# Algorithm Design and Analysis

# Assignment 1

# Deadline: March 23, 2025

1. (24 points) Asymptotic notations.

   (a) (24 points) In each of the following situations, indicate whether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$). Justify your answer.

   1. $f(n) = (n+1)!$ and $g(n) = n!$
      **$f = \Omega(g)$ because $\frac{f(n)}{g(n)} = (n+1)$ is unbounded.**
   2. $f(n) = 2^{n+1}$ and $g(n) = 2^n$
      **$f = \Theta(g)$ because $\frac{f(n)}{g(n)} = 2$ is a constant.**
   3. $f(n) = 2^n$ and $g(n) = 3^n$
      **$f = O(g)$ because $\frac{f(n)}{g(n)} = (\frac{2}{3})^n$ converges to 0.**
   4. $f(n) = n^{1/2}$ and $g(n) = 5^{\log_2 n}$
      **$f = O(g)$ because $\log_2 5 > 1/2$, so that $\frac{f(n)}{g(n)} = \frac{n^{1/2}}{n^{\log_2 5}}$ converges to 0.**
   5. $f(n) = 100n + \log n$ and $g(n) = n + (\log n)^2$
      **$f = \Theta(g)$ because $\frac{f(n)}{g(n)} = 100$ is a constant.**
   6. $f(n) = (\log n)^{\log n}$ and $g(n) = n/\log n$
      **$f = \Omega(g)$ because $\frac{f(n)}{g(n)} = (n+1)$ is unbounded.**
   7. $f(n) = (\log n)^{\log n}$ and $g(n) = 2^{(\log_2 n)^2}$
      **$f = O(g)$ because $\frac{f(n)}{g(n)} = \frac{(\log n)^{\log n}}{2^{(\log_2 n)^2}}$ converges to 0.**
   8. $f(n) = \sum_{i=1}^{n} i^k$ and $g(n) = n^{k+1}$

      - if k > -1, $f = \Theta(g)$ because $\frac{f(n)}{g(n)} = \frac{\sum_{i=1}^{n} i^k}{n^{k+1}}$ converges to $\frac{1}{k+1}$.
      - if k <= -1, $f = O(g)$ because $\frac{f(n)}{g(n)} = \frac{\sum_{i=1}^{n} i^k}{n^{k+1}}$ is unbounded.

   (b) (Not for credit, just for fun: 0 points) Suppose $f : \mathbb{Z}^+ \to \mathbb{R}^+$ and $g : \mathbb{Z}^+ \to \mathbb{R}^+$ are increasing functions. Is it always true that we have either $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$?
   **I think the answer is no. Suppose $f(n) = 2^n$, while n=2k and $f(n) = 2^{n-1} + 1$, while n=2k+1; $g(n) = 2^{n-1} + 1$, while n=2k and $g(n) = 2^n$, while n=2k+1. Apparently, $f(n) = O(g(n))$, n=2k, while $f(n) = \Omega(g(n))$, n=2k+1, which means we have neither at $\mathbb{Z}^+$.**

2. (25 points) Given an array $A[1 \cdots n]$ of integers, a pair of indices $(i, j)$ is an *inversion* if $i < j$ and $A[i] > A[j]$. Design an algorithm that counts the number of inversions in $O(n \log n)$ time.

**Suppose p is the number of inversions.**

**Devide the array A into two halves(let's say M,N with length m,n) with indices** $(i, j)$**.**

**For i from 0 to m-1,j from 0 to n-1,repeat the following steps:**

**1:Record** $\min(Mi, Nj)$ **in A'**

**2:If Mi < Nj, then p=p+j and i=i+1; If Mi > Nj, then j=j+1 (a bit counterintuitive but practical)**

**3:if i > n, break;or if j >m, then p=p+(n-i)*j and break.**

**4:Record the rest to A'.**

**Apparently, T(n)=2T(n/2)+O(2n), which means** $T(n) = O(n \log n)$**.**

3. (25 points) Given an array of $n$ integers $x_1, x_2, ..., x_n$, there are queries of the following form: given an integer $1 \leq k \leq n$, you need to return the $k$-th smallest integers in the array. Obviously, if we use $O(n \log n)$ time to preprocess the array by sorting it, we can answer each query in $O(1)$ time. In the class, we see that each query can be answered in $O(n)$ time without any preprocessing (the Median-of-the-Medians algorithm). Now, design an $O(n)$ preprocessing algorithm so that you can answer each query in $O(k)$ time.
**Suppose f is the preprocessing function, defined as follows:**
**1:Use the Median-of-the-Medians algorithm to find the median, denoted as m.**
**2:Divide the array into three parts:**

- **L:those smaller than m;**
- **M:those equal to m;**
- **R:those larger than m.**

**As is mentioned above, step 2 takes O(n) time.**
**Repeat: apply f on L till there is only one integer in L.**
**After the preprocess above, we can get an array of medians of L, which could be used as a mark in the searching step.**
**Searching:**

- **If k < |L|, in that L is an ordered array, T=O(k);**
- **If |L| < k < |L|+|M|, clearly k is the answer;**
- **If k > |L|+|M|, in that $|R| < \frac{n}{2}$, $T = O(\frac{n}{2}) < O(k)$.**

**In conclusion, the searching step takes O(k) time, which satisfies the conditions.**

4. (26 points) You are given a 2D discrete topographical map $A[0,\ldots,n-1;0,\ldots,m-1]$ representing a landscape. The number $A[i,j]$ represents the altitude at position $(i,j)$. If it rains over the landscape, the water will form pools at each position where $A[i,j]$ is less than each adjacent position, i.e., those $A[i',j']$ for which $|i-i'|+|j-j'|=1$. (You can assume all altitudes are distinct and that there is a "wall" at the edge of the map. For example, water pools at $(0,0)$ if $A[0,0]$ is less than $A[0,1]$ and $A[1,0]$.)

   (a) Suppose $m=1$, so $A$ is a 1D array. Give a divide and conquer algorithm for finding *one* position where water pools. Write a recurrence for this algorithm. Analyze its running time.

   **Algorithm: Suppose mid=n//2, compare A[1,mid-1],A[1,mid] and A[1,mid+1].**
   **Apparently, if A[1,mid-1] > A[1,mid] and A[1,mid+1] > A[1,mid], then A[1,mid] is the pool.**
   **Else, choose the smaller half and repeat the steps above.**
   **Recurrence: T(n)=T(n/2)+O(1).**
   **Thus T(n)=O(log n).**

   (b) Give another divide and conquer algorithm when $m=n$. Analyze its running time with a recurrence relation.

   **Because m=n, searching direction is still on a line, so actually (2) equals to (1).**
   **Algorithm: Suppose mid=n//2=m//2, compare A[mid,mid],A[mid+1,mid+1] and A[mid-1,mid-1].**
   **Apparently, if A[mid,mid] < A[mid+1,mid+1] and A[mid,mid] < A[mid-1,mid-1], then A[mid,mid] is the pool.**
   **Else, choose the smaller direction and repeat the steps above.**
   **Recurrence: T(n)=T(n/2)+O(1).**
   **Thus T(n)=O(log n).**

   (c) Generalize your algorithms from part (a) and (b) to work for any $m$ and $n$. The running time should *smoothly* interpolate between the running times of (a) and (b). **Algorithm: Suppose $mid_m$=m//2, $mid_n$=n//2, compare A[$mid_m$,$mid_n$] with A[$mid_m \pm 1$,$mid_n$] and A[$mid_m$,$mid_n \pm 1$].**
   **If A[$mid_m$,$mid_n$] is minimum,then it's the pool.**
   **Else, choose the smallest direction, and repeat the steps above.**
   **Recurrence: for each step, T(m,n)=T(m/2,n)+O(n) or T(m,n/2)+O(m).**
   **Thus T(m,n)=O(m+n).**

5. How long does it take you to finish the assignment (including thinking and discussion)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Please write down their names here.

   **About 7 hours.**

   **4**

   **Ruifeng Shang.**