CS218 Design and analysis of algorithms **Jan-Apr 2025**

Exercise sheet 1

Lecture 1-3 (Jan 6, 7, 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 $f : \mathbb{R} \to \mathbb{R}$ 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. Given n numbers, we want to find if there are two numbers which are equal. Prove that any comparison based algorithm must take $\Omega(n \log n)$ algorithm. You can do an adversarial argument. When you ask for comparing two numbers, adversary decides an answer only at that time. Adversary is trying to make you ask enough comparisons, so as to sort the n numbers.

An adversarial argument refers to the process of intentionally arguing from a contrary perspective to challenge the original position

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

11. 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.

    for n>2, log(n!) >= nlog(n)

12. True or false?

    - $2n + 3$ is $O(n^2)$. True, but a better bound would be O(n)

- $\sum_{i=1}^{n} i^2$ is $O(n^2)$. False, it is n^3 algorithm.
- $\sum_{i=1}^{n} 1/i$ is $O(\log n)$. true, just do integral of 1/x;
- n^n is $O(2^n)$. False 2^(nlog[2](n))
- 2^{3n} is $O(2^n)$. False
- $(n+1)^3$ is $O(n^3)$. True.
- $(n+\sqrt{n})^2$ is $O(n^2)$. True.
- $\log(n^3)$ is $O(\log n)$. True.

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

13. 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?

14. 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?

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

Newton Raphson: The method converges very quickly to the correct value. Each iteration approximately doubles the number of correct digits in the estimate.

Lecture 4 (Jan 13) Reducing to a subproblem

16. Prove that

how to prove it?

$$\int_1^{n+1} (1/x)dx \leq 1 + 1/2 + 1/3 + \cdots + 1/n \leq 1 + \int_1^{n} (1/x)dx.$$

Solve the integrals to get the bounds in closed form.

17. 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

18. In the class, it was suggested that maximum subarray sum problem can be converted into the maximum profit problem, which I gave as an exercise. Here is how.

Given the array A, compute another array with cumulative sums. That is, an array S whose ith entry is $A[1] + A[2] + \cdots + A[i]$. Now, observe that the subarray sum $A[j+1] + A[j+2] + \cdots + A[i]$ is same as $S[i] - S[j]$. Thus, our problem becomes to find $\max_{i>j}(S[i] - S[j])$.

This can be viewed as the max profit problem. The array S contains prices for day 1 to day n. We want to buy on a day and sell on a later day and the goal is to maximize profit.

Can you do the other direction, i.e., convert the max profit problem into maximum subarray sum problem?

19. **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?

20. 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.

little bit of doubt?

21. 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. Won't admit the output outside the subset.

Example input: 4, 6, 2, 3, 5, 4. Seems similar to stable/perfect matching problem.

Output: 2, 3, 4, 5, 6

22. 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

tough!!

23. 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.

Lecture 5 (Jan 14)

24. Given n intervals, design an $O(n \log n)$ algorithm to find the total length of their union.

25. Given n intervals, design an $O(n \log n)$ algorithm to find the number of intervals which do not have any overlap with any other interval.

26. Given n intervals, design an $O(n \log n)$ algorithm to find the largest set of mutually intersecting intervals.

27. (Out of syllabus) Consider the interval containment problem and the above three problems. Argue that any comparison based algorithm (that only makes comparisons between various start and end times) for any of these problems must take $\Omega(n \log n)$ time.

28. **Non-dominated points.** For points in 2 dimensions, we say (a, b) dominates (c, d) if $a \geq c$ and $b \geq d$. Given a set of n points in 2 dimensions, design an $O(n \log n)$ time algorithm to compute the set of points which are not dominated by any other point.

1 Lecture 6 (Jan 16)

29. **Area coverage.** Recall the problem we saw in the class. Given rectangles whose top edges lies on a common horizontal line, we want to find to total area covered by their union. We saw a divide and conquer approach in the class. Can you solve the problems instead with the following iterative approaches. In some cases, you may need some special data structures to store the current outline.

    - Sort the rectangles in increasing order of left endpoints and process them in this order.
    - Sort the rectangles in decreasing order of heights and process them in this order.
    - Sort the rectangles in increasing order of heights and process them in this order.
    - Sort all the left and right endpoints put together (with their labels left/right). Go over this list from left to right and at every x-coordinate maintain the set of "current" heights. Current heights means heights of all those rectangles which cover the current x-coordinate. Clearly, we also need the maximum of these heights. How will you update the set of current heights when you see a left end point or right end point? What kind of data structure will you need so that the update in each iteration can be made in $O(\log n)$ time.

30. **Majority Fingerprints.** You are given a collection of n fingerprints. You are also told that more than $(n/2)$ of these are identical to each other. You are only given access to an equality test, which takes two fingerprints and tells whether they are identical or not. Using this equality test, can you find out the one with more than $n/2$ copies in $O(n \log n)$ time?

31. **Significant inversions.** In an array A of integers, a pair (i, j) is called a significant inversion if $i < j$ and $A[i] > 2A[j]$. Design an $O(n \log n)$ time algorithm to find the number of significant inversions.

32. **Hidden lines.** Consider n lines in the 2D plane:
    At any x maximum y will win the race.
    $$y = m_i x + c_i$$
    for $1 \leq i \leq n$. We want to compute the set of lines which will be visible when one sees from $y = +\infty$. Let us elaborate. At an x-coordinate x_0, that line will be visible which has maximum the y coordinate among all lines. That is, the line which maximizes $m_i x_0 + c_i$. A line is called hidden if it is not visible at any x-coordinate. Design an $O(n \log n)$ time algorithm to find the set of visible (which are not hidden) lines. Assume the input is given as $(m_1, c_1), (m_2, c_2), \ldots,$.

33. **Linear programming.** Suppose you are given a list of linear inequalities in two variables (x, y) of the following form.
    $$\begin{aligned} y &\leq a_1 x + b_1 \\ y &\leq a_2 x + b_2 \\ &\vdots \\ y &\leq a_n x + b_n \end{aligned}$$
    Here each a_i is a number (positive/negative/zero) and each b_i is a **positive** number. We are interested in the region defined by these inequalities, together with
    $$y \geq 0.$$
    That is, we are interested in the set of points which simultaneously satisfy all these inequalities. This set of points forms a (convex) polygon. We want to find the list of corners of this polygon. This is useful in solving linear programs; optimal point is one of the corners.

    See Figure 1, for an example. It shows the region (shaded area) defined by the following six inequalities together with $y \geq 0$. The polygon we get is a pentagon, that is, a polygon with 5 corners. This is because two of the inequalities become redundant.

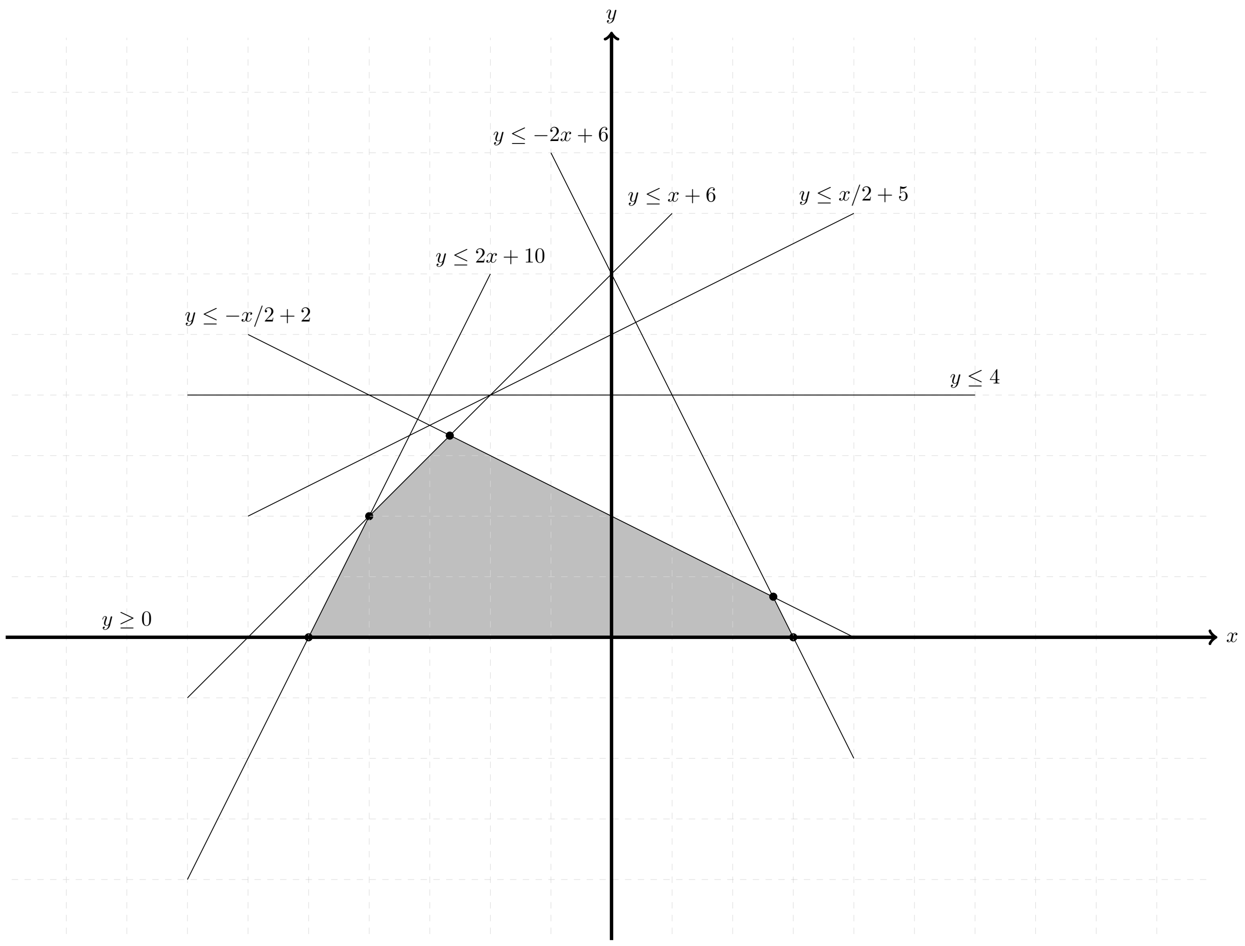

Figure 1: A polygon defined by 7 given inequalities, but with only 5 vertices.

$$
\begin{aligned}
y &\le 2x + 10\\
y &\le x + 6\\
y &\le x/2 + 5\\
y &\le -x/2 + 2\\
y &\le -2x + 6\\
y &\le 4
\end{aligned}
$$

Design an $O(n \log n)$ time algorithm that takes as input an array of pairs $[(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)]$ and outputs the list of corners of the polygon defined by inequalities

$$y \le a_i x + b_i \text{ for } 1 \le i \le n \text{ and } y \ge 0.$$

34. **Convex hull (Out of syllabus).** Consider the following problem that generalizes two problems: *Non-dominated points* and *area coverage by rectangles.* Given a set of n points in 2 dimensions, design an $O(n \log n)$ time algorithm to compute their convex hull. Convex hull is the smallest convex set containing the given points. For a finite set of points in 2D, their convex hull is a polygon. A polygon can be described by an ordered list of its corners or an ordered list of its edges. You need to compute the convex hull in one of these forms.

Lecture 7 (Jan 20)

35. Design a divide and conquer based algorithm ($O(n \log n)$ time) for the significant inversion problem.

36. Problems 28, 29, 31, 32, 33, 34 can be solved using two ways: (i) divide and conquer and (ii) sorting in an appropriate order, then scanning the input in that order and maintaining a suitable data structure like balanced binary search tree.

37. Augmented binary search tree means the usual BST with each node storing some additional information. This additional information can be useful in designing algorithms for certain problems. Which of the following information can be updated appropriately in $O(\log n)$ time, when a node is inserted or deleted in the BST. Assume that the height of the tree is $O(\log n)$.

    - each node x stores the number of nodes in the subtree rooted at x. This! while adding a node.
    - each node x stores the number of nodes larger than x. This, similar to significant inversion.
    - each node x stores its distance from the root. While adding a node this can be added.
    - each node x stores a pointer to its successor (that is the smallest number larger than x). Pred and succ of any nodecan be found in log(n)
    - each node x stores the largest depth of a node in the subtree rooted x. This is same as distance from the root!
    - each node x stores its rank in the BST.

38. **Faster Fibonacci.** Assuming unit cost multiplications/additions, can you compute the nth Fibonacci number in $O(\log n)$ operations? Try to convert the problem into computing nth power of a 2×2 matrix.