Name: PEIGUO GUAN

zID: z5143964

1. [20 marks] You're given an array A of n integers, and must answer a series of n queries, each of the form: "How many elements *a* of the array A satisfy $L_k \le a \le R_k$?", where L_k and R_k $(1 \le k \le n)$ are some integers such that $L_k \le R_k$. Design an O(nlogn) algorithm that answers all of these queries.

Answer:
Firstly, the array is not in order, so sorting the array by Merge Sort whose time complexity is *O(nlogn)*.

Then by using loop, count $a = 0$, and we can compare the element and *a* add 1 when finding the first element in array A which is larger than or equal to the L_k, and return the element which is larger than R_k.

```
A = MergeSort(A)
count = 0
i=0
for i-> n:
  if A[i]>= Lk:
  a+=1
  if A[i]> Rk:
  return a
```

2. [20 marks, both (a) and (b) 10 marks each] You are given an array S of n integers and another integer x.
(a) Describe an O(nlogn) algorithm (in the sense of the worst case performance) that de-termines whether or not there exist two elements in S whose sum is exactly x.
(b) Describe an algorithm that accomplishes the same task, but runs in O(n) expected (i.e., average) time. Note that brute force does not work here, because it runs in $O(n^2)$ time.

Answer:
(a):At first, sorting the array S by using Merge Sort and the time complexity in the worst case is *O(nlogn)*.

Assume two elements whose sum could be x and set them as Head of array S (*h*) and Tail of array S (*t*). Since the array is sorted, it is easy to find that, if *h*+*t* is less than x, *h* should be S[*h*+1], on the contrary, if *h*+*t* is larger than x, t should be S[*t*-1] so as to close to the input number x. If none of this two situation, we can return the value *h* and

t, cause their sum equal to x. In worst case every element are searched once and the time complexity of the progress is *O(n)*.

So in worst case, the time complexity is *O(nlogn)*.

```
S = MergeSort(S)
i=0
j= length(S)-1
h=S[i]
t=S[j]
while h<t:
  if h+t <x:
      i +=1
      h=S[i]
  if h+t >x:
      j -= j
      t=S[j]
  else:
      return h,t
```

(b):By using Hash table to check if the element is existed in the array. Hash all the elements, and then go through all the element(assume *i*) in S, check if $x - i$ is in the table which cost O(1), and if there are two same element sum up to x like $2 * i = x$ then, check if there are two *i* in hash table.

In total, every element will go through once and the time complexity is O(n).

3.[20 marks, both (a) and (b) 10 marks each; if you solve (b) you do not have to solve (a)] You are at a party attended by n people (not including yourself), and you suspect that there might be a celebrity present. A celebrity is someone known by everyone, but does not know anyone except themselves. You may assume everyone knows themselves. Your task is to work out if there is a celebrity present, and if so, which of the n people present is a celebrity. To do so, you can ask a person X if they know another person Y (where you choose X and Y when asking the question).

(a) Show that your task can always be accomplished by asking no more than $3n - 3$ such questions, even in the worst case.

(b) Show that your task can always be accomplished by asking no more than $3n - \lfloor \log_2 n \rfloor - 2$ such questions, even in the worst case.

Answer:

(a) Suppose, numbered these n people from 1…n. There is one celebrity in the party.

Suggest we pick one people from these *n* people as *a*, the rest people are n-1 people, at this time, set *i* as people from 2…n except the first person *a*.

Then check that, if *a* knows person *i,* if a does then a cannot be a celebrity cause celebrity does not know anyone, in this situation, *i* could be celebrity and let *a = i,*

i=i+1. Else if *a* does not know person *i*, *a* still could be a celebrity, and let *i=i+1*.

After asking n-1 questions, we can come out that *a* is the person most likely to be a celebrity.

However, to verify as a celebrity has 2 requirements:

1) Celebrity is known by everyone. 2) Celebrity does not know anyone.

So, It is possible that *a* is not a celebrity. We need to check by asking n-1 questions (*i for i* from people 2-n) that "does *i* know *a*?" if all the answer is yes, then *a* could be a celebrity(requirement 1 is fit) otherwise, *a* is not a celebrity.

Then ask n-1 questions for "does *a* know *i?*" if all the answer is no, the we can conclude that *a* is celebrity(requirement 2 is fit), otherwise there is not celebrity.

In total: The total questions is *n-1+n-1+n-1 = 3n-3* even in the worst case.

(b) In this question, we can construct a full binary tree to find the celebrity. At first, there are n number of people, and construct a perfect binary tree with the depth of d=$\lfloor \log_2 n \rfloor$ with n leaves, and if $2^d < n$, add two children to the leftmost $n - 2^d$ leaves of perfect binary tree. It cost n-1 to find the potential celebrity like in (a), and to verify the celebrity, $\lfloor \log_2 n \rfloor$ questions has been answered in previous steps(the left child knows the right child as in (a) and closer to the root), which is the one on each level.
So, from the questions (a) we can get the total questions are *3n-3*, but in we don't need to ask $\lfloor \log_2 n \rfloor$ more questions which is *3n-3* - $\lfloor \log_2 n \rfloor$ in the end, and it is smaller than the 3n−$\lfloor \log_2 n \rfloor$−2 as mentioned in the worst case.

4. [20 marks, each pair 4 marks] Read the review material from the class website on asymptotic notation and basic properties of logarithms, pages 38-44 and then determine if f(n) = Ω(g(n)), f(n) = O(g(n)) or f(n) = Θ(g(n)) for the following pairs. Justify your answers.

$f(n)$	$g(n)$
$(\log_2 n)^2$	$\log_2(n^{\log_2 n}) + 2\log_2 n$
n^{100}	$2^{n/100}$
$\sqrt{n}$	$2^{\sqrt{\log_2 n}}$
$n^{1.001}$	$n \log_2 n$
$n^{(1+\sin(\pi n/2))/2}$	$\sqrt{n}$

You might find the following inequality useful: if f(n), g(n), c > 0 then f(n) < c g(n) if and only if log f(n) < log c + log g(n)

Answer:

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

let q = $\log_2 n$, so n = $2^q, f(q) = q^2$

$g(n) = \log_2 n^{\log_2 n} + 2\log_2 n = \log_2 n * \log_2 n + 2\log_2 n = q^2 + q$

The domain term is both a square, so $f(n) = \Theta(g(n))$

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

$when \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^{100}}{2^{n/100}} = 0, there\ is\ a\ time\ g(n)\ will\ grow\ faster\ than\ f(n)$

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

(3): $f(n) = n^{\frac{1}{2}}, g(n) = 2^{\sqrt{\log_2 n}} = 2^{(\log_2 n)^{\frac{1}{2}}}$

Let $(\frac{f(n)}{g(n)})^{(\log_2 n)^{\frac{1}{2}}} = \frac{n^{\frac{1}{2}(\log_2 n)^{\frac{1}{2}}}}{2^{(\log_2 n)}} = \frac{n^{\frac{1}{2}(\log_2 n)^{\frac{1}{2}}}}{n}$

There exist a $n_0$that will $\frac{1}{2}(\log_2 n_0)^{\frac{1}{2}}$ >= 1

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

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

let q = $\log_2 n$, so n = $2^q, f(q) = (2^q)^{1.001}, g(n) = 2^q * q$

$$\lim_{n\to\infty} \frac{f(q)}{g(q)} = \frac{(2^q)^{1.001}}{2^q * q} = \frac{(2^q)^{0.001}}{q} = 0$$

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

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

$\frac{1+\sin\left(\frac{\pi n}{2}\right)}{2} \in [0,1]$

So $f(n) = \theta(g(n))$

5. [20 marks, each recurrence 5 marks] Determine the asymptotic growth rate of the solutions to the following recurrences. If possible, you can use the Master Theorem, if not, find another way of solving it.

(a) $T(n) = 2T(n/2) + n(2 + \sin n)$
(b) $T(n) = 2T(n/2) + \sqrt{n} + \log n$
(c) $T(n) = 8T(n/2) + n^{\log n}$
(d) $T(n) = T(n-1) + n$

Answer:

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

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

$n^{\log_b a} = \Theta(n) = f(n)$

$T(n) = \Theta(n \log n)$

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

$a = 1, b = 2,$ let $f(n) = \sqrt{n} + \log n$

$n^{\log_b a} = \Theta(n) > f(n) = n^{\frac{1}{2}}$, when $\varepsilon = ½$ then $f(n) = n^{\log_a b - 1/2} = n^{\frac{1}{2}}$

So, $T(n) = \Theta(n)$

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

$\alpha = 3, \beta = \log n$, a =8, b=2, $f(n) = n^{\log n}$

$f(n) = n^{\log n} \neq O(n^{\log_b a - \varepsilon})$

$f(n) = n^{\log n} \neq \Theta(n^{\log_b a})$

There exist ε $f(n) = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{3+\varepsilon})$

And c<1 , n is as large as possible, there exist $af\left(\frac{n}{b}\right) = 8f\left(\frac{n}{2}\right) = 8 * \frac{n^{\log n - \log 2}}{2}$,

$cf(n) = cn^{\log n}$, $8 * \frac{n^{\log n}}{2} \leq cn^{\log n} * \frac{n^{\log 2}}{2}$

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

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

a=1,b=1 and b not larger than 1 so master theorem cannot work.

$T(n) = T(0) + 1 + 2 + 3 + \cdots n = T(0) + \frac{n(n+1)}{2}$

So T(n)=O(n^2)