CS218 Design and analysis of algorithms **Jan-Apr 2024**

Exercise sheet 1

Lecture 1-3 (Jan 4, 8, 9) Binary search and variants

1. Consider a building with infinitely many floors. You need to find the highest floor h from which an egg can be dropped without breaking. Can you do it with $O(\log h)$ egg droppings?

2. Given an array with n positive integers $[a_1, a_2, \ldots, a_n]$ and a target value S, find the minimum length subarray whose sum is at least S. Can you do it in $O(n \log n)$ time?

   A subarray means a contiguous subset. That is for some $i \leq j$, $[a_i, a_{i+1}, \ldots, a_{j-1}, a_j]$.

3. Given two sorted integer arrays (with all distinct numbers in them) of size n, we want to find the median of the union of the two arrays. Can you find it by accessing only $O(\log n)$ entries in the two arrays.

4. Given an array of n integers and an integer S, find a pair of integers in the array whose sum is S. Can you do it in $O(n \log n)$ time?

5. Let $f\colon \mathbb{R} \to \mathbb{R}$ be a convex function that is promised to have a minimizing point. The function $f(x)$ and its derivative $f'(x)$ are not given explicitly, but via oracle access. That is, you can give any point $x \in \mathbb{R}$ and the oracle will give the values of $f(x)$ and $f'(x)$. How many queries do you need you find the point minimizing $f(x)$?

6. Given an integer a, check if it is of the form b^k for some (unknown) integers b and $k > 1$. Can you do this in time $O(\log^c a)$ for some constant c?

7. You are given a list of landholdings, say $a_1, a_2, \ldots, a_n$ (in hectares, can be zero). We want to give land to everyone who has less than f hectares and bring them up to f hectares. For this, we need to take away land from everyone who has more than c hectares, bring them down to c hectares, and redistribute the obtained land. (1) What is the highest value of f that can be feasible? (2) For a chosen value of f, find the right value of c?

8. You are standing on the number line at $x = 0$. There is a hidden treasure at $x = N$ for some unknown integer N (it could be positive or negative). You have a detector which will beep if you pass through the location of the treasure. What should be your strategy to minimize the total distance traveled and find the treasure? Can you ensure that the total distance travelled is $O(N)$.

   One strategy is to just keep traveling in the positive direction. If the treasure is on the positive side, then you can find it with distance travelled N. But, if it is on negative side, you will never find it. So this strategy does not work.

9. Prove that $\log(n!) \geq (n/2)\log(n/2) = (1/2)n \log n - n/2$.

10. Prove that $\log(n!) = \sum_{i=1}^{n} \log i \geq \int_1^n \log x \, dx$. Solve the integral to get a lower bound.

11. True or false?

    - $2n + 3$ is $O(n^2)$.
    - $\sum_{i=1}^{n} i^2$ is $O(n^2)$.
    - $\sum_{i=1}^{n} 1/i$ is $O(\log n)$.
    - n^n is $O(2^n)$.
    - 2^{3n} is $O(2^n)$.

- $(n+1)^3$ is $O(n^3)$.
- $(n+\sqrt{n})^2$ is $O(n^2)$.
- $\log(n^3)$ is $O(\log n)$.

Below exercises are just for interest, not in the syllabus.

12. We have seen the binary search method for computing square root of a number. For each query, we compute the square of current value. Is there a faster implementation like the division algorithm, where we don't need to compute the square in each iteration from scratch?

13. Consider two algorithms for finding square root of an integer, Babylonian method (Newton-Raphson method, check wiki) and Binary search. Which one do you think is faster?

14. Compare two algorithms to compute a root of a polynomial. Binary search and Newton-Raphson.

Lecture 4 (Jan 11) Reducing to a subproblem

15. Prove that
$$\int_1^{n+1} (1/x)dx \leq 1 + 1/2 + 1/3 + \cdots + 1/n \leq 1 + \int_1^n (1/x)dx.$$
Solve the integrals to get the bounds in closed form.

16. Suppose there is a trader for a particular commodity, whose license allows them to buy it only once and sell it only once during a season. Naturally, the commodity can be sold only after it is bought. The price of the commodity fluctuates every day. The trader wants to compute the maximum profit they could have made in the last season.

    Design an $O(n)$-time algorithm, where the input is the list of prices $\{p_1, p_2, \ldots, p_n\}$ for the n days in the last season and the output is the maximum possible profit. Assume addition, subtraction, comparison etc are unit cost. Sample input: Prices: 70, 100, 140, 40, 60, 90, 120, 30, 60.

    Output: Max profit: 80

17. **Celebrity.** There is a party with n people, among them there is 1 celebrity. A celebrity is someone who is known to everyone, but she does not know anyone. You need to identify the celebrity by only asking the following kind of queries: ask the ith person if they know the j person. Can you do this in $O(n)$ queries?

18. We want to evaluate a degree $n-1$ polynomial $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ at a given point $x = \alpha$.

    Input: integers $a_0, a_1, \ldots, a_{n-1}$ in an array. And an integer α.

    Output: $a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{n-1}\alpha^{n-1}$

    Design an algorithm for this which uses only $O(n)$ multiplications and additions.

19. You are given a function $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ (it is given as an integer array of size n where all the entries are between 1 and n. ith entry in the array is $f(i)$). You need to find the largest subset $S \subseteq \{1, 2, \ldots, n\}$ such that f is one-one and onto when restricted to S. In other words, (1) every element in S is mapped to an element in S and (2) no two elements in S are mapped to the same element in S. Design an $O(n)$ algorithm for this.

    Example input: 4, 6, 2, 3, 5, 4.

    Output: 2, 3, 4, 5, 6

20. Given a set of intervals we want to count the number of intervals which are not contained in any other interval. For example, interval (2, 4) is contained in (2, 5). Interval (2, 4) is contained in (1, 6). Interval (1, 4) is not contained in (2, 5). Interval (1, 4) is not contained in (0, 3).

    You can assume input is two integer arrays L and R of size n each. $L[i]$ is the left endpoint and $R[i]$ is the right endpoint of the ith interval. Design an $O(n \log n)$ algorithm for this.

    Example input: $L = [5, 3, 8, 11, 9, 7, 2, 15]$. $R = [10, 8, 12, 16, 20, 15, 13, 17]$.

    Output: 3

21. Given n integers and a number k, we want to compute the sum of products of all k size subsets. That is, given $a_1, a_2, \ldots, a_n$, we want to compute

$$\sum_{i_1 < i_2 < \cdots < i_k} a_{i_1} a_{i_2} \cdots a_{i_k}.$$

    Note that this is nothing but the coefficient of x^{n-k} in the polynomial $(x - a_1)(x - a_2) \cdots (x - a_n)$. Design an $O(nk)$ time algorithm for this.