COMP9101 ASSIGNMENT 1

DESIGN AND ANALYSIS OF ALGORITHMS

CONG CONG Z3414050

1.

First step, using merge sort algorithm to sort the n element array S --- O(nlogn)

Second step, for each query, trying to find the number of elements (SmallN) smaller than L_k,

And the number of elements (LargeN) larger the R_k, then the number of elements in between of L_k and R_k equals n – (SmallN + LargeN).

For the second part, we can use binary search to find the index of the largest element in the array that is smaller than L_k, --- O(log)

Similarly, using binary search to find the index of the smallest element in the array that is larger than R_k, --- O(log)

Then, as there are n query to answer, the second step will be run n times, which results in a time complexity of --- O(nlogn)

Thus, in total this algorithm gives an overall time complexity of O(nlogn).

2.

(a):

First step, using mergesort algorithm to sort the n element array S --- O(nlogn)

Second step, iterate over the array, for each element i, using binary search algorithm to try to find (x-i), if found, return true, otherwise return false --- O(nlogn)

This two step algorithm has an overall complexity of O(nlogn)

(b):

In this part, I think using hash table is a suitable way.

First step, construct the hast table takes n time steps ---O(n)

Because I need to iterate over the n integer array at least once.

Second step, iterate over the n integer array again, and this time check if the (x-i) exists in the hash table, here i is the element in the array. As searching in hash table requires---O(1)

And for n elements, the total complexity is O(n).

Thus, the overall complexity of using hash table is O(n).

3.

(a):

In the case that there's no celebrity and everyone knows only themselves.

First step, ask n-1 persons(except for ourselves) a question, then after n-1 questions, we are left with one person X who is potentially the celebrity. I am able to eliminate a person because If I ask A " Do you know B?", if A replies yes, then A is not the celebrity, moreover, if A says no, then B is not the celebrity.

Second step, after obtaining this potential celebrity X, I need to ask X, (n-1) questions to make sure that X does not know anyone else.

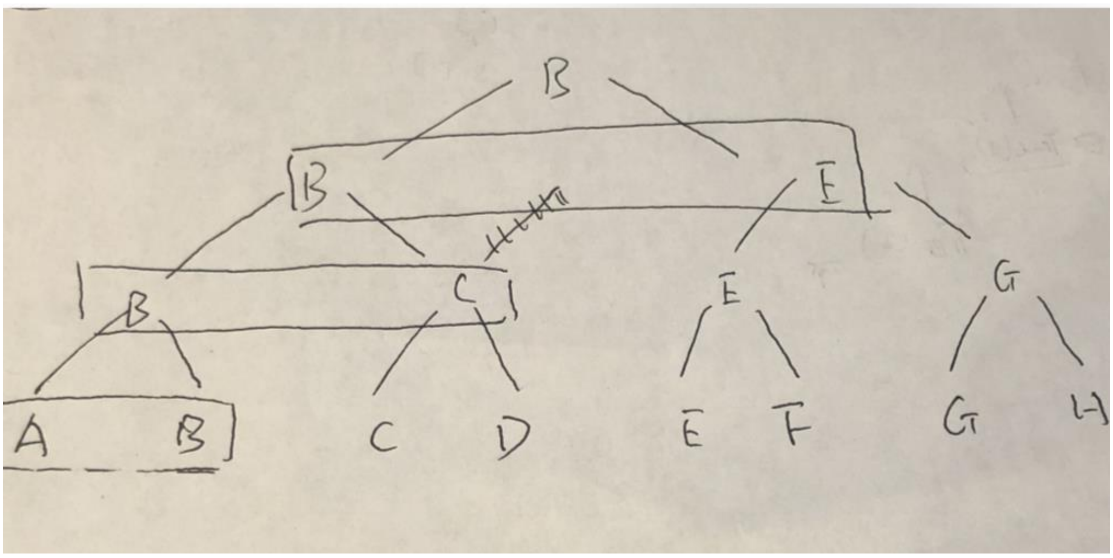
Third step, ask the rest (n-1) persons a question to make sure that they all know X.

All three step takes (n-1) questions, which results in 3(n-1) = 3n-3 questions in total.

(b):

If we consider the first step above as a tree, we have every single person as our leaf and we eliminate haft of the number of persons at each layer and finally we will obtain the potential celebrity X.

Suppose that we have 8 people at the party,A,B,C,D,E,F,G,H



As we can see from the graph, the number of questions that B, which is the potential celebrity, is equals to the height of the tree, which is log_n.

Thus, in the third step descripted above, we are able to save $\lfloor log_n \rfloor$ questions, and this will give us an even better result: 3n-3- $\lfloor log_n \rfloor$

4.

a): $f(n) = (log_2 n)^2$ $g(n) = log_2(n^{log_2 n}) + 2log_2 n$

f(n) and g(n) have the same asymptotic growth rate.

Firstly, $g(n) = log_2 n * log_2 n + 2log_2 n = (log_2 n)^2 + 2log_2 n$

here, we can see that $f(n) \le g(n)$ as:

$$(log_2 n)^2 \le (log_2 n)^2 + 2log_2 n, for\ all\ n \ge 1$$

Secondly, we can see that $c * g(n) \le f(n)$:

$$\frac{1}{2} * ((log_2 n)^2 + 2log_2 n) \le (log_2 n)^2 \ for\ all\ n \ge 4$$

In this case, there exist positive constant $c_1 = \frac{1}{2}, c_2 = 1$ such that

$$0 \le c_1 * g(n) \le f(n) \le c_2 * g(n) for\ all\ n \ge 4$$

Thus, $f(n) = \theta(g(n))$

b): $f(n) = n^{100}$ $g(n) = 2^{n/100}$

Since both f(n) and g(n) are monotonically increasing function, thus we can take the log of both f(n) and g(n):

$$\log(f(n)) = \log(n^{100}) = 100 * \log n$$

$$\log(g(n)) = \log\left(2^{\frac{n}{100}}\right) = \frac{\log(2)}{100} * n$$

From the above equations, we can see that when n tends to infinity, $\log(g(n))$ will grows faster than $\log(f(n))$ as n grows faster than $\log n$. Thus, $f(n) = O(g(n))$.

c): $f(n) = \sqrt{n}$ $g(n) = 2^{\sqrt{log_2 n}}$

As both f(n) and g(n) are monotonically increasing function, we can imply the same method from last question, Taking the log_2 of both function, we get:

$$\log(f(n)) = \sqrt{n} = \frac{1}{2} * log_2 n$$

$$\log(g(n)) = 2^{\sqrt{log_2 n}} = \sqrt{log_2 n} * log_2(2)$$

From the above equations, we can see that when n tends to infinity, $\log(f(n))$ will grows faster than $\log(g(n))$ as $log_2 n$ grows faster than $\sqrt{log_2 n}$. Thus, $f(n) = \Omega(g(n))$.

d): $f(n) = n^{1.001}$ $g(n) = n * log_2 n$

For this question, I use L'Hˆopital Rule:

$$\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \frac{1.001 * n^{0.001}}{log_2 n + \frac{1}{\ln(2)}} = \frac{\infty}{\infty}$$

Since the first derivative still results in $\frac{\infty}{\infty}$, I use L'Hˆopital Rule again:

$$\lim_{n\to\infty} \frac{f''(n)}{g''(n)} = \frac{1.001 * 0.001 * n^{-0.999}}{\frac{1}{n * \ln(2)}} = 1.001 * 0.001 * n^{0.001} * \ln(2) = \infty$$

This means that f(n) grows faster than g(n), thus, $f(n) = \Omega(g(n))$.

e): $\mathrm{f(n)} = n^{(1+\sin(\frac{\pi n}{2}))/2}$ $\mathrm{g(n)} = \sqrt{n}$

For this question, I know that $\sin\left(\frac{\pi n}{2}\right)$ ranges from [-1,1] which means that $\frac{1+\sin\left(\frac{\pi n}{2}\right)}{2}$ ranges from [0, 1] and $\mathrm{f(n)}$ ranges from [1,n], whereas $\mathrm{g(n)} = \sqrt{n} = n^{0.5}$, as f(n) in this case is not a monotonically increasing function, there doesn't exist any constant c1 that $c1 * g(n)$ is always smaller or equal to $\mathrm{f(n)}$, moreover, I can't think of any constant c2 that $c2 * g(n)$ is always larger than $\mathrm{f(n)}$

Thus, is this case, I think f(n) is not asymptotically comparable with g(n), in other words, all three relations(O, Ω, θ) don't satisfy.

5:

a):$T(n) = 2 * T\left(\frac{n}{2}\right) + n(2 + \sin n)$

In this question, based on Master Theorem:

$$a = 2, b = 2 \; and \; f(n) = n(2 + \sin n)$$

Let $g(n) = n^{log_b a} = n$, as $\sin n$ is in range [-1,1], f(n) ranges from [n,3n], for any constant $0 < c1 \leq 1$, $c1 * g(n) \leq f(n)$. Moreover, for any constant$3 0 \leq c2$, $f(n) \leq c2 * g(n)$, so we have proved $f(n) = \theta(g(n))$, which satisfy the second case in Master Theorem, then:

$$T(n) = \theta(nlog_2 n)$$

b): $T(n) = 2 * T\left(\frac{n}{2}\right) + \sqrt{n} + logn$

Based on Master Theorem:

$$a = 2, b = 2 \; and \; f(n) = \sqrt{n} + logn$$

Let $g(n) = n^{log_b a} = n$, for f(n), as n grows larger, $\sqrt{n}$ grows faster than $logn$ and finally $\sqrt{n} = n^{1/2}$dominates, thus, the first case of Master Theorem satisfies:

$$f(n) = O(n^{log_b a - \varepsilon}) = O(n^{1-\varepsilon}) \text{ ,then:}$$

$$T(n) = \theta(n)$$

c): $T(n) = 8 * T\left(\frac{n}{2}\right) + n^{logn}$

Based on Master Theorem:

$$a = 8, b = 2 \; and \; f(n) = n^{logn}$$

Let $g(n) = n^{log_b a} = n^3$, as logn is an increasing function, it will eventually pass 3, thus, the first two cases of Master Theorem do not satisfy, let's try to use case 3, firstly, f$(n) = n^{logn} = \Omega(n^{3+\varepsilon}) for \; some \; \varepsilon > 0$. Secondly, I try to find a constant c < 1 such that:

$$a * f\left(\frac{n}{b}\right) \leq c * f(n)$$

$$= 8 * f\left(\frac{n}{2}\right) \leq c * f(n)$$

$$= 8 * (n/2)^{\log(\frac{n}{2})} \leq c * n^{logn}$$

$$= 8 * \left(\frac{1}{2}\right)^{\log\left(\frac{n}{2}\right)} * (n)^{\log\left(\frac{n}{2}\right)} \leq c * n^{logn}$$

$$= 8 * \left(\frac{1}{2}\right)^{\log\left(\frac{n}{2}\right)} * \frac{1}{n^{log2}} * n^{\mathrm{logn}} \leq c * n^{logn}$$

Now, it can be seen that as long as $8 * \left(\frac{1}{2}\right)^{\log\left(\frac{n}{2}\right)} * \frac{1}{n^{log2}} \leq c < 1$ is true, the second condition is satisfied, it can be observed that as n grows larger, the numerator $\left(\frac{1}{2}\right)^{\log\left(\frac{n}{2}\right)}$ grows slower than the denominator n^{log2}, thus, $8 * \left(\frac{1}{2}\right)^{\log\left(\frac{n}{2}\right)} * \frac{1}{n^{log2}}$ will be smaller than 1. Since both conditions for case 3 are met,

$$T(n) = \theta(n^{logn})$$

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

Based on Master Theorem:

$$a = 1, b = 2 \; and \; f(n) = \; n$$

Let $g(n) = \; n^{log_b a} = n^{log_2 1} = 0$, and $f(n) = \; n$, in this case none of the conditions of Master Theorem is hold, so try to unwind the recurrence and add up the linear overheads:

$$T(n) = T(n-1) + n$$
$$T(n-1) = T(n-2) + n - 1$$
$$T(n-2) = T(n-3) + n - 2$$
$$T(n-3) = T(n-4) + n - 3$$
$$\dots$$

$$T(1) = T(0) + 1$$

Unwinding every term, it can be seen that:

$$T(n) = T(0) + \; n + n - 1 + n - 2 + n - 3 + \cdots + 1$$

Add up all the linear overheads:

$$sum \; of \; linear \; overheads = n + n - 1 + n - 2 + n - 3 + \cdots + 1$$
$$= \frac{(n+1) * n}{2} = \frac{n^2 + n}{2}$$

Then,

$$T(n) = T(0) + \; \frac{n^2 + n}{2}$$

$$T(n) = T(0) + \; \frac{n^2 + n}{2} \leq n^2, for \; n \; \geq 1$$

Thus, $T(n) = O(n^2) \;\; for \, n \; \geq 1$

Moreover,

$$\frac{1}{2} * n^2 \leq T(n) = T(0) + \; \frac{n^2 + n}{2}, for \; n > 0$$

So, $T(n) = \theta(n^2) \;\; for \, n > 0$