Problem 1 (20 Points)

Prove the following generalization of the master theorem. Given constants $a \geq 1, b > 1, d \geq 0$, and $w \geq 0$, if $T(n) = 1$ for $n < b$ and $T(n) = aT(n/b) + n^d \log^w n$, we have

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

Solution. By the recurrence relation, we have

$$\begin{aligned} T(n) &= aT(n/b) + n^d \log^w n \\ &= a^2 T(n/b^2) + n^d \log^w n + a(n/b)^d \log^w (n/b) \\ &= a^3 T(n/b^3) + n^d \log^w n + a(n/b)^d \log^w (n/b) + a^2 (n/b^2)^d \log^w (n/b^2) \\ &\quad \vdots \\ &= n^d \log^w n + a(n/b)^d \log^w (n/b) + a^2 (n/b^2)^d \log^w (n/b^2) + \\ &\qquad \cdots + a^{\log_b n} (n/b^{\log_b n})^d \log^w (n/b^{\log_b n}) \\ &< n^d \log^w n (1 + (a/b^d) + (a/b^d)^2 + \cdots + (a/b^d)^{\log_b n}) \end{aligned}$$

If $a < b^d$, then we have $a/b^d < 1$. Thus,

$$T(n) < n^d \log^w n \frac{1 - (a/b^d)^{\log_b n}}{1 - a/b^d} = O(n^d \log^w n).$$

If $a > b^d$, choose $\varepsilon > 0$ such that $\log_b a > d + \varepsilon$. Since $n^d \log^w n = O(n^{d+\varepsilon})$, applying the original master theorem on the following recurrence will yield $T(n) = O(n^{\log_b a})$:

$$U(n) = aU(n/b) + O(n^{d+\varepsilon}).$$

If $a = b^d$, we have $e/b^d = 1$. Therefore,

$$T(n) < n^d \log^w n \cdot \log_b n = O(n^d \log^{w+1} n).$$

This solution is provided by ***Prof. Biaoshuai Tao****.* □

Problem 2 (15 points)

A k-way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements. Design an efficient algorithm using divide-and-conquer (and give its time complexity).

Solution. The algorithm is given in Algorithm 1.

Algorithm 1 One-Third-Merge-Sort

```
$k$-MERGEOPERATION($\{a_1^{(1)}, \dots, a_n^{(1)}\}, \dots, \{a_1^{(k)}, \dots, a_n^{(k)}\}$)
1: if $n \le 2$ then
2:     Merge the two arrays and get $a'$
3:     return a'
4: Divide $k$ arrays into 2 groups $a^{(1)}, a^{(2)}, \dots, a^{\left(\frac{k}{2}\right)}$ and $a^{\left(\frac{k}{2}+1\right)}, a^{\left(\frac{k}{2}+2\right)}, \dots, a^{(k)}$
5: Recursively merge $\frac{k}{2}$ arrays $a^{(1)}, a^{(2)}, \dots, a^{\left(\frac{k}{2}\right)}$ into $u$
6: Recursively merge $\frac{k}{2}$ arrays $a^{\left(\frac{k}{2}+1\right)}, a^{\left(\frac{k}{2}+2\right)}, \dots, a^{(k)}$ into $v$
7: Merge $u$ and $v$ into $a'$
8: return $a'$
```

Correctness analysis: We prove it by induction.

- Case 1: $n = 1$. Clearly, the minimum element of the array is in one of the k arrays, and apparently, no matter which array it in, it is the minimum element as well. then our algorithm will collect it.
- Case 2: $n = k$. When we collect the k-th-smallest element, the $(k+1)$-th element is whether in the same array with k-th or not. if it is the former condition, when the pointer moves our algorithm will find it. While in the latter case, we believe the element will be the minimum element in that array, which our pointer pointed. Otherwise there exists an element smaller than the $(k+1)$-th while bigger than the k-th, which is no possible.

Time complexity: in every recursion, the problem is divided into 2 sub-problems, and each has half size of the orignal problem. In addition, an $O(kn)$ cost is needed for merging two arrays. Then we have a recursive relationship $T(k) = 2T(\frac{k}{2}) + O(kn)$. Just expand the expression as follow

$$\begin{aligned} T(k) &= O(kn) + 2O\left(\frac{kn}{2}\right) + \cdots + 2^{\log_2 k} O\left(\frac{kn}{2^{\log_2 k}}\right) \\ &= (1 + \log_2 k)\, O(kn) \\ &= O\left(nk \log k\right) \end{aligned}$$

Therefore, the time complexity of this algorithm is $O(nk \log k)$. It is much more efficient than simply merging arrays one-by-one, whose time complexity is $O(k^2 n)$. The improvement comes from the equal division, which makes sub-problems at the same level have the same scale.

This solution is provided by ***T.A. Panfeng Liu*** *and* ***Stu. Xiangyuan Xue****.* □

Problem 3 (15 points)

Given two sorted lists $a_1, a_2, ..., a_n$ and $b_1, b_2, ..., b_m$ with $n < m$. Design a divide and conquer algorithm to find the median of the union of the two lists. Your algorithm should have running time better than $O(m)$.

Solution. Without loss of generality, we can assume $n + m$ is odd. The case when $n + m$ is even is similar. Indeed, the median of the union list is just the $\left(\frac{n+m+1}{2}\right)$-smallest number. Here we give a general algorithm to find the ℓ-smallest number of the union of the two lists, where $1 \leq \ell \in \mathbb{N}^+ \leq n + m$.

Suppose ℓ is odd (the case when ℓ is even is similar) and $n \leq m$. If $\frac{\ell+1}{2} > n$, then we let $p = \lceil \frac{n}{2} \rceil$ and $q = \ell+1-p$. Otherwise, let $p = \frac{\ell+1}{2}$ and $q = \frac{\ell+1}{2}$. Hence, we always have $p + q = \ell + 1$. Consider the comparison between a_p and b_q,

- If $a_p = b_q$, then the median is exactly this number.
- If $a_p > b_q$, it means that the numbers in $a_p, \ldots, a_n$ are at least $(\ell+1)$-smallest and can not be median of the union list, and the numbers in $b_1, \ldots, b_{q-1}$ are at most $(\ell - 1)$-smallest and also can not be median of the union list. We can remove the two parts and continue to find the $(\ell - q + 1)$-smallest number among $a_1, \ldots, a_{p-1}, b_q, \ldots, b_m$.
- If $a_p < b_q$, then the median is exactly b_q if $n = 1$. Otherwise, it means that the numbers in $a_1, \ldots, a_{p-1}$ are at most $(\ell - 1)$-smallest and can not be median of the union list, and the numbers in $b_q, \ldots, b_m$ are at most $(\ell + 1)$-smallest and also can not be median of the union list. We can remove the two parts and continue to find the $(\ell - p + 1)$-smallest number among $a_p, \ldots, a_n, b_1, \ldots, b_{q-1}$.

To apply this algorithm in this problem, we can just let $\ell = \frac{n+m+1}{2}$.

Correctness: According to the above explanation, each time we remove the numbers that can not be the median of the union list. Hence, the lengths of the two lists are monotone decreasing. Therefore, the algorithm will finally terminate and finally find the median.

Time complexity: Each time we find a_p and b_q, it costs $O(1)$. Each time we divide the minimum length of the two lists by 2 or divide ℓ by 2. Therefore, the total time complexity is at most $O(\log(n)) + O(\log(m)) + O(\log(n + m)) = O(\log(m))$, which is better than $O(m)$.

This solution is provided by ***T.A. Panfeng Liu*** *and* ***Stu. Xiangyuan Xue****.* $\square$

Problem 4 (30 points)[†]

An integer triple (x, y, z) forms a *good order* if $y - x = z - y$. Given an integer n, we call a permutation of $\{1, \ldots, n\}$ (denoted by $X = \{x_1, x_2, \ldots, x_n\}$) is *out-of-order* if it has the following property: for every $i < j < k$, the triple (x_i, x_j, x_k) does not form a *good order* (i.e., $x_j - x_i \neq x_k - x_j$).

(a) Prove that an integer triple (x, y, z) forms a *good order* only if x and z are both even or both odd.

(b) Write down a permutation of $\{1, \ldots, 4\}$, and quickly prove it is *out-of-order* by the claim above.

(c) Prove that an integer triple (x, y, z) where x, y, z are all odd forms a good order only if $((x + 1)/2, (y + 1)/2, (z + 1)/2)$ forms a *good order*.

(d) Design a divide and conquer algorithm runs in $O(n \log n)$ time to output an *out-of-order* permutation of $\{1, \ldots, n\}$. (The running time can be improved to $O(n)$.)

Solution. **(a)** When (x, y, z) forms a good order, it means that $y - x = z - y$. Therefore, $x + z = 2y$, which implies that x and z are both even or both odd.

(b) Consider such permutation $4, 2, 1, 3$. All the possible triples are

$$(4, 2, 1), (4, 2, 3), (4, 1, 3), (2, 1, 3).$$

It is easy to verify all of them do not satisfy the requirement of good order.

(c) We can notice the following fact holds:

$$\begin{aligned}
&(x, y, z) \text{ forms a good order.} \\
\Leftrightarrow\ & y - x = z - y \\
\Leftrightarrow\ & (y + 1) - (x + 1) = (z + 1) - (y + 1) \\
\Leftrightarrow\ & \frac{y+1}{2} - \frac{x+1}{2} = \frac{z+1}{2} - \frac{y+1}{2} && \text{(Since all of them are odd)} \\
\Leftrightarrow\ & \left(\frac{x+1}{2}, \frac{y+1}{2}, \frac{z+1}{2}\right) \text{ forms a good order,}
\end{aligned}$$

(d) Inspired by the second subproblem, if (x, y, z) forms a good order. Then, x and z should be both even or both odd. So we can just put all the odd numbers of $\{1, \ldots, n\}$ on the left side and put all the even numbers of $\{1, \ldots, n\}$ on the right side. Hence, there will not occur triples whose leftmost one locates in the first half and rightmost one locates in the last half. The remaining problem is how to ensure now good orders inside the two halves. Inspired by (c), we can just use divide and conquer.

The formal description of this algorithm is as follows: We use divide and conquer. When $n = 1$, we just output $\{1\}$. When n is even, let the output of $\{1, \ldots, \frac{n}{2}$ be $\{a_1, \ldots, a_{\frac{n}{2}}\}$. Then we output

$$\{2a_1 - 1, \ldots, 2a_{\frac{n}{2}} - 1, 2a_1, \ldots, 2a_{\frac{n}{2}}\}.$$

When n is larger than 1 and odd, let the output of $\{1, \ldots, \frac{n+1}{2}$ and $\{1, \ldots, \frac{n-1}{2}\}$ be $\{a_1, \ldots, a_{\frac{n+1}{2}}\}$ and $\{b_1, \ldots, b_{\frac{n-1}{2}}\}$ respectively. Then we output

$$\{2a_1 - 1, \ldots, 2a_{\frac{n+1}{2}} - 1, 2b_1, \ldots, 2b_{\frac{n-1}{2}}\}.$$

Now let us demonstrate the correctness of this algorithm by induction. If the algorithm works correctly when $n \le m \in \mathbb{N}^+$. When $n = m + 1$, if there exists a good order (x_i, x_j, x_k) in the output permutation. According to subproblem (b), x_i and x_k should be both even or both odd. Hence, the three numbers should locate in the same part. Nevertheless, no matter which part they locate, it can not form good order according to the correctness when $n = \frac{m}{2}$ (or $n = \frac{m+1}{2}, \frac{m-1}{2}$ if m is odd).

Denote the time complexity when input is n by $T(n)$. According to our previous explanation, we have

$$T(n) = T\left(\frac{n}{2}\right) + O(n) \le 2 \cdot T\left(\frac{n}{2}\right) + O(n) \qquad \text{(when } n \text{ is even)}$$

$$T(n) = T\left(\frac{n+1}{2}\right) + T\left(\frac{n-1}{2}\right) + O(n) \qquad \text{(when } n \text{ is odd)}$$

Therefore, the time complexity $T(n)$ is $O(n \log n)$.

This solution is provided by ***T.A. Panfeng Liu*** *and* ***Stu. Xiangyuan Xue***. □

Problem 5 (25 points)

In the *cake cutting* problem, you are going to allocate a piece of birthday cake to a group of n children. Different parts of the cake may have different toppings: some parts may be covered by chocolate, some parts may be covered by strawberry, some parts may be covered by pineapple, and so on. Different children may have different preferences over different parts of the cake. The objective is to allocate the cake to the n children *fairly* such that each child believes he receives the average value based on his preference. This is formally modelled as follows.

The cake is modelled by the 1-dimensional interval $[0,1]$. Each child i has a *value density function* $f_i : [0,1] \to \mathbb{R}_{\geq 0}$ representing i's preference over the cake $[0,1]$. Given an interval $I \subseteq [0,1]$, child i's *value* on I is given by the Riemann integral $\int_I f_i(x)dx$. An *allocation* is a collection of n intervals $(I_1,\dots,I_n)$ such that every pair of intervals I_i and I_j can only intersect at the endpoints, where I_i is the interval allocated to child i. An allocation $(I_1,\dots,I_n)$ is *proportional* if $\int_{I_i} f_i(x)dx \geq \frac{1}{n}\int_0^1 f_i(x)dx$ for each child i, i.e., each child i thinks he receives the average value based on his value density function. In this question, you are going to analyze a classical cake cutting algorithm, *the Even-Paz algorithm*, which is described below.

For each child i, a half-half point $x_i \in [0,1]$ is computed such that the ratio of the values for the two pieces $[0,x_i]$ and $[x_i,1]$ is $\lfloor \frac{n}{2} \rfloor : \lceil \frac{n}{2} \rceil$. That is

$$\int_0^{x_i} f_i(x)dx = \frac{1}{n}\left\lfloor \frac{n}{2} \right\rfloor \cdot \int_0^1 f_i(x)dx \quad \text{and} \quad \int_{x_i}^1 f_i(x)dx = \frac{1}{n}\left\lceil \frac{n}{2} \right\rceil \cdot \int_0^1 f_i(x)dx.$$

Next, we find the child i^* such that exactly $\lfloor \frac{n}{2} \rfloor$ children's half-half points are less than or equal to child i^*'s half-half point x_{i^*}. Let S be the set of those $\lfloor \frac{n}{2} \rfloor$ children, and $\bar{S}$ be the set of the remaining children. The cake is then cut to two parts $[0,x_{i^*}]$ and $[x_{i^*},1]$. An allocation of $[0,x_{i^*}]$ to the children in S and an allocation of $[x_{i^*},1]$ to the children in $\bar{S}$ are then computed *recursively*.

(a) Write down the Even-Paz algorithm in pseudo-codes.

(b) Suppose the half-half point x_i for each child i can be computed in $O(1)$ time. Analyze the time complexity of the Even-Paz algorithm.

(c) Prove that the allocation returned by the Even-Paz algorithm is proportional.

Solution. (a) See Algorithm 2.

(b) We solve the problem by:

1) Calculating $\{x_i\}$.
2) Finding the median of $\{x_i\}$.
3) Solving two half-sized problem.

By assumption 1) costs $n \times O(1) = O(n)$ time. 2) is a special case of finding the k-th smallest number, so it

Algorithm 2 Even-Paz Algorithm

Input: Interval $[0,1]$, value density function f_i for $i = 1, \ldots, n$

Output: An proportional allocation $(I_1, \ldots, I_n)$

```
1:  function EVENPAZ($[l, r], C$)
2:      if $|C| = 1$ then
3:          return $I_{C_1} \leftarrow [l, r]$
4:      for $i \leftarrow 1 : |C|$ do
5:          $x_i \leftarrow i$'s half-half point
6:      $x_{i^*} \leftarrow$ SELECT($\{x_i\}, \left\lfloor \frac{|C|}{2} \right\rfloor$)        ▷ Finding the median is a special case of finding the $k$-th smallest element.
7:      $S \leftarrow \{i : x_i \leq x_{i^*}\}$
8:      $\bar{S} \leftarrow C \setminus S$
9:      return EVENPAZ($[l, x_{i^*}], S$) $\cup$ EVENPAZ($[x_{i^*}, r], \bar{S}$)
10: EVENPAZ($[0, 1], \{1, \ldots, n\}$)
```

costs $O(n)$.

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

therefore, by Master Theorem, $T(n) = O(n \log n)$.

(c) **Proof 1 (Divide-and-Conquer, Bottom-Up Approach).**

Terminating case: trivial.

Merge: For all children $i \in S$, by recursively applying the algorithm,

$$\int_{I_i} f_i(x)dx \geq \frac{1}{|S|} \int_l^{x_{i^*}} f_i(x)dx = \frac{1}{\left\lfloor \frac{|C|}{2} \right\rfloor} \int_l^{x_{i^*}} f_i(x)dx. \tag{1}$$

Without loss of generality, assume $\int_l^r f_i(x)dx = 1$ for all $i \in C$. By algorithm,

$$\int_l^{x_{i^*}} f_i(x)dx \geq \int_l^{x_{i^*}} f_{i^*}(x)dx = \frac{1}{|C|} \left\lfloor \frac{|C|}{2} \right\rfloor \int_l^r f_{i^*}(x)dx = \frac{1}{|C|} \left\lfloor \frac{|C|}{2} \right\rfloor \int_l^r f_i(x)dx. \tag{2}$$

Combining (1) and (2),

$$\int_{I_i} f_i(x)dx \geq \left(\frac{1}{\left\lfloor \frac{|C|}{2} \right\rfloor} \right) \left(\frac{1}{|C|} \left\lfloor \frac{|C|}{2} \right\rfloor \int_l^r f_i(x)dx \right) = \frac{1}{|C|} \int_l^r f_i(x)dx.$$

Similarly, for all children $i \in \bar{S}$, the argument holds as well.

Therefore the divide-and-conquer algorithm is correct.

Proof 2 (Intuitive, Top-Down Approach).

Without loss of generality, assume $\int_l^r f_i(x)dx = 1$ for all $i \in C$. After each cut at x_{i^*}, for all children $i \in S$,

$$\int_l^{x_{i^*}} f_i(x)dx \geq \int_l^{x_{i^*}} f_{i^*}(x)dx = \frac{1}{|C|} \left\lfloor \frac{|C|}{2} \right\rfloor \int_l^r f_{i^*}(x)dx = \frac{1}{|C|} \left\lfloor \frac{|C|}{2} \right\rfloor \int_l^r f_i(x)dx \tag{3}$$

By algorithm (go back until the first level), we have[1]

$$\int_l^r f_i(x)dx \geq \frac{1}{n}|C| \int_0^1 f_i(x)dx. \tag{4}$$

[1] Details omitted, assume n to be 2's power to get intuition.

Combining (3) and (4),

$$\int_{l}^{x_{i^*}} f_i(x)dx \geq \left(\frac{1}{|C|}\left\lfloor\frac{|C|}{2}\right\rfloor\right)\left(\frac{1}{n}|C|\int_{0}^{1} f_i(x)dx\right) = \frac{1}{n}\left\lfloor\frac{|C|}{2}\right\rfloor\int_{0}^{1} f_i(x)dx \geq \frac{1}{n}\int_{0}^{1} f_i(x)dx.$$

Similar for children $i \in \bar{S}$.

After each cut, every child is still satisfied with his/her "share" of cake. When the algorithm terminates, each "share" is only shared by one child, so the assignment is valid, this assignment satisfies everyone.

This solution is provided by ***T.A. Kangrui Mao***. □