Assignment 1

z5142340 Haiyu LYU

Question1:

① Sort the array A by using Merge Sort. $O(n\log n)$

② Use binary search to find L_k. $O(\log n)$

③ Use the same algorithm as above to find R_k. $O(\log n)$

④ Each binary search takes $O(\log n)$, so a series of n queries take $O(n\log n)$

⑤ Therefore, this algorithm runs in $O(n\log n)$overall.

Question2:

(a).

① First, let us sort the array A by using Merge Sort. $O(n\log n)$

② Let A_s be the smallest element and A_l be the largest element.
If A_s+A_l > x, largest number can not be the one of solution.
If A_s+A_l < x, smallest number can not be the one of solution.
If A_s+A_l = x, we can get the solution.
So, if A_s or A_l is not the one of solution, we can remove that element in array A.

③ Let us repeat the ② algorithm, after at most (n-1) iterations, we can get the solution or find that there is no such elements that satisfies the condition.

④ In summary, we can do this algorithm in $O(n\log n)$

(b).

① The data structure that allows an almost instantaneous search is Hash Table.

② Inserting all integers and building lookup will take $O(n)$.

③ We loop through array A. For each element A[i], we check if there is a value that satisfies x-A[i] in Hash table.

④ This algorithm runs in $O(n)$ overall.

Question3:

(a).

① We choose the first person as candidate. Ask everyone in turn E (2 to n) ‘candidate know you?’
If you know, then the candidate can't be a celebrity, E becomes the new candidate.
If you don’t know, then the candidate will continue to maintain the status of the candidate.
After asking $(n-1)$ times, the candidate may be a celebrity but cannot be completely sure.

② We need to consider the situation where the candidate is not a celebrity
Ask everyone (except the candidate) 'Do you know the candidate?'
If he knows, then he still maintains the status of a candidate.
If he doesn’t know, then there is no celebrity.
This may take $(n-1)$ steps.

③ In order to avoid that the candidate know others and others also know the candidate.
Ask everyone (except the candidate) ‘candidate knows you’
If answers are all no, then the candidate is finally identified as a celebrity.
Otherwise, there are no celebrities.
Finally ask $(n-1)$ times.

This algorithm will ask $3n-3$ questions.

(b).

If we can use the binary search for **(a)**, we may find that there are some questions between ① and ③ that are repeated.

At ①, we can treat two people as a pair in this method and ask them to answer the following questions. For example, we asked the first two people 1 and 2. Firstly ask if 1 to know 2. If 1 knows 2, then 1 is removed. On the other hand, if 1 does not know 2, 2 is removed. Then ask 3 and 4 in the same question. Finally, we will always ask the last person.
The purpose of this is to avoid $\lfloor log_2 n \rfloor$ questions that are duplicated.
Therefore, we just need ask $3n-3-\lfloor log_2 n \rfloor$

Question4:

(a).

$f(n) = (log_2 n)^2 \qquad g(n) = log_2(n^{log_2 n}) + 2\, log_2 n$

$g(n) = log_2 n \bullet log_2 n + 2\, log_2 n = (log_2 n)^2 + 2\, log_2 n$

$f(n) = \Theta(g(n))$

(b).

$f(n) = n^{100} \qquad g(n) = 2^{n/100}$

$log_2\big(f(n)\big) = 100\, log_2 n$

$log_2\big(g(n)\big) = \frac{n}{100}\, log_2 2 = \frac{n}{100}$

$f(n) = O(g(n))$

(c).

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

$log_2(f(n)^2) = log_2 n$

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

$f(n) = \Omega(g(n))$

(d).

$f(n) = n^{1.001} \qquad g(n) = n\, log_2 n$

$a = 1.001$

$S(n) = \frac{n^{a-1}}{log_2 n}$

We can computer the derivative $S(n)'$, and this is increasing. ($n > e^{1/a-1}$)

$n^{a-1} > log_2 n$

$n \bullet n^{a-1} > log_2 n$

$O(n^a) > O(n\, log_2 n)$

$f(n) = \Omega(g(n))$

(e).

$f(n) = n^{(1+sin(\pi n/2))/2} \qquad g(n) = \sqrt{n}$

$log(f(n)) = (1 + sin(\pi n/2))/2$

$log(g(n)) = \frac{1}{2}$

$sin(\pi n/2)$ has periodicity, so we can not determine the relationship between $f(n)$ and $g(n)$

Question5:

(a).

$T(n) = 2T\left(\frac{n}{2}\right) + n(2 + sin\,n)$

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

$n^{log_b a} = n^{log_2 2} = n$

Because $1 \leq 2 + sin\,n \leq 3$, we can see $2 + sin\,n$ as constant c.

For Master Theorem case 2:

$f(n) = cn = \Theta(n^{log_2 2})$

$T(n) = \Theta(n^{log_2 2}\,log_2 n) = \Theta(n\,log_2 n)$

(b).

$T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n} + log\,n$

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

$n^{log_b a} = n^{log_2 2} = n$

$\sqrt{n}$ is increase a lot more than $log\,n$, because $log\,n = 2\,log\,\sqrt{n}$.

So, the complexity of $f(n)$ is $O\left(n^{\frac{1}{2}}\right)$.

For Master Theorem case 1:

$f(n) = O\left(n^{\frac{1}{2}}\right) = O(n^{1-\varepsilon})$

$T(n) = \Theta(n^{log_2 2}) = \Theta(n)$

(c).

$T(n) = 8T\left(\frac{n}{2}\right) + nlog\,n$

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

$n^{log_b a} = n^{log_2 8} = n^3$

For Master Theorem case 3:

$a\,f\,(n/b) \leq c\,f(n)$

$8\,f(n/2) \leq cf(n)$

$8(\frac{n}{2})^{logn/2} \leq cf(n)$

$$\frac{1}{2^{log\frac{n}{2}-3}} \bullet n^{log\,n - log\,2} \leq cn^{log\,n}$$

$$\frac{1}{2^{log\frac{n}{2}-3}} \leq c < 1$$

Therefore, $T(n) = \Theta(f(n)) = \Theta(n^{log\,n})$

(d).

By mathematical induction, we can see clearly that:

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

$T(n-1) = T(n-2) + n - 1$

$\dots\dots$

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

It looks like arithmetic sequence, and the common difference is d=1.

$$T(n) = S(n) = \frac{n(n-1)}{2} = O(n^2)$$