COMP3121 Assignment1

Q1:

Step1: Firstly, It is better to use merge sort because the worst case of merge sort algorithm is nlog(n).

Step2: Then, We can use binary search to check whether or not element a is exist. The sorted list A must have the head and tail, so it is easily to find medium position and get the medium number. Iteration the function to find whether or not element a is exist . Comparing the medium number with the L(k) and R(k) , if the medium number greater than L(k), we should check the array A between medium number and maximum number. Otherwise, check the other side until medium number between minimum and maximum. This step same with R(k).
Find one query should spend 2*log(N) time complexity. So , it will spend 2*N*log(N) for n query.

Overall, it will spend N*log(N) in sort and 2*N*log(N) in search. We can conclude that N*log(N) for all time complexity.
There are Pseudocode to explain the binary search:

```
BinaryL(k) (A,low,high)     //A is sorted array by increase order
        medium = (low+high)/2
        If L(k) > A [medium]
            Binary(A,medium,high)
        else If L(k) < A [medium]
            Binary(A,low,medium)
        else if L(k) == A [medium]
            return medium
    return 0
```

Q2:
A:

Checking whether or not exist two number whose sum equal to integer x in array S.

Step1: Firstly, sort the array s by using merge sort and it cost N*log(N) time complexity.

Step2: Then, we should define two pointers. One pointer on tail and another on head in array S. Sum the head and tail which pointer located. Then, the target integer x compare to sum. If the sum less than x, tail pointer move left otherwise head pointer move to right. Repeat the step until find the result.

Overall, it spend N*log(N) in sort and N to search . We can conclude that N*log(N) for all time complexity.
There are Pseudocode to explain the Two sum:

```
Two sum(S,tail,head)     //S is sorted array list.
    If S[tail]+S[head] < x&&tail<=head
        Two sum(S,tail,head+1)
    else If S[tail]+S[head] > x&&tail<=head
        Two sum(S,tail-1,head)
    else If S[tail]+S[head]==x
        return true
    return false
```

B:

Use the Hash table algorithm to solve O(n) problem. Loop the Array S, each element a will be save in hash table and at the meantime check whether or not exist x-a element in S. If yes, find the x. Otherwise, loop continue until end of array.
Overall, It spend O(n) to find is or not exist target x.

There are Pseudocode for hash:

```
HashTwoSum(S):
    For i in S
        If Hash(x-i) exist
            Return true
        Hash(i) = x-i
```

Q3:
A:
Give a example for a b c d four candidates.

```
        A    B    C    D        A ask B if not
         \  /      \  /        C ask D if not
          A         C              A ask C if not
             \     /
               A                   finish
```

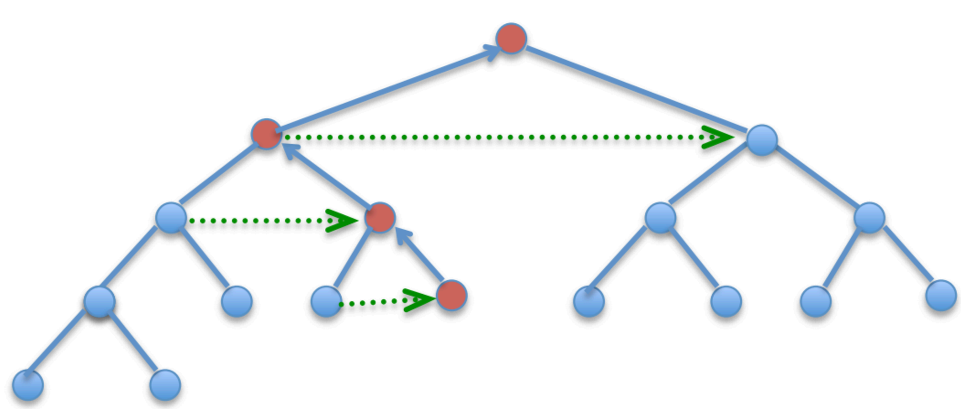
Step1: For example, let person a ask b weather or not it knows b .If a say yes, then b have opportunity to be celebrity. Otherwise, a could be celebrity candidate. Ask same question for c and d, if c does not know d, c can be candidate. Finally, ask a and c, and a does not know c , then a can be candidate.

Step2:Therefore, we can make sure one candidate (who have chance) for ask one question at worst case. For n person to ask question, we have to ask n-1 question until the last one.

Step3: We should check whether or not the last person is celebrity. Let the last person ask everyone except itself and let everyone ask the last person. If every people say yes and the candidate say no, the last people will be celebrity otherwise have no celebrity. This time complexity is 2n-2.

Thus, time complexity for find celebrity is 3*(n-1) = 3n-3.

B:



Step1: Since the question a, It took 3n-3 questions to find a celebrity .

Step2: As we can see the picture, we can assume the candidate is a root and all the red node is the candidate and blue one is others. We can easily find every level exist the red candidate ask one question or others ask the red candidate. Therefore, we repeat log2(n) for each level. If remove duplicate questions, we can conclude that time complexity is 3n-3-log2(n).

Q4:

Using L' Hˆopital Rule to calculate this question,

If lim f(n) / g(n) = c (constance),the growth rate between f(n) g(n) will be flat. Thus, f(n) = Theta(g(n))

If lim f(n) / g(n) -> 0,the growth rate of f(n) become slowly compare with g(n), Thus, f(n) = O(g(n))

If lim f(n) / g(n) -> Infinite, the growth rate of f(n) become rapidly compare with g(n),Thus, f(n) = Omega(g(n))

1. $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \frac{(\log_2 n)^2}{\log_2 n^{\log_2 n} + 2*\log_2 n} = \frac{(\log_2 n)'}{(\log_2 n+2)'} = \frac{2*ln(n)}{2*ln(n)} = 1$

   Thus, f(n) = Theta(g(n))

2. $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \frac{100*100*(\log_2 n)}{n} = 0$

   Thus, f(n) = O(g(n))

3. $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \frac{\sqrt{n}}{2^{\sqrt{\log_2 n}}} = \lim_{n\to\infty}\frac{f(n)}{g(n)} = \frac{1}{2} * \sqrt{\log_2 n} = infinite$

   Thus, f(n) = Omega(g(n))

4. $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \frac{n^{1.001}}{n\log_2 n} = \frac{n}{n^{0.999} * \log_2 n} = 0$

   Thus, f(n) = O(g(n))

5. $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \frac{n^{(1+sin(\frac{\pi n}{2}))/2}}{\sqrt{n}}$ can not compare because sin() is wave function. It is not decide the growth rate between f(n) and g(n)

Q5:

a,b,c By using master Theory : T(n) = aT(n/b) +f(n)

a. a = 2,b = 2,f(n) = n(sin(n)+2)

$n^{\log_a b} = n^1 = n, since\ 1 \le n(\sin(n)+2) \le 3, then\ n \le n(\sin(n)+2) \le 3n, f(n) = \theta(n)$

The growth rate is $\theta(n * \log n)$.

b. a = 2,b = 2, f(n) $= \sqrt{n} + \log n$

$n^{\log_a b} = n^1 = n, by\ using\ L'$Hˆopital Rule $\lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \frac{n\sqrt{n}+\log n}{n} \to 0$

$Then, f(n) = 0(n^{1-\varepsilon})$

Then, the growth rate is θ(n).

c. a=8, b=2, f(n) $= n^{\log n}$

$n^{\log_b a} = n^3$, *by using* L'Hˆopital Rule $\lim_{n\to\infty} (\frac{f(n)}{g(n)}) = \frac{n^3}{n^{\log n}} \to \infty, Then, f(n) = \ omega(n^{3+\varepsilon})$

8*f(n/2)<=c*f(n) $\left(n^{\log \frac{n}{2}}\right) * 8 \leq c * n^{\log n}, so\ c = \frac{\log 2}{8}$

Then, the growth rate is $\theta\left(n^{\log n}\right)$.

d. the master theory is not applicable because none of this condition hold.

T(0) = 0. T(1) = T(0) + 1. T(2) = T(1) + 2 … T(n) = T(n-1) + n.

T(n) = n+n-1+n-2+…+1+0 = $\frac{n*(n+1)}{2}$ = O(n^2).

So, the growth rate is O(n^2).