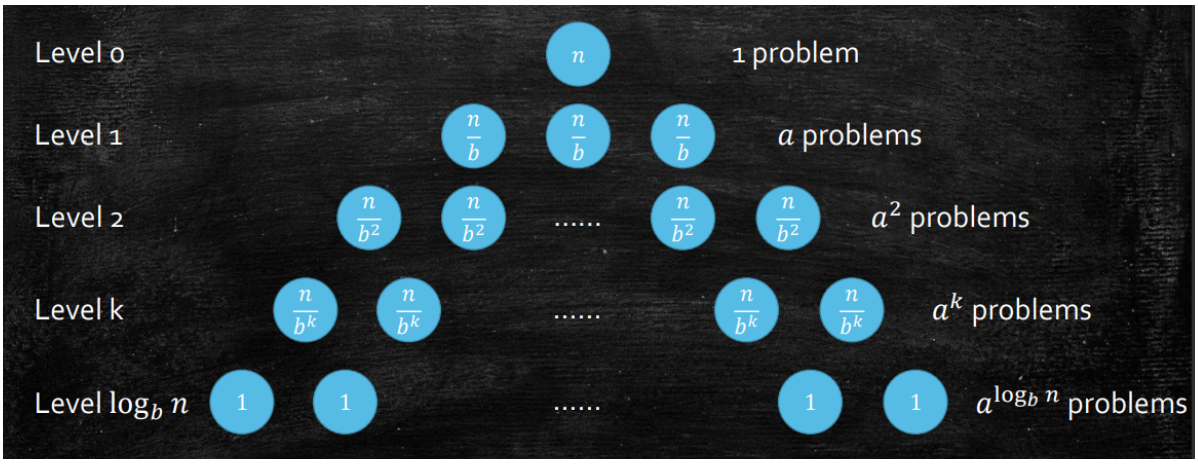
Assignment 1

王凯灵 521030910356

Problem 1

According to the recursion tree below, there are $a^{log_b n}$ unit problems.



ppt-rip

The total runtime of all unit problems is $O(n^{log_b a})$.

The total runtime is

$$\begin{aligned} & O(n^d \log^w n) + a \cdot O\left(\left(\frac{n}{b}\right)^d \log^w \frac{n}{b}\right) + \cdots + a^k \cdot O\left(\left(\frac{n}{b^k}\right)^d \log^w \frac{n}{b^k}\right) + \cdots + O(n^{\log_b a}) \\ & = O(n^d) \cdot O(\log^w n + \frac{a}{b^d} \log^w \frac{n}{b} + \cdots + \left(\frac{a}{b^d}\right)^k \log^w \frac{n}{b^k} + \cdots + \left(\frac{a}{b^d}\right)^{\log_b n} \log^w \frac{n}{b^{\log_b n}}) \end{aligned} \tag{1}$$

Use $[\log_b n] + 1$ instead if $\log_b n$ is not an integer.

Note that $log^w n < log^w \frac{n}{b^k}$ when $b > 1$. We have total runtime less than

$$O(n^d \log^w n) \cdot O\left(\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i\right) \tag{2}$$

- ### for case $a < b^d$

  We have $\frac{a}{b^d} < 1$. $\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i$ is $O(1)$. So $T(n) = O(n^d log^w n)$

- ### for case $a = b^d$

  We have $\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i = log_b n + 1$ is $O(\log_b n)$. The total runtime $O(n^d \log^{w+1} n)$

- ### for case $a > b^d$

  Reorganize equation 1 as follows:

$$O\left(\sum_{k=0}^{\log_b n} a^k \cdot \left(\frac{n}{b^k}\right)^d \left(\log \frac{n}{b^k}\right)^w\right) \tag{3}$$

It is not yet an increasing sequence. So we construct $\{c_k\}$, $c_0 = \log^w n$

$$c_{k+1} = \begin{cases} \log^w \frac{n}{b^{k+1}} & \text{if } \frac{c_k}{\log^w \frac{n}{b^{k+1}}} < \frac{a}{b^d} \\ c_k \cdot \frac{b^d}{a} & \textit{otherwise} \end{cases} \tag{4}$$

Replace the log part with $\{c_k\}$ in 3. This ensures monotonicity. The total runtime is now decided by the last item in 3.

$$\begin{aligned} O\left(\sum_{k=0}^{\log_b n} a^k \cdot \left(\frac{n}{b^k}\right)^d \left(\log \frac{n}{b^k}\right)^w\right) &= O\left(a^{\log_b n} \cdot \left(\frac{n}{b^{\log_b n}}\right)^d \left(\log \frac{n}{b^{\log_b n}}\right)^w\right) \\ &\le O\left(a^{\log_b n}\right) \cdot O\left(\log^w n \cdot \left(\frac{b^d}{a}\right)^{\log_b n}\right) \\ &= O\left(a^{\log_b n}\right) \cdot O\left(\log^w n \cdot \frac{n^d}{n^{\log_b a}}\right) \\ &= O\left(a^{\log_b n}\right) \cdot O\left(\log^w n \cdot n^{d - \log_b a}\right) \end{aligned} \tag{5}$$

Since $a > b^d$, $d - \log_b a < 1$, the result of 5 is les than $O(n^{\log_b a})$

Conclusion

$$T(n) = \begin{cases} O\left(n^d \log^w n\right) & \text{if } a < b^d \\ O\left(n^{\log_b a}\right) & \text{if } a > b^d \\ O\left(n^d \log^{w+1} n\right) & \text{if } a = b^d \end{cases} \tag{6}$$

Problem 2

Obviously if we merge these k arrays two by two or use k pointer algorithm, the time complexity will be $O(k^2 \cdot n)$. To use divide and conquer, the simplest way is to just divide the k arrays into two groups until there are only two arrays. The algorithm is shown below.

Algorithm 1 k-way merge

Input: k sorted arrays eith n elements each
Output: one single sorted array
  if there are only two arrays left **then**
    merge these two arrays using double pointer
    return merged array
  end if
  apply the algorithm on two $\frac{k}{2}$ sets of arrays
  merge the returned two arrays into one **return** this single array

The correctness is obvious. To merge two arrays, the time complexity is $O(n + m)$, and we have $T(n) = 2T(\frac{n}{2}) + O(n)$. Apply the mast theorem, we find $a = b = 2, d = 1$. Here $a = b^d, T(n) = O(kn \log k)$.

Problem 3

It's easy to realize that simply combining the two arrays will cause $O(n)$ complexity. To do better than $O(n)$ using divide and conquer, it's easy to relate to binary search, based on which I give the following algorithm. Actually, to find the medium is the same as to find the $(m+n+1)/2th$ largest in the two arrays.

Algorithm 2 two-array medium

Input: sorted arrays A, B (1-based) with length n and m
Output: the medium of the two arrays
  $i \leftarrow (m+n+1)/4$
  $a_{left}, b_{left} \leftarrow 0$
  while $i > 1$ **do**
    if $A[a_{left}+i] < B[b_{left}+i]$ **then**
      $a_{left}+=i+1$
    else
      $b_{left}+=i+1$
    end if
    $i \leftarrow (n+m-a_{left}-b_{left}+1)/4$
  end while
  if $m+n$ is odd **then**
    return $min(A[a_{left}], B[b_{left}])$
  else
    return the average of the smaller 2 of A[i], A[i+1], B[i] and B[i+1]
  end if

Explanation:

I'm not so sure whether this is also divide and conquer. To find the $k=(m+n+1)/2th$ largest, we compare $A[i]$ and $B[i]$, $i=(m+n+1)/4.$ Suggest $A[i]$ is smaller, then $A[1]$ to $A[i]$ couldn't be the $k\ th$ largest. delete them and continue to find the $k/2th$ largest in the new A and B. In the end, calculate the medium according to the parity of $m+n$.

Time complexity:

$T(n)=T\left(\frac{n}{2}\right)+O(1)$, apply the master theorem to find $T(n)=\log(n)$, better than $O(n)$.

Problem 4

- ## (a)

  I don't really understand the meaning of the existence of problem (a). Since we can write $y-x=z-y$ into $x+z=2\cdot y$, if the parity of x and z are different, $x+z$ will be odd. Then y mustn't be an integer.

- **(b)**

  I use $\{3,1,4,2\}$ as an example. All of the triples are: $(3,1,4),(3,1,2),(3,4,2),(1,4,2)$, none of which forms a good order.

- **(c)**

  From problem (a), we can see that only if means necessary and insufficient. So I only need to prove (x,y,z) forms a good order $\rightarrow\left(\frac{x+1}{2},\frac{y+1}{2},\frac{z+1}{2}\right)$ forms a good order.

  We have $x+z=2\cdot y$, and so

  $$\frac{x+1}{2}+\frac{z+1}{2}=\frac{x+z}{2}+1=y+1=2\cdot\frac{y+1}{2} \tag{7}$$

  I guess I have proved it.

- **(d)**

  First divide the numbers by parity, because triple formed by numbers across the groups mustn't from a good order, for x_i and x_k have different parity.

  Let's define a function:

  $$fold(x)=\begin{cases}\frac{x+1}{2} & \text{if } x \text{ is odd}\\ \frac{x}{2} & \text{if } x \text{ is even}\end{cases} \tag{8}$$

  For the odd numbers, apply the conclusion from problem (c), $\{fold(x_n)\}$ is out-of order means $\{x_n\}$ is out-of -order. For the even numbers, as (x,y,z) forms a good order $\rightarrow$ $\left(\frac{x}{2},\frac{y}{2},\frac{z}{2}\right)$ forms a good order, the process is the same. What's more, the results of the two groups after applying the fold function is the same. They can share the sequence.

  if n is odd, we can add 1 to n to make it even, because if we finish the process and remove this $n+1$ later, the result is still out-of-order. This ensures the correctness.

  Now I can write the algorithm. The function $fold(x)$ is the same as 8. And the operation $arrange\ by\ index$ means, if we arrange a,b,c,d by $1,3,2,4$, the result is a,c,b,d. The function $cat(a,b)$ means directly link two arrays.

Algorithm 3 Construct out-of-range

Input: n
Output: an out-of-order permutation
  if $n=1$ **then**
    return 1
  end if
  divide odd and even: $\{1,3\cdots 2,4\cdots\}$
  fold the odd part: $A:\{1,3\cdots k\}\rightarrow B:\{1,2\cdots\frac{k+1}{2}\}$
  apply this algorithm on B to get B_i
  arrange A by B_i to form A_i
  return $cat(A_i, A_i+1)$

Let's use 10 as an example. First we get $\{1,3,5,7,9,2,4,6,8,10\}$, and $B_1 = \{1,2,3,4,5\}$. Similarly, $B_2 = \{1,2,3\}$, $B_3 = \{1,2\}$. Obviously $B_3 = B_{3i}$. Arrange $A_3 = \{1,3\}$ by B_{3i} and return $B_{2i} = \{1,3,2\}$. Arrange $A_2 = \{1,3,5\}$ by B_{2i} and return $B_{1i} = \{1,5,3,2,4\}$. The final result is thus $\{1,9,5,3,7,2,10,6,4,8\}$.

This algorithm is $O(n)$ because $T(n) = T(\frac{n}{2}) + O(n)$.

What about the $O(n \log n)$ version? Maybe just to deal with the even number group the same as the odd group. This way, $T(n) = 2T(\frac{n}{2}) + O(n)$

Problem 5

- **(a)**

Algorithm 4 Even-Paz algorithm

Input: a cake (initially[0,1]), num of *child* n and their value density function f_i
Output: an allocation $I : \int_{I_i} f_i(x)dx \geq \frac{1}{n}\int_0^1 f_i(x)dx$
  suggest the current cake is $[a, b]$
  if $n = 1$ **then**
    Give the cake to this *child*
  end if
  for each $child_i$ **do**
    calculate the half-half point x_i: ▷ $O(n)$
    $\int_0^{x_i} f_i(x)dx = \frac{1}{n}\left\lfloor \frac{n}{2} \right\rfloor \cdot \int_0^1 f_i(x)dx$
    use medium algorithm to find the medium x_i^* ▷ $O(n)$
  end for
  select the $\frac{n}{2}th$ x_i to divide the cake into two parts
  apply this algorithm: the former $\frac{n}{2}$ children shares the cake $[a, x_i^*]$ ▷ recursion
  apply this algorithm: the children left shares the rest of the cake ▷ recursion

We can use the top-k algorithm as the medium algorithm here.

- **(b)**

$T(n) = 2T(\frac{n}{2}) + O(n)$

Each time, the cake is cut by half and is divided by half of the children. Each round, we need to calculate the x_i for each child, which is $O(n)$. The time complexity is $O(n \log n)$, according to the result of Problem 1.

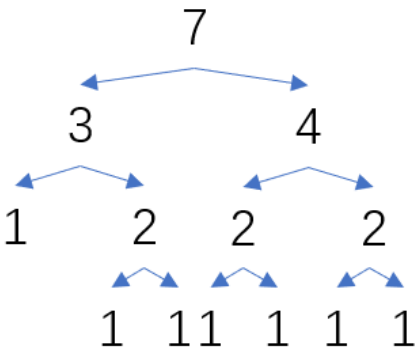
- **(c)**

To simplify the problem, let $\int_0^1 f_i(x)dx = 1$.

Each time we divide the cake into two, each child in the former group shares no less than $\frac{1}{n}\left\lfloor \frac{n}{2} \right\rfloor$ of the cake to be divided. If n is even, this value is exactly $\frac{1}{2}$. If n is odd, this value is $\frac{n-1}{2n}$. For children in the latter group, they each share more than $\frac{1}{2}$.

The most easy case is $n = 2^k$, and each time $\frac{n}{2}$ kids shares $\frac{1}{2}$ cake. In stage $k = \log n$, each child share at least $\frac{1}{2^k}$ of the cake, which is $\frac{1}{n}$. For cases $n \neq 2^k$ are discussed below.

There are $\lceil \log n \rceil$ stages in all. Of all the children, $2\left(n - 2^{\lfloor \log n \rfloor}\right)$ end at stage $\lceil \log n \rceil$ while the rest end at stage $\lfloor \log n \rfloor$. Since the children in the latter group tend to be more, they are more likely to be end at stage $\lceil \log n \rceil$. Here is a specific example.



We can see that if a child is always in the former group, he\she(\other gender) must be in the $\lfloor \log n \rfloor$ stage, and thus shares at least $\frac{n-1}{2n} \cdot \frac{\frac{n-1}{2}-1}{n-1} \cdots \frac{1}{3} = \frac{1}{n}$. On the $\lceil \log n \rceil$ stage, the difference is to replace a $\frac{k-1}{2k}$ with $\frac{k+1}{2k}$ and multiply $\frac{1}{2}$. But the k is hard to decide. It's better to look from the button.

Then I realized the branches are caused by 3. If three person shares cake $[a, b]$, they end up get at least $\frac{1}{3}$ each. We just have to see the situations on the $\lceil \log n \rceil - 2$ stage, on which there are only 3 or 4. Select a route in the tree with 3 on it: $n - a - b - c - d \cdots - 3$. The 3 gets $\frac{a}{n} \cdot \frac{b}{a} \cdots \frac{3}{d} = \frac{3}{n}$. Similarly, the 4 gets $\frac{4}{n}$. So each child gets at least $\frac{1}{n}$.

Problem 6

I didn't know about the time. I didn't finish it at one time. I'll give 3 for the difficulty. Level 4 should cost me several days and level 5 should only be solved only by gods like xxyQwQ and fstqwq. For now, I have no collaborators.

I expect an example format for pseudo-code.

Added collaborators after reviewing: [Ariel Procaccia]