List of Abbreviations and Symbols

$A[1..n]$	An array indexed from 1 to n of n elements.
$\mathbb{N}$	Set of all nature numbers, i.e., $\{1, 2, 3, \dots\}$.
$\mathbb{R}$	Set of all real numbers.
$\mathbb{Z}$	Set of all integers, i.e., $\{\dots, -2, -1, 0, 1, 2, \dots\}$.

Modifiers

To help you with what problems to try, problems marked with **[K]** are key questions that tests you on the core concepts, please do them first. Problems marked with **[H]** are harder problems that we recommend you to try after you complete all other questions (or perhaps you prefer a challenge). Good luck!!!

Contents

§1 Introductory problems

Exercise 1.1. [K] You are given an array $A[1..2n-1]$ of integers. Design an algorithm which finds all of the n possible sums of n consecutive elements of A **which runs in time $\mathbf{O(n)}$**. Thus, you have to find the values of all of the sums

$$\begin{aligned} S[1] &= A[1] + A[2] + \ldots + A[n-1] + A[n]; \\ S[2] &= A[2] + A[3] + \ldots + A[n] + A[n+1]; \\ &\vdots \\ S[n] &= A[n] + A[n+1] + \ldots + A[2n-2] + A[2n-1], \end{aligned}$$

and your algorithm **should run in time $\mathbf{O(n)}$**.

Exercise 1.2. [K] Given an array $A[1..n]$ which contains all natural numbers between 1 and $n-1$, design an algorithm that runs in $O(n)$ time and returns the duplicate value, using:

(a) $O(n)$ additional space.

(b) $O(1)$ additional space.

Exercise 1.3. [K] You are given an array A of n distinct integers.

(a) You have to determine if there exists a number (not necessarily in A) which can be written as a sum of squares of two distinct numbers from A in two different ways (note: $A[m]^2 + A[k]^2$ and $A[k]^2 + A[m]^2$ counts as a single way) and which runs in time $n^2 \log n$ in the **worst case** performance. Note that the brute force algorithm would examine all quadruples of elements in A and there are $\binom{n}{4} = O(n^4)$ such quadruples.

(b) Solve the same problem but with an algorithm which runs in the **expected time** of $O(n^2)$.

Exercise 1.4. [H] You are conducting an election among a class of n students. Each student casts precisely one vote by writing their name, and that of their chosen classmate on a single piece of paper. We assume that students are not allowed to vote for themselves. However, the students have forgotten to specify the order of names on each piece of paper – for instance, "Alice Bob" could mean that Alice voted for Bob, or that Bob voted for Alice!

(a) Show how you can still uniquely determine how many votes each student received.

(b) Hence, explain how you can determine which students did not receive any votes. Can you determine who these students voted for?

(c) Suppose every student received at least one vote. What is the maximum possible number of votes received by any student? Justify your answer.

(d) Using parts (a) and (c), or otherwise, design an algorithm that constructs a list of votes of the form "X voted for Y" consistent with the pieces of paper. Specifically, each piece of paper should match up with precisely one of these votes. If multiple such lists exist, produce any. An $O(n^2)$ algorithm earns partial credit, but you should aim for an $O(n)$ algorithm.

Hint — *first, use part (c) to consider how you would solve it in the case where every student received at least one vote. Then, apply part (b).*

Exercise 1.5. [H] 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 who does not know anyone else present. 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 $3n - \lfloor \log_2 n \rfloor - 3$ such questions, even in the worst case.

Exercise 1.6. [H] There are N teams in the local cricket competition and you happen to have N friends that keenly follow it. Each friend supports some subset (possibly all, or none) of the N teams. Not being the sporty type – but wanting to fit in nonetheless – you must decide for yourself some subset of teams (possibly all, or none) to support. You don't want to be branded a copycat, so your subset must not be identical to anyone else's. The trouble is, you don't know which friends support which teams, so you can ask your friends some questions of the form "Does friend A support team B?" (you choose A and B before asking each question). Design an algorithm that determines a suitable subset of teams for you to support and asks as few questions as possible in doing so.

Exercise 1.7. [H] You are in a square orchard of $4n$ by $4n$ equally spaced trees. You want to purchase apples from precisely n^2 many of those trees, which also form a square. Fortunately, the owner is allowing you to choose such a square anywhere in the orchard and you have a map with the number of apples on each tree. Your task is to choose a square that contains the largest amount of apples and which runs in time $O(n^2)$. Note that the brute force algorithm would run in time $\Theta(n^4)$.

§2 Searching algorithms

Exercise 2.1. [K] Let M be an $n \times n$ matrix of distinct integers $M(i,j)$, $1 \le i \le n$, $1 \le j \le n$. Each row and each column of the matrix is sorted in the increasing order, so that for each row i, $1 \le i \le n$,

$$M(i,1) < M(i,2) < \ldots < M(i,n)$$

and for each column j, $1 \le j \le n$,

$$M(1,j) < M(2,j) < \ldots < M(n,j).$$

You need to determine whether M contains an integer x in $O(n)$ time.

Exercise 2.2. [K] You are given an array of $A[1..2^n]$ distinct integers. You are tasked to find the largest and the second largest number using only $2^n + n - 2$ comparisons.

Exercise 2.3. [K] Assume you have an array of $2n$ distinct integers. Find the largest and the smallest number using $3n - 2$ comparisons only.

Exercise 2.4. [K] Let $f : \mathbb{N} \to \mathbb{Z}$ be a monotonically increasing function. That is, for all $i \in \mathbb{N}$, we have that $f(i) < f(i+1)$. Our goal is to find the smallest value of $i \in \mathbb{N}$ so that $f(i) \ge 0$. Design an $O(\log n)$ algorithm to find the value of i so that $f(i) \ge 0$ for the first time.

Hint — *If the function notation is confusing, it might help to consider this problem in the context of an* ***array****. Specifically, how are arrays and functions closely related?*

Exercise 2.5. [K] You're given an array $A[1..n]$ of integers, and you are required to answer a series of n queries, each of the form: "how many elements of the array have a value between L and R (inclusively)?", where L and R are integers. Design an $O(n \log n)$ algorithm that answers all n of these queries.

Exercise 2.6. [H] Assume you are given two arrays A and B, each containing n distinct positive numbers and the equation $x^8 - x^4y^4 = y^6 + x^2y^2 + 10$. Design an algorithm which runs in time $O(n \log n)$ which finds if A contains a value for x and B contains a value for y that satisfy the equation.

Exercise 2.7. [H] You are a fisherman, trying to catch fish with a net that is W meters wide. Using your advanced technology, you know that the positions of all N fish in the sea can be represented as integers on a number line. There may be more than one fish at the same location.
To catch the fish, you will cast your net at position x, and will catch all fish with positions between x and $x + W$, **inclusive**. Given N, W and an array $X[1..N]$ denoting the positions of fish in the sea, give an $O(N \log N)$ algorithm to find the maximum number of fish you can catch by casting your net once.
For example, if $N = 7, W = 3$ and $X = [1, 11, 4, 10, 6, 7, 7]$, then the most fish you can catch is 4: by placing your net at $x = 4$, you will catch one fish at position 4, one fish at position 6 and two fish at position 7.

Exercise 2.8. [H] Your army consists of a line of N giants, each with a certain height. You must designate precisely $L \le N$ of them to be leaders. Leaders must be spaced out across the line; specifically, every pair of leaders must have at least $K \ge 0$ giants standing in between them. Given N, L, K and the heights $H[1..N]$ of the giants in the order that they stand in the line as input, find the *maximum* height of the *shortest* leader among all valid choices of L leaders. We call this the *optimisation* version of the problem.

For instance, suppose $N = 10, L = 3, K = 2$ and $H = [1, 10, 4, 2, 3, 7, 12, 8, 7, 2]$. Then among the 10 giants, you must choose 3 leaders so that each pair of leaders has at least 2 giants standing in between them. The best choice of leaders has heights 10, 7 and 7, with the shortest leader having height 7. This is the best possible for this case.

(a) In the *decision* version of this problem, we are given an additional integer T as input. Our task is to decide if there exists some valid choice of leaders satisfying the constraints whose shortest leader has height no less than T.
Give an algorithm that solves the decision version of this problem in $O(N)$ time.

(b) Hence, show that you can solve the optimisation version of this problem in $O(N \log N)$ time.

Exercise 2.9. [H] Let A be an array of n integers, not necessarily distinct or positive. Design a $\Theta(n \log n)$ algorithm that returns the maximum sum found in any contiguous subarray of A.

Note that there is an $O(n)$ algorithm that solves this problem; however, the technique used in that solution is much more involved and will be taught in due time.

§3 Sorting algorithms

Exercise 3.1. [K] You are given an array $A[1..n]$ of integers and another integer x.

(a) Design an $O(n \log n)$ algorithm (in the sense of the worst case performance) that determines whether or not there exist two elements in A whose sum is exactly x.

(b) Design an algorithm that accomplishes the same task, but runs in $O(n)$ **expected** (i.e., average) time.

Exercise 3.2. [K] Given two arrays of n integers, design an algorithm that finds out in $O(n \log n)$ steps if the two arrays have an element in common.

Exercise 3.3. [K] Suppose that you are taking care of n kids, who took their shoes off. You have to take the kids out and it is your task to make sure that each kid is wearing a pair of shoes of the right size (not necessarily their own, but one of the same size). All you can do is to try to put a pair of shoes on a kid, and see if they fit, or are too large or too small; you are NOT allowed to compare a shoe with another shoe or a foot with another foot. Describe an algorithm whose expected number of shoe trials is O$(n \log n)$ which properly fits shoes on every kid.

Hint — *Try "double"* QuickSort*; one which uses a foot as a pivot for shoes and another one which uses a shoe as a pivot for feet.*

Exercise 3.4. [H] Given n real numbers $x_1, \ldots, x_n$ in the interval $[0, 1)$, devise an algorithm that runs in linear time which outputs a permutation of the n numbers, say $y_1, \ldots, y_n$, such that

$$\sum_{i=2}^{n} |y_i - y_{i-1}| < 2.$$

Hint — *this is easy to do in $O(n \log n)$ time: just sort the sequence in ascending order. In this case, $\sum_{i=2}^{n} |y_i - y_{i-1}| = \sum_{i=2}^{n} (y_i - y_{i-1}) = y_n - y_1 \le 1 - 0 = 1$. Here $|y_i - y_{i-1}| = y_i - y_{i-1}$ because all the differences are non-negative, and all the terms in the sum except the first and the last one cancel out. To solve this problem, one might think about tweaking the* BucketSort *algorithm, by carefully avoiding sorting numbers in the same bucket.*

Exercise 3.5. [H] Given n real numbers $x_1, \ldots, x_n$ where each x_i is a real number in the interval $[0, 1)$, devise an algorithm that runs in linear time and that will output a permutation of the n numbers, say $y_1, \ldots, y_n$, such that $\sum_{i=2}^{n} |y_i - y_{i-1}| < 1.01$.

Hint — *Refer to the previous exercise. How do you alter the previous exercise to do this problem?*

§4 Complexity analysis

Exercise 4.1. [K] Determine if $f(n) = \Omega(g(n))$, $f(n) = O(g(n))$, both (i.e. $f(n) = \Theta(g(n))$) or neither 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}	$2^{n/100}$
$\sqrt{n}$	$2^{\sqrt{\log_2 n}}$
$n^{1.001}$	$n \log_2 n$
$n^{(1+\sin(\pi n/2))/2}$	$\sqrt{n}$

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

Exercise 4.2. Classify the following pairs of functions by their asymptotic relation; that is, determine if $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, both (i.e. $f(n) = \Theta(g(n))$) or neither.

(a) [K] $f(n) = 1/n$, $g(n) = \sin(1/n)$.

(b) [K] $f(n) = \log(n!)$, $g(n) = \log(n^n)$. What does this say about the growth rate between $f_1(n) = n!$ and $f_2(n) = n^n$?

(c) [H] $f(n) = n^{\log n}$, $g(n) = (\log n)^n$.

(d) [H] $f(n) = (-1)^n$, $g(n) = \tan(n)$.

(e) [H] $f(n) = n^{5/2}$, $g(n) = \left[\log\left(\sum_{k=0}^{\infty} \frac{4^{k+n} n^k}{k!}\right)\right]^2$.

Hint — *Consider the Taylor series expansion of $h(x) = e^x$.*

Exercise 4.3. [H] Define $f : \mathbb{N} \to \mathbb{R}$ by

$$f(n) = \begin{cases} 1 & \text{if } n \leq 2, \\ f\left(\left\lceil \frac{n}{\log_2 n} \right\rceil\right) + n & \text{if } n \geq 3. \end{cases}$$

Show that $f(n) = \Theta(n)$.

Hint — *Try showing that $f(n) = O(n)$ inductively and then $f(n) = \Omega(n)$ to conclude that $f(n) = \Theta(n)$.*