(n * \log n)$ as well. Hence, this algorithm satisfies $O(n * \log n)$.

Question 2.

Solution:

(a).

Algorithm:

```
    Firstly, use "Merge-Sort" algorithm to sort array S.
    For i = 0 to (n − 1), do:
        Use binary search to see whether (x − S[i]) exists in array S.
        if (x − S[i]) gets found in array S :
            if x ≠ 2 ∗ S[i]:
                return True
            if x == 2 ∗ S[i]:
                if S[i − 1] == S[i] or S[i + 1] == S[i]:
                    return True
    return False
```

As Merge-Sort satisfies $O(n*\log n)$, and the for-loop satisfies $O(n*\log n)$, hence, this algorithm satisfies $O(n * \log n)$.

(b).
Algorithm:

Firstly, build an empty hash table T.
Then, for $i = 0$ to $(n - 1)$, do:
if $T's$ slot $S[i]$ is empty:
Store a pair of value $P = (S[i], 1)$ in T with hash value $= S[i]$.
else:
Store a pair of value $P = (S[i], 2)$ in T with hash value $= S[i]$.
For $i = 0$ to $(n - 1)$, do:
if $(x - S[i])$ gets found in T:
if $x \neq 2 * S[i]$:
return True
if $x == S[i]$:
if $P[1] == 2$: (PS: This means there are at least two values equal to $S[i]$ in T)
return True
return False

As building an empty hash table takes $O(1)$ time, storing n numbers into a hash table takes $O(n)$ time, searching n numbers in a hash table takes $O(1)$ * n = $O(n)$ time, hence, this algorithm takes $O(n)$ time.

Question 3.

Solution:

According to the question, we could get some information below:
1. Celebrity requirements: known by everyone, but celebrity does not know anyone else except himself/herself.
2. There could exist at most 1 celebrity. Assume A and B both are celebrity, then A should not know B and B should not know A as well because the celebrity could not know anyone else. However, due to the celebrity identity of A, B should know A, and A should know B vice verse. Hence, there is a contradiction, so there would exist at most 1 celebrity.
3. Ask a person X if X knows another person Y. If X knows Y, then X could not be celebrity as celebrity could not know anyone else. If X does not know Y, then Y could be celebrity as celebrity needs to be know by everyone. Hence, as long as we ask a person X if X knows another person Y, one of them needs to be eliminated out of the celebrity identity by only 1 question.

(a).
Part-1 process:

Number the n persons as $a_1, a_2, ..., a_n$.
Build a list $L = [a_1, a_2, ..., a_n]$.
while $(L.length \neq 1)$:
Ask $L[0]$ if $L[0]$ knows $L[1]$.

If $L[0]$ knows $L[1]$, do:
Eliminates $L[0]$ from L.
If $L[0]$ does not know $L[1]$, do:
Eliminates $L[1]$ from L.

PS: This while-loop process would take $(n-1)$ questions and eliminates $(n-1)$ persons.

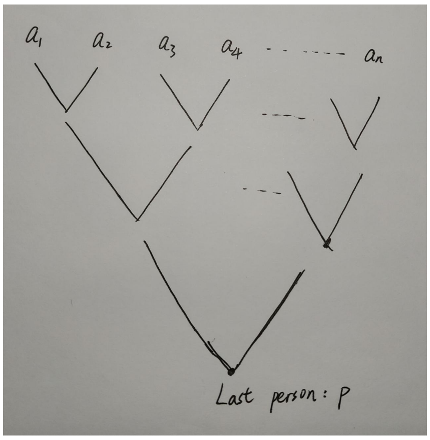
Part-2 process: As there is only 1 person in L staying unidentified (name the last person left as "P"), ask the other $(n-1)$ persons whether they all know P. If there exists 1 person does not know P, then return False.

Part-3 process: Ask P whether P knows the other $(n-1)$ persons. If P knows at least one of them, then return False. If P does not know anyone of them, then P is the celebrity, then return P.

In the worst case, part-2 process could take $(n-1)$ questions and part-3 process could take $(n-1)$ questions as well. Hence, the number of total questions are $(n-1)+(n-1)+(n-1) = 3n-3$. However, in part-1 process, P could be asked that if P knows other persons, or other person could be asked that if them know P, and this type of questions would be duplicated at least 1 time in part-2 or part-3 process. As a result, this algorithm could take $3n-3-1 = 3n-4$ questions in the worst case, and it satisfies the requirements.

(b).
Part-1 process:
Number the n persons as $a_1, a_2, ..., a_n$.
Build a list $L = [a_1, a_2, ..., a_n]$.
while ($L.length \neq 1$):
For i = 1, 3, 5, ..., (separated by 2) do:
Set two neighboring persons a_i and a_{i+1} as a pair (a_i, a_{i+1}).
For each pair (a_i, a_{i+1}), ask a_i whether a_i knows about his/her partner a_{i+1}.
If a_i knows a_{i+1}, do:
Eliminates a_i from L.
If a_i does not know a_{i+1}, do:
Eliminates a_{i+1} from L.

PS: Just like a inverted binary-tree, and here is the sketch map:



Part-2 process: As there is only 1 person in L staying unidentified (name the last person left as "P"), ask the other $(n-1)$ persons whether they all know P. If there exists 1 person does not know P, then return False.

Part-3 process: Ask P whether P knows the other $(n-1)$ persons. If P knows at least one of them, then return False. If P does not know anyone of them, then P is the celebrity, then return P.

In part-1 process, the number of questions is:
$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ... + 1 = \frac{\frac{n}{2}*(1-\frac{1}{2}^{\lfloor \log_2 n \rfloor -1})}{1-\frac{1}{2}} = n - n * 2^{1-\lfloor \log_2 n \rfloor}$

In the worst case, we need to do the whole process of part-2 and part-3. During part-1 process, there are 3 possible situations related to person P:
(1). P always be asked that if P knows other people, and P always answers No.
(2). Some other people always be asked that if they know P, and they always answer Yes.
(3). The combination of (1) and (2).

And the number of questions that relates to P in (1), (2) or (3) is the same, which is $(\lfloor log_2 n \rfloor - 1)$, the binary processing length of part-1 process. But these questions are duplicated in part-2 or part-3 process, hence, we could eliminate these repetitive questions.

As a result, in the worst case, the number of all the questions we need to ask is:
$(n - n * 2^{1-\lfloor log_2 n \rfloor}) + (n-1) + (n-1) - (\lfloor log_2 n \rfloor - 1)$
$= 3n - \lfloor log_2 n \rfloor - 1 - n * 2^{1-\lfloor log_2 n \rfloor}$

Besides, $n * 2^{1-\lfloor log_2 n \rfloor} = \frac{2n}{2^{\lfloor log_2 n \rfloor}} \geq \frac{2n}{2^{log_2 n}} = \frac{2n}{n} = 2$.

Hence, the total number of questions $\leq 3n - \lfloor log_2 n \rfloor - 1 - 2 = 3n - \lfloor log_2 n \rfloor - 3$. Therefore, this algorithm satisfies the requirements.

Question 4.

Solution:

1. $f(n) = (log_2 n)^2, g(n) = log_2(n^{log_2 n}) + 2 * log_2 n$

$\Rightarrow g(n) = log_2 n * log_2 n + 2 * log_2 n$
Set $c = 1$, $n_0 = 1$, then $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
$\Rightarrow f(n) = O(g(n))$.

Assume there exists positive numbers c and n_0, which satisfy $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
$\Leftrightarrow 0 \leq (log_2 n)^2 + 2 * log_2 n \leq c * (log_2 n)^2$
We could set $c = 2, n_0 = 4$, then we could get $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
$\Rightarrow g(n) = O(f(n))$.
As a result, $f(n) = \Theta(g(n))$.

2. $f(n) = n^{100}, g(n) = 2^{\frac{n}{100}}$

Assume there exists positive numbers c and n_0, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
$\Leftrightarrow n^{100} \leq c * 2^{\frac{n}{100}}$
$\Leftrightarrow \log_2 (n^{100}) \leq \log_2 (c * 2^{\frac{n}{100}})$
$\Leftrightarrow 100 * \log_2 n \leq \log_2 c + \frac{n}{100} * \log_2 2 = \log_2 c + \frac{1}{100} * n$
We could set $c = 2, n_0 = 2^{100}$, then we could get $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
$\Rightarrow f(n) = O(g(n))$.
Assume there exists positive numbers c and n_0, such that $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
$\Leftrightarrow \frac{n}{100} * \log_2 2 \leq \log_2 c + 100 * \log_2 n$
As for sufficient large n, n is much bigger than $\log_2 n$, hence, there does not exist such a constant c to satisfy the requirement.
Hence, there does not exist positive numbers c and n_0, such that $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
As a result, $f(n) = O(g(n))$.

3. $f(n) = \sqrt{n}, g(n) = 2^{\sqrt{\log_2 n}}$

Assume there exists positive numbers c and n_0, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
$\Leftrightarrow \sqrt{n} \leq c * 2^{\sqrt{\log_2 n}}$
$\Leftrightarrow \log_2 \sqrt{n} \leq \log_2 c + \log_2 2 * \sqrt{\log_2 n}$
$\Leftrightarrow \sqrt{\log_2 n} * (\frac{1}{2} * \sqrt{\log_2 n} - 1) \leq \log_2 c$
As $\log_2 n$ would get infinitely large for sufficient large n, we could not find such a constant c to meet this inequality .
Hence, there does not exist positive numbers c and n_0, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
Assume there exists positive numbers c and n_0, such that $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
$\Leftrightarrow 2^{\sqrt{\log_2 n}} \leq c * \sqrt{n}$
$\Leftrightarrow \sqrt{\log_2 n} \leq \log_2 c + \frac{1}{2} * \log_2 n$
$\Leftrightarrow 0 \leq \log_2 c + \sqrt{\log_2 n} * (\frac{1}{2} * \sqrt{\log_2 n} - 1)$
We could set $c = 1, n_0 = 16$, then we could get $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
$\Rightarrow g(n) = O(f(n))$, or $f(n) = \Omega(g(n))$.

4. $f(n) = n^{1.001}, g(n) = n * \log_2 n$

Assume there exists positive numbers c and n_0, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
$\Leftrightarrow n^{1.001} \leq c * n * \log_2 n$
$\Leftrightarrow n^{0.001} \leq c * \log_2 n$
We could set $n_0 \geq 2$, hence, $\log_2 n \geq 1$.
$\Leftrightarrow \frac{n^{0.001}}{\log_2 n} \leq c$
When n gets sufficient large, we need to find such a constant c to satisfy this inequality. However,

$$\lim_{n\to\infty} \frac{n^{0.001}}{\log_2 n} = \lim_{n\to\infty} \frac{0.001 * n^{-0.999}}{\frac{1}{n*ln2}} = \lim_{n\to\infty} (0.001 * n^{0.001} * ln2) = \infty$$

Hence, we could not find a constant c to meet this inequality. In other words, there does not exist positive numbers c and n_0, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.

Assume there exists positive numbers c and n_0, such that $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
$\Leftrightarrow n * \log_2 n \leq c * n^{1.001}$
$\Leftrightarrow \frac{\log_2 n}{n^{0.001}} \leq c$
Besides, when n gets sufficient large, we could get:

$$\lim_{n\to\infty} \frac{\log_2 n}{n^{0.001}} = \lim_{n\to\infty} \frac{\frac{1}{n*ln2}}{0.001 * n^{-0.999}} = \lim_{n\to\infty} \frac{1}{0.001 * n^{0.001} * ln2} = 0$$

We could set $c = 1000, n_0 = 2^{1000}$, then the inequality would holds for all $n \geq n_0$, hence, $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
Hence, $g(n) = O(f(n))$.
As a result, $f(n) = \Omega(g(n))$.

5. $f(n) = n^{\frac{1+\sin(\frac{\pi n}{2})}{2}}, g(n) = \sqrt{n}$

Assume there exists positive numbers c and n_0, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
$\Leftrightarrow n^{\frac{1+\sin(\frac{\pi n}{2})}{2}} \leq c * n^{\frac{1}{2}}$
$\Leftrightarrow \frac{1+\sin(\frac{\pi n}{2})}{2} * \log_2 n \leq \log_2 c + \frac{1}{2} \log_2 n$
$\Leftrightarrow \frac{\sin(\frac{\pi n}{2})}{2} * \log_2 n \leq \log_2 c$
If $n = 4k + 1$(k is an integer) and n is sufficient large, we could not find such a constant c to meet the inequality as $\sin(\frac{\pi n}{2}) = 1$.
Hence, there does not exist positive numbers c and n_0, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$.
Assume there exists positive numbers c and n_0, such that $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.

$\Leftrightarrow n^{\frac{1}{2}} \leq c * n^{\frac{1+\sin(\frac{\pi n}{2})}{2}}$
$\Leftrightarrow \frac{1}{2} * \log_2 n \leq \log_2 c + \frac{1+\sin(\frac{\pi n}{2})}{2} * \log_2 n$
$\Leftrightarrow 0 \leq \log_2 c + \frac{\sin(\frac{\pi n}{2})}{2} * \log_2 n$
If $n = 4k + 3$(k is an integer) and n is sufficient large, we could not find such a constant c to meet the inequality as $\sin(\frac{\pi n}{2}) = -1$.
Hence, there does not exist positive numbers c and n_0, such that $0 \leq g(n) \leq c * f(n)$ for all $n \geq n_0$.
As a result, $f(n) \neq O(g(n))$ and $f(n) \neq \Omega(g(n))$.

Question 5.

Solution:

(a). $T(n) = 2 * T(n/2) + n * (2 + \sin(n))$

$\Rightarrow a = 2, b = 2, f(n) = n * (2 + \sin(n))$
$\Rightarrow n^{\log_b a} = n^{\log_2 2} = n$
Assume there exists positive numbers c and n_0, such that $0 \leq f(n) \leq c * n$ for all $n \geq n_0$.
$\Leftrightarrow n * (2 + \sin(n)) \leq c * n$
$\Leftrightarrow 2 + \sin(n) \leq c$
We could set $c = 3, n_0 = 1$, then we could get $0 \leq f(n) \leq c * n$ for all $n \geq n_0$.
$\Rightarrow f(n) = O(n)$.
Assume there exists positive numbers c and n_0, such that $0 \leq n \leq c * f(n)$ for all $n \geq n_0$.
$\Leftrightarrow n \leq c * n * (2 + \sin(n))$
$\Leftrightarrow 1 \leq c * (2 + \sin(n))$
We could set $c = 1, n_0 = 1$, then we could get $0 \leq n \leq c * f(n)$ for all $n \geq n_0$.
$\Rightarrow n = O(f(n))$, or $f(n) = \Omega(n)$.
As a result, $f(n) = \Theta(n)$.
Condition of Master Theorem's case 2 is satisfied.
Then $T(n) = \Theta(n^{log_b a} * \log_2 n) = \Theta(n * \log_2 n)$.

(b). $T(n) = 2 * T(n/2) + \sqrt{n} + \log n$

$\Rightarrow a = 2, b = 2, f(n) = \sqrt{n} + \log n$
$\Rightarrow n^{\log_b a} = n^{\log_2 2} = n$
Assume $f(n) = O(n^{(\log_b a) - \varepsilon}) = O(n^{1-\varepsilon})$, for some $\varepsilon > 0$.
Which means there exists positive numbers c and n_0, such that $0 \leq f(n) \leq c * n^{1-\varepsilon}$ for all $n \geq n_0$.
$\Leftrightarrow n^{\frac{1}{2}} + \log n \leq c * n^{1-\varepsilon}$
We could set $c = 2$, then we get:
$n^{\frac{1}{2}} + \log n \leq n^{1-\varepsilon} + n^{1-\varepsilon}$
We could set $\varepsilon = 1/2$, then we get:
$n^{\frac{1}{2}} + \log n \leq n^{\frac{1}{2}} + n^{\frac{1}{2}}$

We could set $n_0 = 2^{10}$, which means that this inequality holds for all $n \geq n_0$.
$\Leftrightarrow f(n) = O(n^{(\log_b a)-\varepsilon}) = O(n^{1-\varepsilon})$, for some $\varepsilon > 0$.
Condition of Master Theorem's case 1 is satisfied.
Then $T(n) = \Theta(n^{log_b a}) = \Theta(n)$.

(c). $T(n) = 8 * T(n/2) + n^{\log n}$

$\Rightarrow a = 8, b = 2, f(n) = n^{\log n}$
$\Rightarrow n^{\log_b a} = n^{\log_2 8} = n^3$
Assume $f(n) = \Omega(n^{3+\varepsilon})$ for some $\varepsilon > 0$.
$\Leftrightarrow$ There exists positive numbers c and n_0, such that $0 \leq c{*}n^{3+\varepsilon} \leq f(n) = n^{\log n}$ for all $n \geq n_0$.
We could set $c = 1, \varepsilon = 1$, and we get:
$n^4 \leq n^{\log n}$
We could set $n_0 = 2^4 = 16$, then this inequality holds for all $n \geq n_0$.
$\Leftrightarrow f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ for some $\varepsilon > 0$.
Assume for some $0 < c < 1$, $a * f(\frac{n}{b}) \leq c * f(n)$.
$\Leftrightarrow 8 * (\frac{n}{2})^{\log \frac{n}{2}} \leq c * n^{\log n}$
$\Leftrightarrow \log_2 8 + \log_2 \frac{n}{2} * \log_2 \frac{n}{2} \leq \log_2 c + \log_2 n * \log_2 n$
$\Leftrightarrow 3 + (\log_2 n - \log_2 2) * (\log_2 n - \log_2 2) \leq \log_2 c + \log_2 n * \log_2 n$
$\Leftrightarrow 3 + (\log_2 n)^2 - 2 * \log_2 n + 1 \leq \log_2 c + (\log_2 n)^2$
$\Leftrightarrow 4 \leq \log_2 c + 2 * \log_2 n$
We could set $c = \frac{1}{2}, n_0 = 2^3$, then this inequality holds for all $n \geq n_0$.
$\Leftrightarrow$ For some $0 < c < 1$, $a * f(\frac{n}{b}) \leq c * f(n)$.
Condition of Master Theorem's case 3 is satisfied.
Hence, $T(n) = \Theta(f(n)) = \Theta(n^{\log n})$.

(d). $T(n) = T(n-1) + n$

Hence, we could conclude these equations below:
$T(n) = T(n-1) + n$
$T(n-1) = T(n-2) + n - 1$
$T(n-2) = T(n-3) + n - 2$
......
$T(2) = T(1) + 2$
We could sum these equations together, and we get:
$T(n) = T(1) + n + (n-1) + (n-2) + ... + 2$
$\Rightarrow T(n) = T(1) + \frac{n*(1+n)}{2} - 1$
Hence, $T(n) = \Theta(n^2)$.