**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 $f\colon \mathbb{R} \to \mathbb{R}$ 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. 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.

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.

12. 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.

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.

## Lecture 4 (Jan 13) Reducing to a subproblem

16. 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.

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 $i$th 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 $i$th 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 $P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$ at a given point $x = \alpha$.

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

    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.

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 $n$. $i$th 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

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 $i$th 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

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:

    $$y = m_i x + c_i$$

    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.

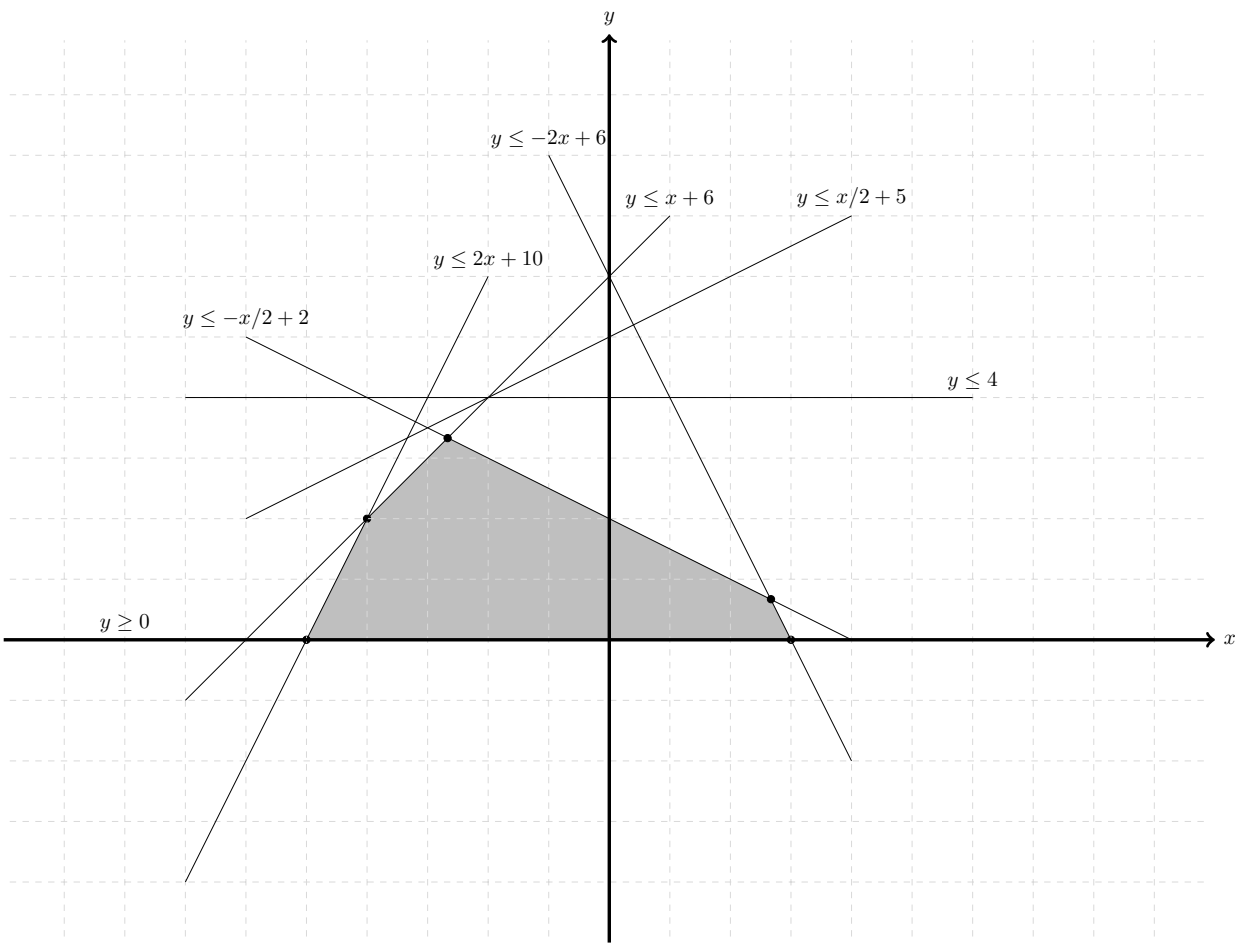



Figure 1: A polygon defined by 7 given inequalities, but with only 5 vertices.

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

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 $x$.
    - each node $x$ stores the number of nodes larger than $x$.
    - each node $x$ stores its distance from the root.
    - each node $x$ stores a pointer to its successor (that is the smallest number larger than $x$).
    - each node $x$ stores the largest depth of a node in the subtree rooted $x$.
    - each node $x$ stores its rank in the BST.

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

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

Exercise sheet 2

# Lecture 8, 9 Fibonacci, Integer Multiplication

1. Let us try to apply the divide and conquer approach on the integer multiplication problem. Suppose we want to multiply two $n$-bit integers $a$ and $b$. Write them as

$$a = a_1 2^{n/2} + a_0$$

$$b = b_1 2^{n/2} + b_0$$

   The product of the two integers can be written as

$$ab = a_1 b_1 2^n + (a_1 b_0 + a_0 b_1) 2^{n/2} + a_0 b_0.$$

   Can you compute these three terms $a_1 b_1$, $a_1 b_0 + a_0 b_1$, $a_0 b_0$, using only **three multiplications of** $n/2$ **bit integers** and a few additions/subtractions? If yes, then we will get an $O(n^{1.58})$ time algorithm.

2. Can you find square of an $n$-bit integer $a$, using square subroutine on **five** $n/3$ bit integers and a few additions/subtractions? You need to compute the following five terms using the square operation only 5 times.

$$P^2, PQ, 2PR + Q^2, QR, R^2.$$

3. Can you find multiplication of two $n$-bit integers, using the multiplication subroutine on **six** pairs of $n/3$ bit integers and a few additions/subtractions? You need to compute the following five terms using the multiplication operation only six times.

$$a_0 b_0, a_1 b_0 + a_0 b_1, a_0 b_2 + a_1 b_1 + a_2 b_0, a_1 b_2 + a_2 b_1, a_2 b_2.$$

   Now, do this with only five multiplications.

4. Solve the recurrences

   - $T(n) = 6T(n/3) + O(n)$.
   - $T(n) = 5T(n/3) + O(n)$.
   - $T(n) = 8T(n/4) + O(n)$.
   - $T(n) = 7T(n/4) + O(n)$.

   Arrange the items in increasing order of complexity.

5. Can you find square of an $n$-bit integer $a$, using square subroutine on **2k-1** integers with $n/k$ bits and a some additions/subtractions? What's the running time you get? What if you take $k$ as something like $n/2$? Does that give you a really fast algorithm?

6. Show that $n2^{\sqrt{logn}}$ is asymptotically smaller than $n^{1.01}$. That is, show that there exists a number $N$, such that for all $n > N$,

$$n2^{\sqrt{logn}} < n^{1.01}.$$

   Do you think it works if we replace 1.01 with any constant greater than 1.

7. Write a program to compare different multiplication algorithms for multiplying 1024 bit integers. You can try the school method, Karatsuba, Toom-cook, or any combination of these.

8. **Matrix Multiplication.** Let us say we have 8 numbers $a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4$ and we consider these **seven** expressions.

$$p_1 = (a_1 + a_4)(b_1 + b_4), \quad p_2 = (a_3 + a_4)b_1, \quad p_3 = a_1(b_2 - b_4), \quad p_4 = a_4(b_3 - b_1)$$
$$p_5 = (a_1 + a_2)b_4, \quad p_6 = (a_3 - a_1)(b_1 + b_2), \quad p_7 = (a_2 - a_4)(b_3 + b_4)$$

**(a)** Compute the following four sums. This will be helpful later.

$$p_1 + p_4 - p_5 + p_7, \quad p_3 + p_5, \quad p_2 + p_4, \quad p_1 - p_2 + p_3 + p_6$$

Now, we want to apply divide and conquer technique to matrix multiplication. Let $A$ and $B$ be two $n \times n$ matrices, and we want to compute their product $C = A \times B$. The naive algorithm for this will take $O(n^3)$ arithmetic operations. We want to significantly improve this using divide and conquer.

A natural way to split any matrix can be this:

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix},$$

where each $A_i$ is an $n/2 \times n/2$ matrix.

**(b)** Can you express the product matrix $C$, in terms of $A_1, A_2, A_3, A_4$ and $B_1, B_2, B_3, B_4$.

**(c)** Design an algorithm for matrix multiplication using divide and conquer which takes $O(\mathbf{7}^{\log_2 n}) = O(n^{\log_2 7}) = O(n^{2.81})$ time.

# Polynomial mulitplication, convolution, Fast Fourier transform

9. Can you apply Karatsuba's trick on polynomial multiplication and get a runtime bound better than $O(d^2)$ for degree $d$ polynomials.

10. Given probability distributions of two (discrete) random variables, we want to compute the probability distribution for the sum of the two random variables. Do you see how convolution can be used to compute this distribution.

11. **Implementation of FFT.** It is possible to do an iterative implementation of FFT, which is "in-place". That is, we can just work on the input array of length $d$ and need only constant size extra memory. Can you think about such an implementation?

12. Let $P(x) = a_0 + a_1 x + \cdots + a_{d-1}x^{d-1}$ be a degree $d-1$ polynomial. Let the $d$th roots of unity be $\omega^0, \omega^1, \ldots, \omega^{d-1}$. Let the evaluations of $P(x)$ on the $d$th roots of unity be $e_0, e_2, \ldots, e_{d-1}$. That is, for each $0 \leq i \leq d-1$
$$e_i = P(\omega^i) = a_0 + a_1\omega^i + \cdots + a_{d-1}\omega^{(d-1)i}.$$
Define a new polynomial $Q(y)$ as $e_0 + e_1 x + \cdots + e_{d-1}x^{d-1}$. Prove that for each $0 \leq i \leq d-1$
$$Q(\omega^{-i}) = e_0 + e_1\omega^{-i} + \cdots + e_{d-1}\omega^{-(d-1)i} = da_i.$$
Hint: Useful facts to prove, $\omega^{-i} = \omega^{d-i}$, For any $1 \leq i \leq d-1$, $\sum_{j=0}^{d-1} \omega^{ij} = 0$.

13. Suppose we want to evaluate $P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ at the fourth roots of unity $1, -1, i, -i$. Consider how FFT algorithm will compute these. We will represent the algorithm as a circuit, with the a gate shown in figure 1. The gate labeled with $\alpha$ takes two numbers $a$ and $b$ (in that order) as inputs and outputs two numbers $a + \alpha b$ and $a - \alpha b$ (in that order).

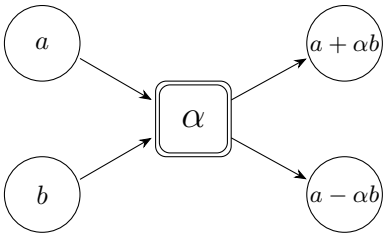



Figure 1: The operation that forms the building block of the FFT algorithm, represented as a gate labeled with a root of unity $\alpha$

The FFT circuit shown in Figure 2 shows the computation for evaluations of a degree 3 polynomial at 4th roots of unity. Fill in the empty circles with the appropriate input values and intermediate values.

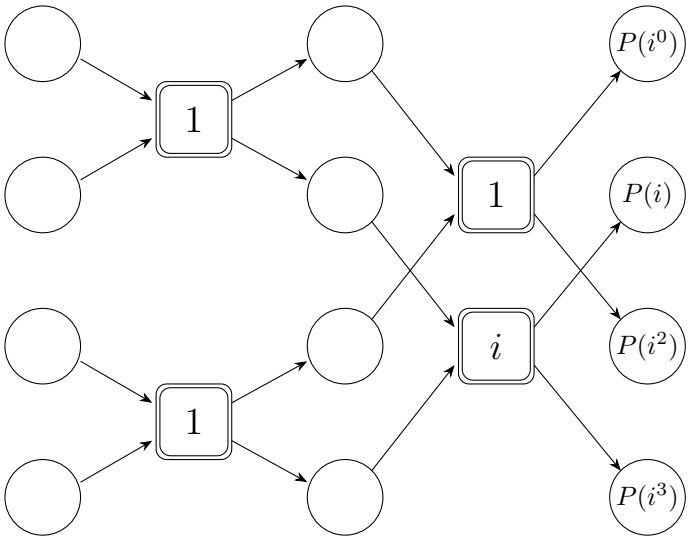



Figure 2: The FFT algorithm for a degree 3 polynomial $P(x)$ evaluated at 4th roots of unity

The FFT circuit shown in Figure 3 shows the computation for evaluations of a degree 7 polynomial at 8th roots of unity. Fill in the empty circles with the appropriate input values and intermediate values.

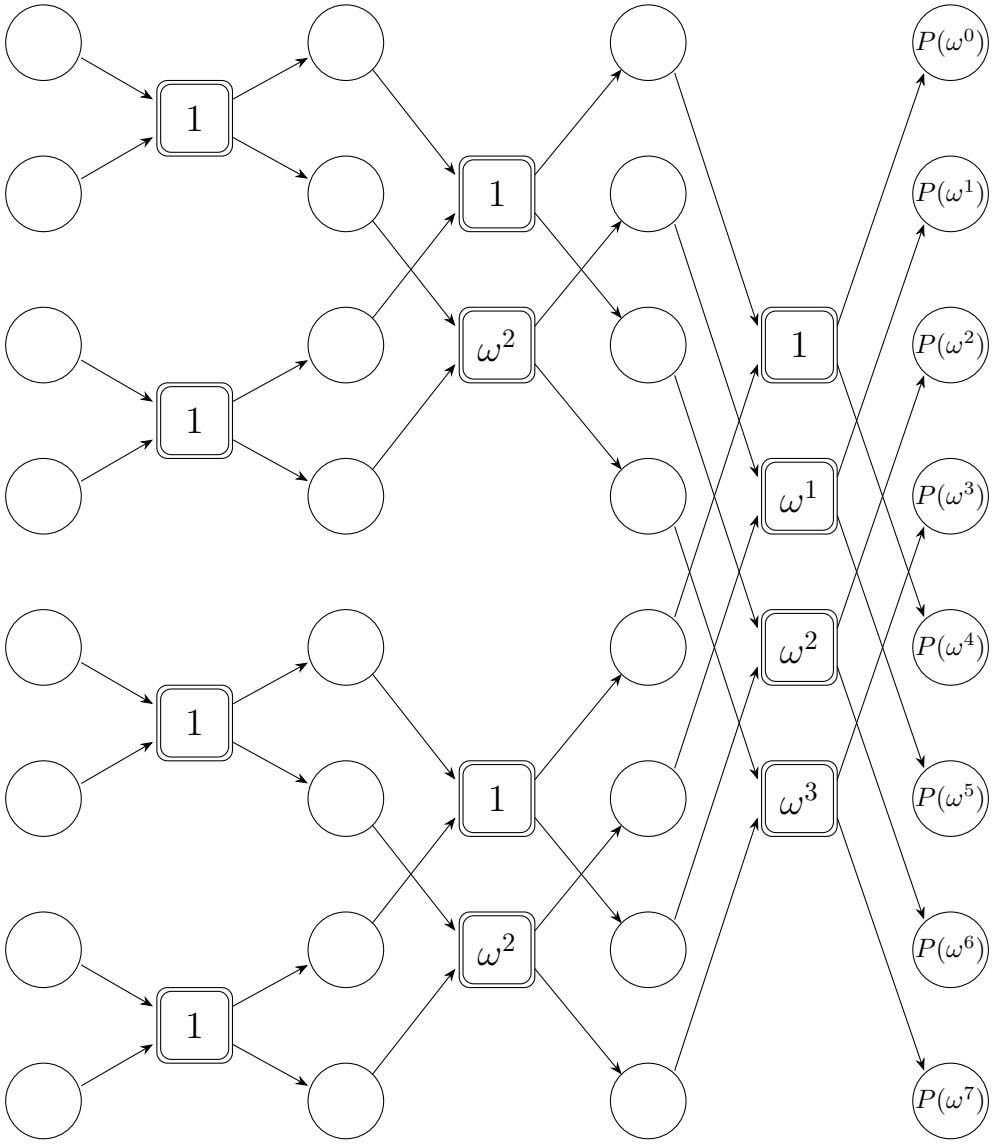



Figure 3: The FFT algorithm for a degree 7 polynomial $P(x)$ evaluated at 8th roots of unity. Here $\omega$ is the primitive 8th root of unity.

**4.** You've been working with some physicists who need to study, as part of their experimental design, the interactions among large numbers of very small charged particles. Basically, their setup works as follows. They have an inert lattice structure, and they use this for placing charged particles at regular spacing along a straight line. Thus we can model their structure as consisting of the points $\{1, 2, 3, \ldots, n\}$ on the real line; and at each of these points $j$, they have a particle with charge $q_j$. (Each charge can be either positive or negative.)

They want to study the total force on each particle, by measuring it and then comparing it to a computational prediction. This computational part is where they need your help. The total net force on particle $j$, by Coulomb's Law, is equal to

$$F_j = \sum_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \sum_{i>j} \frac{Cq_iq_j}{(j-i)^2}$$

They've written the following simple program to compute $F_j$ for all $j$:

```
For j = 1, 2, ..., n
  Initialize F_j to 0
  For i = 1, 2, ..., n
    If i < j then
      Add C q_i q_j / (j - i)^2 to F_j
    Else if i > j then
      Add -C q_i q_j / (j - i)^2 to F_j
    Endif
  Endfor
  Output F_j
Endfor
```

It's not hard to analyze the running time of this program: each invocation of the inner loop, over $i$, takes $O(n)$ time, and this inner loop is invoked $O(n)$ times total, so the overall running time is $O(n^2)$.

The trouble is, for the large values of $n$ they're working with, the program takes several minutes to run. On the other hand, their experimental setup is optimized so that they can throw down $n$ particles, perform the measurements, and be ready to handle $n$ more particles within a few seconds. So they'd really like it if there were a way to compute all the forces $F_j$ much more quickly, so as to keep up with the rate of the experiment.

Help them out by designing an algorithm that computes all the forces $F_j$ in $O(n \log n)$ time.

Figure 4: Kleinberg Tardos: Divide and Conquer Chapter 5, Exercise 4.

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

# Exercises: Greedy, Dynamic Programming

## Greedy algorithms

1. Let $G(V, E)$ be a graph with edge weights and $V = V_1 \cup V_2$ be a partition of vertices. Let $C$ be the set of cut edges connecting $V_1$ with $V_2$, i.e.,

   $$C = \{(u, v) : u \in V_1, v \in V_2\}.$$

   Let $e^*$ be the minimum weight edge in $C$. Prove that there must exists a minimum weight spanning tree containing $e^*$.

2. Let $G(V, E)$ be a graph with edge weights and let $v_1$ be an arbitrary vertex. Let $e^*$ be the minimum weight edge incident on $v_1$. Prove that there must exists a minimum weight spanning tree containing $e^*$.

3. Let $G(V, E)$ be a graph with edge weights. Let $e^*$ be the maximum weight edge. If deletion of $e^*$ disconnects the graph, then $e^*$ must be present in any spanning tree. Else prove that there is a minimum weight spanning tree that does not contain $e^*$. Design an algorithm based on this idea.

4. Interval scheduling: Suppose we have two rooms. And we have a set of requests each with a time interval. We want to allocate the rooms to maximum number of requests. That is, we want to find two subsets $S_1, S_2$ of intervals with $S_1 \cap S_2 = \emptyset$ such that each of two is a set of disjoint intervals and want to maximize $|S_1| + |S_2|$. Prove that the following greedy algorithm will give an optimal solution.

   Initialize $S_1$ and $S_2$ as empty sets. $t_1, t_2$ will denote the largest finish times among the intervals in $S_1$ and $S_2$, respectively. Initially $t_1 = t_2 = 0$. Go over the given intervals in increasing order of finish times. When we are at the $i$-th interval $(a_i, b_i)$,

   - if $a_i < t_1$ and $a_i < t_2$ then discard this interval
   - if $a_i \geq t_1$ and $a_i < t_2$ then put this interval in $S_1$.
   - if $a_i < t_1$ and $a_i \geq t_2$ then put this interval in $S_2$.
   - if $a_i \geq t_1 \geq t_2$ then put this interval in $S_1$.
   - if $a_i \geq t_2 > t_1$ then put this interval in $S_2$.

   Prove the optimality of this greedy algorithm. One way to prove this is as follows. Assume that there is an optimal solution that agrees with the choices made by greedy algorithm on first $i - 1$ intervals. Modify this solution to get another optimal solution that agrees with the choices made by greedy algorithm on first $i$ intervals. Then inductively, you can finish the proof.

5. Suppose you are given a set of $n$ course assignments today, each of which has its own deadline. Let the $i$-th assignment have deadline $d_i$ and suppose to finish the $i$-th assignment it takes $\ell_i$ time. Given that there are so many assignments, it might not be possible to finish all of them on time. If you finish an assignment at time $t_i$ which is more than its deadline $d_i$, then the difference $t_i - d_i$ is called the lateness of this assignment (if $t_i < d_i$ then the lateness is zero). Since you want to maintain a balance among courses, you want that the maximum lateness over all assignments is as small as possible. You want to find a schedule for doing the assignments which minimizes the maximum lateness over all assignments.

   Example: $d_1 = 20, \ell_1 = 10$, $d_2 = 40, \ell_2 = 20$, $d_3 = 60, \ell_3 = 30$. If the assignments are done in order $(1, 3, 2)$ then the maximum lateness will be 20 (for assignment 2). If the assignments are done in order $(1, 2, 3)$ then the maximum lateness will be 0.

   Can you show that a greedy algorithm will give you an optimal solution?

- Greedy Strategy 1: Do the assignments in increasing order of their lengths ($\ell_i$).
- Greedy Strategy 2: Do that assignment first whose deadline is the closest.
- Greedy Strategy 3: Do that assignment first for which $d_i - \ell_i$ is the smallest.

Strategies 1 and 3 don't work. Give examples to show that they don't work.

Strategy 2 works. Prove that it works correctly as follows. Consider any particular schedule for the assignments. Pick any two assignments $A_i$ and $A_j$ which are adjacent, that is, $A_j$ is done immediately after $A_i$. If $d_i \leq d_j$ do then nothing, but if $d_i > d_j$ then swap the positions of $A_i$ and $A_j$. Argue that after this step, the maximum lateness cannot increase.

Repeatedly use the same argument to show that increasing order of deadlines is an optimal schedule.

6. Consider another variant. Now, all assignments are equally long, so let's say each takes a unit time to finish. The $i$th assignment has deadline $d_i$ and a reward $r_i$. You get the reward only if you finish the assignment within the deadline, otherwise the reward is zero. Design an algorithm to find the maximum possible reward you can get.

7. Hard problem. Consider another variant. For each assignment, you know its deadline $d_i$ and the time $\ell_i$ it takes to finish it. Suppose you get zero marks for finishing an assignment after its deadline. So, either you should do the assignment within the deadline or not do it at all. How will you find the maximum number of assignments possible within their deadlines.

8. Given a set of intervals, you need to assign a color to each interval such that no two intersecting intervals have the same color. Design an efficient algorithm find a coloring with minimum number of colors. To put the problem in another way, given arrival and departure times of trains at a station during the day, what is the minimum number of platforms that is sufficient for all trains.

9. For the above problem on minimum number of platforms, consider the following greedy strategy. Mark the platforms as $1, 2, 3, \ldots,$. For each platform $j$, maintain the time $t_j$ at which the last train from that platform departed. Initially, all $t_j = 0$. Sort the trains in increasing order of departure times (finish times). For the $i$-th train with $(arrival, departure) = (a_i, d_i)$, send it to the platform $j$ such that $t_j \leq a_i$ and which has the largest value of $t_j$. Prove or disprove that the algorithm utilizes the minimum number of platforms possible.

10. Given a list of $n$ natural numbers $d_1, d_2, \ldots, d_n$, we want to check whether there exists an undirected graph $G$ on $n$ vertices whose vertex degrees are precisely $d_1, d_2, \ldots, d_n$ (that is the $i$th vertex has degere $d_i$) and construct such a graph if one exists. $G$ should not have multiple edges between the same pair of vertices and should not have self-loop (edges having same vertex as the two endpoints).

    Example 1: (2, 1, 3, 2) . The graph with the set of edges $(v_1, v_3), (v_1, v_4), (v_2, v_3), (v_3, v_4)$ has this degree sequence.

    Example 2: (3, 3, 1, 1) . There is no graph on four vertices having these degrees.

11. Suppose you are an advertisement company who wants to advertise something to all $n$ people in the city. You know that each of these $n$ people will come to the city center on Sunday for some interval of time. You have acquired these time intervals for all people through some unethical means. You cannot put ads at the city center, but you can pay people to carry your ad on them (maybe by wearing a t-shirt). Assume that if $X$ is carrying the ad, then anyone whose time interval intersects with the time interval of $X$ will see the ad (of course, $X$ will also see the ad). You want to choose minimum number of people to whom you should pay so that everyone sees the ad. Design an algorithm for this and prove its correctness.

12. Suppose you have $n$ objects and you are given pairwise distances between those objects $\{d_{i,j} : 1 \leq i < j \leq n\}$. For a clustering of these objects into $k$ clusters, say $S_1 \cup S_2 \cup \cdots \cup S_k = \{1, 2, \ldots, n\}$, we define the spacing of this clustering is the minimum distance between any two objects that are in different clusters. Formally, the spacing is

$$\min\{d_{i,j} : 1 \leq i < j \leq n, i \text{ and } j \text{ belong to different clusters }\}.$$

We want to find a clustering with $k$ clusters that maximizes the spacing. Can you do this in $O(n \log n)$ time? Think first about the $k = 2$ case.

# Dynamic Programming

**Flavour 1: where the subproblems are on suffixes/prefixes of the input.**

13. Work out the remaining details of the problem of finding a set of disjoint intervals that maximizes the total length. Describe an iterative (non-recursive) implementation. Argue that the running time is $O(n \log n)$.

14. The algorithm we described in the class only computed the optimal total length. Update the pseudocode to also output an optimal set of intervals.

15. Design an $O(n \log n)$ time algorithm for the weighted version of the above problem: each interval has a weight and we want to select a subset of disjoint intervals that maximizes the total weight.

16. Design an $O(n \log n)$ time algorithm for the counting version of the above problem, that is, we want to find the number of possible subsets of disjoint intervals.

17. You are going on a car trip from city $A$ to city $B$ that will take multiple days. On the way, you will encounter many cities. You plan to drive only during the day time and on each night you will stay in one of the intermediate cities. Suppose you can drive at most $d$ kilometers in a day. You are given the distances of the intermediate cities from city $A$, say, $d_1, d_2, \ldots, d_n$. And distance of $B$ from $A$ is $d_{n+1}$. You are also given the costs of staying in various cities for one night, say, $c_1, c_2, \ldots, c_n$. Find a travel schedule, that is, in which all cities you should do a night stay, such that your total cost of staying is minimized.

18. We are given a directed graph, where each edge goes from a lower index vertex to a higher index vertex. Want to find the longest path from vertex 1 to vertex $n$.

19. Given a sequence of numbers, find the longest increasing subsequence in $O(n \log n)$ time.

20. Given an array of integers, you want to find a subset with maximum total sum such that no two elements in the subset are adjacent. For example, for the array $\{6, 4, 3, 2, 1, 5\}$, the desired subset is $\{6, 3, 5\}$ with total sum 14. Design an $O(n)$-time algorithm for this problem, where $n$ is the length of the array.

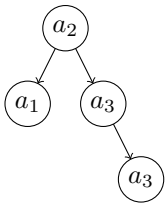
**Flavour 2: where the subproblem is on a substring of the input.**

21. The naive algorithm to multiply two matrices of dimensions $p \times q$ and $q \times r$ takes time $O(pqr)$. Suppose we have four matrices $A, B, C, D$ which are $2 \times 4, 4 \times 3, 3 \times 2, 2 \times 5$ respectively. If you multiply $ABCD$ in the order $(AB)(CD)$, it will take time $2 \times 4 \times 3 + 3 \times 2 \times 5 + 2 \times 3 \times 5 = 84$, on the other hand if you multiply in the order $A((BC)D)$, it will take time $4 \times 3 \times 2 + 4 \times 2 \times 5 + 2 \times 4 \times 5 = 104$.

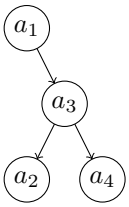
    Given matrices $A_1, A_2, \ldots, A_n$ with dimensions $p_1 \times p_2, p_2 \times p_3, \ldots, p_n \times p_{n+1}$, design an algorithm to find the order in which you should multiply $A_1 A_2 \cdots A_n$ which minimizes the multiplication time.

22. Suppose you have $n$ keys $a_1 < a_2 < \cdots < a_n$ and you want to build a binary search tree that allows efficient search for these keys. If the search queries are distributed uniformly across the keys then it would make sense to build a balanced binary tree. However, if the some keys are more frequent than others then a balanced binary tree might not be the most efficient structure. Consider an example with four keys $a_1 < a_2 < a_3 < a_4$ with the binary search tree shown in Figure 1a.

    Suppose their search frequencies are distributed like $0.7, 0.1, 0.1, 0.1$, respectively. Naturally, we can assume that the search time for a key is proportional to its depth (distance from the root). Then, the average search time for this search tree (Figure 1a) will be $0.7 \times 2 + 0.1 \times 1 + 0.1 \times 2 + 0.1 \times 3 = 2$.




(a) A binary search tree




(b) A binary search tree

Now, consider another search tree in Figure 1b. The average search time for this tree will be $0.7 \times 1 + 0.1 \times 2 + 0.1 \times 3 + 0.1 \times 3 = 1.5$, which is better than the first search tree.

Given search frequencies of $n$ keys, say $f_1, f_2, \ldots, f_n$, we want to find the binary search tree that minimizes the average search time defined as $\sum_i f_i d_i$, where $d_i$ is the depth of the node containing $a_i$.

23. Segmented least squares. Section 6.3 from Kleinberg Tardos

    **Flavour 3: where the subproblem (recursive call) has two different parameters.**

24. A subsequence of a string is obtained by possibly deleting some of the characters without changing the order of the remaining ones. For example, *ephn* is a subsequence of *elephant*.

    You are given a string $A$ of length $n$ and another string $B$ of length $m$ ($\leq n$). If we want to check whether $A$ contains $B$ as a subsequence, there is greedy algorithm for it: For $1 \leq i \leq m$, match the character $B[i]$ with its first occurrence in $A$ after the matching of $B[i-1]$.

    For example, if $A = bacbcbabcacba$ and $B = bcbca$, then the greedy approach will match $B$ as follows (shown in red).

    $$bacbcbabcacba$$

    Now, suppose to match with the $j$-th character in string $A$, you have to pay a cost $p_j$. And to match $B$ with a subsequence in $A$, you have to pay the sum of costs of the matched characters.

    In the above example, if the prices for the characters in $A$ were $2, 5, 3, 1, 2, 5, 3, 1, 3, 1, 2, 4, 1$ then the matching $bacbcbabcacbaa$ has cost $2 + 3 + 1 + 2 + 3 = 11$. While the matching $bacbcbabcacba$ has cost only $1 + 2 + 1 + 2 + 1 = 7$.

    Design an algorithm that given $A, B$ and $p_1, p_2, \ldots, p_n$, can find the minimum cost subsequence in $A$ that can be matched with $B$. Ideally your algorithm should run in time $O(mn)$.

25. **Knapsack problem.** Suppose there are $n$ objects with their weights being $w_1, w_2, \ldots, w_n$ and their values being $v_1, v_2, \ldots, v_n$. You need to select a subset of the objects such that the total weight is bounded by $W$, while the total value is maximized. Your algorithm should run in time $\text{poly}(n, W)$.

    Consider the case when the weights are too large, that is, they are exponential in $n$. Then the above algorithm is not really efficient. Suppose on the other hand, then values $\{v_1, v_2, \ldots, v_n\}$ are small. Can you design an algorithm running in time $\text{poly}(n, \sum_i v_i, \log W)$?

    **Flavour 4: where subproblems are parameterized by a prefix and something more**

26. Suppose you have a movable shop that you can take from one place to another. You usually take your shop to one of the two cities, say A and B, depending on whichever place has more demand. Suppose you have quite accurate projections for the earnings per day in both the cities for the next $n$ days. However, you cannot simply go to the higher earning city each day because it takes one whole day and costs $c$ to move from one city to the other. For example, if you are in city A on day 5 and want to move to city B, then on day 6 you will have no earnings, you will pay a cost of $c$ and on day 7 you will have earnings of city B. Design a polynomial time algorithm that takes as input the moving cost ,the earnings per day in the two cities say, $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, and outputs a schedule for the $n$ days that maximizes the total earnings. Assume that you can start with any of the two cities on day 1, without costing anything.

27. Kleinberg Tardos Section 6.5 RNA secondary structure

28. Kleinberg Tardos Exercises 10, 16, 28

29. Recall the Bellman Ford algorithm seen in the class. How will you find the optimal path?

30. What does Bellman Ford algorithm output if there are negative weight cycles? Can you detect negative cycles if there exists one.

31. For the sequence alignment problem, can you design an algorithm for finding the optimal alignment that takes only $O(m+n)$ space and $O(mn)$ time? Hint: Divide and conquer

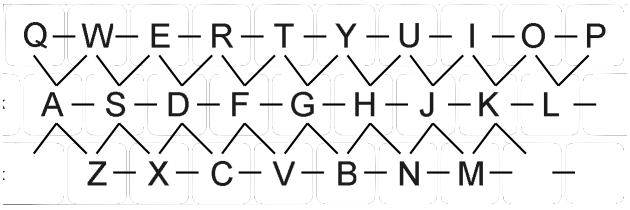



Figure 2: Keyboard with edges showing the adjacent characters.

32. Consider an instance of the edit distance problem where strings are made of english alphabet and the edit costs are as follows.

    - inserting or deleting a character – cost 4
    - substituting a character with a different character
      - if the two characters are adjacent on the keyboard then – cost 3 (see Figure 2, where edges show the pairs of characters which are adjacent)
      - otherwise – cost 6

    Find a way to modify string LINUX to DOUBT with minimum cost.

33. Damerau-Levenshtein distance between two strings (over any alphabet) is defined as the minimum number of valid operations required to convert one string into another, where the valid operations are – deletion of a character, insertion of a character, substituting one character with another, and swapping of two adjacent characters. There is no restriction on the order in which these operations are performed. For example, the two strings `from` and `north` have distance 4, as shown below.

    `from` $\leftrightarrow$ `form` $\leftrightarrow$ `norm` $\leftrightarrow$ `norh` $\leftrightarrow$ `north`

    We want to design a dynamic programming algorithm to compute the Damerau-Levenshtein distance between two given strings. Let the two given strings be $a_1a_2\cdots a_n$ and $b_1b_2\cdots b_m$, where each $a_i$ and $b_j$ belong to an alphabet $\Sigma$.

    Let $d(i,j)$ denote the distance between the substrings $a_1a_2\cdots a_i$ and $b_1b_2\cdots b_j$, for $0 \leq i \leq n$ and $0 \leq j \leq m$ ($i = 0$ or $j = 0$ just means empty substring).

    We will build an $(n+1) \times (m+1)$ table $D$ whose $(i,j)$ entry is supposed to be $d(i,j)$ for $0 \leq i \leq n$ and $0 \leq j \leq m$. Finally, $D(n,m)$ will be our answer.

    Set $D(0,i) = i$ and $D(j,0) = j$ for each $i$ and $j$.

    To fill the rest of the table we use the following equation

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 & \text{(deletion)} \\ D(i,j-1)+1 & \text{(insertion)} \\ D(i-1,j-1) & \text{considered only if } a_i = b_j \\ D(i-1,j-1)+1 & \text{(substitution)} \\ D(i-2,j-2)+1 & \text{considered only if } i,j \geq 2, \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \text{ (swapping).} \end{cases}$$

It turns out that this approach is wrong.

Find an example of two strings, where this algorithm will output a larger number than the Damerau-Levenshtein distance. Fill up the table $D$ completely for your example, as per the above equation. Show the correct distance between the two strings in your example, via a sequence of valid operations.

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

# Exercises: Matchings and Network Flow (No submission)

1. A stable matching is called institute optimal if every institute gets the highest possible preference it can get in any stable matching. Prove that the algorithm we saw in the class was gives an institute optimal stable matching. If you want a student optimal stable matching, what will be your algorithm?

2. Define the notion of stable matchings when vertices have a preference order over only a subset of the other side vertices. The number of vertices need not be equal on the two sides. How will you modify the Gale Shapely algorithm to work in this more general setting. Show that a stable matching always exists.

3. Is it possible that there is no stable matching with size equal to maximum size matching in the graph. The graph we consider here is as follows: there is an edge between $u_i$ and $v_j$ if $u_i$ is in the preference list of $v_j$ and $v_j$ is in the preference list of $u_i$.

4. In the roommate allocation problem, we have a complete (non-bipartite) graph. Every vertex has a preference order over the remaining $n-1$ vertices. Find an example where there is no stable matching.

5. Have a look at the guidelines of the COAP (common offer acceptance portal). Do you think their algorithm always gives a stable matching. The offers are given by institutes to multiple students in parallel. In 2024, they did 10 such rounds of institutes making offers and students retaining/rejecting. How many rounds should be sufficient for all possible scenarios? Do you think there is an alternative approach, where the whole process can be done in one day.

6. For an $s$-$t$ path, the bottleneck is defined to be the least capacity of an edge on that path. Can you design an efficient algorithm to find the $s$-$t$ path with largest bottleneck? Or try this variant: given a threshold $\lambda$, is there an $s$-$t$ path where every edge has capacity at least $\lambda$?

7. Construct a flow network with irrational capacities, where it is possible that the maximum flow algorithm takes infinite iterations. Note that there can be a clever way of choosing $s$-$t$ paths. Here we are just saying that there is a possibility of choosing paths that leads to infinite iterations.

8. Try to design a scaling version of the max flow algorithm, that is, where the number of iterations is proportional to $\log(\sum_e c_e)$.

   Hint: take a parameter $\Delta$, which will initially be large and will become smaller over time. First, ignore all the edges whose capacity is less than $\Delta$, and run the max flow algorithm. When it is no longer possible to increase the flow, then reduce the value of $\Delta$ appropriately and repeat.

9. Use max flow algorithm to solve the following problem. Given a graph with a source vertex $s$ and a destination vertex $t$, find the minimum number of vertices which can be removed to make $s$ and $t$ disconnected.

10. TA allocation: for each TA, we are given a list of suitable courses. For each course we are given the required number of TAs. We want to check whether a TA allocation is possible meeting all course demands, where a TA is only assigned to one of their suitable courses. Design an algorithm for this problem using network flow.

    Use max flow min cut theorem to prove the following: if the above TA allocation is not possible then there must be a subset $S$ of courses whose total demand is larger than the total number suitable TAs for them. This is also known as Hall's theorem.

11. Recall the algorithm we discussed for partitioning a poset (partially ordered set) into minimum number of chains. The algorithm went via the bipartite matching algorithm. First, for the given poset $P$ on $n$ elements, we construct a bipartite graph $G_P$ on $n+n$ vertices, say $\{\ell_1, \ell_2, \ldots, \ell_n\}$ and $\{r_1, r_2, \ldots, r_n\}$. We have an edge $(r_i, \ell_j)$ in $G_P$ if and only if $i \leq j$ in the poset.

    - Prove that from a matching of size $n-k$ in $G_P$, we can get a partition of $P$ into $k$ chains. What happens with the unmatched vertices.
    - Prove that for every partition of $P$ into $k$ chains, there is a corresponding matching of size $n-k$ in $G_P$.
    - Prove the optimality of the obtained partition into chains.

12. Use the max flow min cut theorem to prove Dilworth's theorem. That is, for any partially ordered set, the minimum number of chains in which we can partition the set is equal to the maximum antichain (i.e., maximum number of mutually incomparable elements). You can use the above reduction to bipartite matching and then a reduction to max flow. Assume max flow is equal to $s$-$t$-minimum-cut. Then given an $s$-$t$ cut with outgoing capacity of $n-k$, try to construct an antichain of size $k$.

13. Vertex Cover: a set of vertices $S$ such that every edge in the graph has at least one end point in $S$. Observe that for any vertex cover $S$ and any matching $M$, we have $|S| \geq |M|$.

    Use max flow min cut theorem to prove that in bipartite graphs, maximum matching size is equal to minimum vertex cover size.

14. **Project selection** We are given a set of projects, where some projects are pre-requisites for others. This can be represented by a direct acylic graph on projects. An edge from $i$ to $j$ represents that $i$ is a pre-requisite for $j$. Clearly this is a transitive relation. A project can be done only if all its pre-requisite projects have been done.

    Some projects might have a positive value, i.e., you gain a net profit from them. While some other projects may have a negative value, i.e., you have to invest more into them than what you gain from them. The goal is to select a subset of projects which maximizes the total value, under the constraint that if a project is selected then all its pre-requisites must also be selected.

    Reduce this problem to $s$-$t$-minimum-cut problem. That is design an algorithm which can use $s$-$t$-minimum-cut subroutine. Note that minimum cut is also a subset (of vertices) selection problem.

15. Tiger counting: The tiger population in the country is counted once in every four years. To count their number, cameras are installed in the areas visited by them. Tigers are uniquely identified by their stripes. Interestingly, the stripes on the two sides of a tiger are not correlated. Hence, they always install cameras in pairs, where the two cameras face each other. This way they can get pictures from both the sides of the tigers. Sometimes there are obstacles or some malfunctioning, due to which they get only one side of the tiger. Suppose they get (pairs of) images where they have both sides of $k$ distinct tigers. And additionally, they have some images of the left stripes of $m$ distinct tigers and some images of the right stripes of $n$ distinct tigers. When there is no way to figure out which of the left images and right images belong to the same tiger, they take a conservative estimate, which is $k + \max\{m, n\}$.

    At some point, allegedly to show a higher count, they changed the protocol and counted the above scenario as $k+m+n$ distinct tigers. Clearly, the actual count is somewhere between $k + \max\{m, n\}$ and $k+m+n$.

    Let's say there are some experts who can identify which pairs of left and right images are definitely from distinct tigers. For example, for $m=3$, $n=3$, suppose they say that the pairs $(\ell_1, r_2)$, $(\ell_1, r_3)$, $(\ell_2, r_2)$, $(\ell_2, r_3)$ are definitely from distinct tigers. With this additional information, the conservative estimate will be 4 distinct tigers. How will you find this conservative estimate for any given list of pairs from experts.

16. Suppose there are $n$ thermal power plants and $m$ coal mines in the country. The coal is transported to power plants via the train network. Each segment (edge) of the train network has a certain transport capacity per day. Each coal mine has a maximum limit on the amount of coal that can be produced in one day. Each thermal power plant has a certain demand for coal per day. Given all this, we want to decide the coal transportation plan. That is, from which coal mine, how much coal should be sent to which power plant, using which rail route, so that we meet all the demands.

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

# Exercises: NP, NP-completeness

**Independent set:** given a graph and a number $k$, is there a set of $k$ vertices such that there is no edge between them?

**Vertex Cover:** given a graph and a number $k$, is there a set $S$ of $k$ vertices which covers all edges (i.e., every edge in the graph has at least one end point in $S$).

**Set Cover:** Suppose we are given a collection of subsets of a set $U$, say $S_1, S_2, \ldots, S_r \subseteq U$. Can we select $k$ of these subsets such that their union is $U$?

**Directed Hamiltonian cycle:** Given a directed graph with $n$ vertices, is there a (directed) cycle of length $n$? By definition, a cycle cannot have repeated vertices.

1. Which of the following decision problems do you think are in NP?
   - Given a graph $G$ and a number $k$, is the minimum vertex cover size in $G$ equal to $k$?
   - We are given a directed graph, a source vertex, a destination vertex, and a number $k$. Is $k$ the minimum number vertices required to be deleted to disconnect source from destination?
   - Given two C++ programs, are they equivalent (do they have same input output behavior)?
   - Given two C++ programs, are they different (is there an input where they give different outputs)?
   - Given a fully quantified Boolean formula (every variable is quantified with either $\exists$ or $\forall$ in some order), is it true?
2. Reduce the independent set problem to the vertex cover problem.
3. Reduce vertex cover problem to the set cover problem.
4. Suppose the following version of knapsack problem is NP-complete. Given a set of integer weights $w_1, w_2, \ldots, w_n$ and target weights $W_1, W_2$, is there a subset $S$ of the weights whose sum is between $W_1$ and $W_2$, i.e., $W_1 \leq \sum_{i \in S} w_i \leq W_2$?

   Using this fact, prove that the following load balancing problem is NP-complete. Given a set of integer loads $t_1, t_2, \ldots, t_n$ and a target makespan $T$, is there a way to distribute all the loads to two machines so that the maximum load on any machine is at most $T$?
5. *Integer programming*: Given a set of linear inequalities in variables $x_1, x_2, , x_n$, decide if there is an integer solution satisfying all of them simultaneously. For example the set

$$0 \leq x_1, x_2 \leq 1$$
$$x_1 - x_2 \geq 0.$$

   has an integer solution $(1, 0)$ (and also $(1, 1)$, $(0, 0)$).

   Show that Integer Programming is NP-hard. You can try a reduction from SAT to this problem. Given a CNF Booelan formula $\phi$, you need to generate a set $S$ of linear inequalities and prove that $\phi$ is satisfiable if and only if the set $S$ has an integer solution.
6. Reduce SAT problem to Directed Hamiltonian cycle problem.
7. Consider the 3-coloring problem, where we are given a graph and we have to decide if we can color the vertices of the graph with 3 colors, such that any two neighboring vertices get different colors. Reduce this problem to the independent set problem.

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

Exercises: Linear Programming, Approximation, Randomized, Error correction

1. (Line fitting.) Given $n$ points $p_1, p_2, \ldots, p_n \mathbb{R}^d$, with labels $\ell_1, \ell_2, \ldots, \ell_n \in \mathbb{R}$, we want to compute a linear function that best fits with the points and labels. More precisely, find a function $h(x) = a_1x_1 + a_2x_2 + \cdots + a_dx_d + b$ so that we minimize the error function $E(h)$ defined as

$$E(h) = \max_{1 \le j \le n} \{|h(p_j) - \ell_j|\}.$$

   Write a linear program for this.

2. (Curve fitting.) Given $n$ points $p_1, p_2, \ldots, p_n \mathbb{R}^d$, with labels $\ell_1, \ell_2, \ldots, \ell_n \in \mathbb{R}$, we want to compute a quadratic function that best fits with the points and labels. More precisely, find a function $h(x) = \sum_{1 \le i \le j \le d} a_{i,j}x_ix_j$ so that we minimize the error function $E(h)$ defined as

$$E(h) = \max_{1 \le j \le n} \{|h(p_j) - \ell_j|\}.$$

   Write a linear program for finding $h$.

3. (Classification.) Given $n$ points $p_1, p_2, \ldots, p_n \mathbb{R}^d$, which are labeled either positive or negative, we want to compute a linear function that best fits with the points and labels. More precisely, find a function $h(x) = a_1x_1 + a_2x_2 + \cdots + a_dx_d + b$ so that we minimize the hing loss $L(h)$ defined as follows.

   For a point $p_j$, if it's label is positive then we expect $h(p_j)$ to be (significantly) more than zero. Let's say we expect $h(p_j)$ to be at least 1. If that is not true then we consider the difference from 1 as the loss. Define loss with respect to a positively labeled point $p_j$ as

$$L(h, p_j) \begin{cases} = 1 - h(p_j) & \text{if } h(p_j) < 1 \\ = 0 & \text{otherwise.} \end{cases}$$

   Similarly, define loss with respect to a negatively labeled point $p_k$ as

$$L(h, p_k) \begin{cases} = h(p_k) + 1 & \text{if } h(p_j) > -1 \\ = 0 & \text{otherwise.} \end{cases}$$

   Finally we define the hinge loss $L(h)$ over all points as $\sum_j L(h, p_j)$. Write a linear program to find $h$ that minimizes $L(h)$.

4. (Simplex algorithm simulation) Consider the following linear program.

$$\begin{aligned} \max 2x_1 + x_2 \text{ subject to} \\ x_1 &\ge 0 \\ x_2 &\ge 0 \\ x_1 &\le 1 \\ x_1 + x_2 &\le 2 \end{aligned}$$

   - First write the LP in the standard form where we have (two) equations and (four) non-negativity constraints $x_1 \ge 0, x_2 \ge 0, x_3 \ge 0, x_4 \ge 0$.

- Start with the basic feasible solution that has $x_1 = 0, x_2 = 0$. What will be the values of $x_3, x_4$? At any point in the algorithm, the two variables which are zero are called the *non-basic variables* and the remaining are called *basic variables*. For example, in the beginning, $x_3, x_4$ are basic variables and $x_1, x_2$ are non-basic.
- Run the iterations of the simplex algorithm till you get an optimal solution. After each iteration write down (i) the basic feasible solution and (ii) express the objective function in terms of the current non-basic variables (i.e., which are zero). In an iteration, there may be multiple possible choices for which variable to increase. You can just make an arbitrary choice.

Below is how an iteration of simplex algorithm runs.

- Choose one of the non-basic variables to increase. It should be among those whose coefficient in the objective function is positive. If there is no such variable then output the current solution. If there is such a variable then increase it till the maximum possible value while maintaining feasibility. The other non-basic variables (except the chosen one) should remain zero in this iteration.
- This gives you a new basic feasible solution and a new set of basic and non-basic variables
- Express the objective function in terms of the non-basic variables.
- Express the basic variables in terms on non-basic variables.

5. Prove that when the simplex algorithm stops, that is, when we express the objective function in terms of non-basic variables and all coefficients turn out to be negative, then we are at an optimal solution. What is the optimal value at this point?

6. (Initial basic feasible solution.) We said that finding initial basic feasible solution itself is a challenging problem. We will solve it by framing it as another linear program. Suppose the given linear program (LP1) is

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ &\vdots \\ a_{k,1}x_1 + a_{k,2}x_2 + \cdots + a_{k,n}x_n &= b_k \\ x_1, x_2, \ldots, x_n &\geq 0. \end{aligned}$$

Here $a_{i,j}$s and $b_i$s are given as input and $x_j$s are unknowns. Without loss of generality, we can assume that $b_1, b_2, \ldots, b_k \geq 0$. Because otherwise we can simply multiply -1 on both the sides of the equation.

We write the following new linear program (LP2).

$$\begin{aligned} \min z_1 + z_2 + \cdots + z_k \text{ subject to} \\ a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n - b_1 &= z_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n - b_2 &= z_2 \\ &\vdots \\ a_{k,1}x_1 + a_{k,2}x_2 + \cdots + a_{k,n}x_n - b_k &= z_k \\ x_1, x_2, \ldots, x_n &\geq 0. \\ z_1, z_2, \ldots, z_k &\geq 0. \end{aligned}$$

Here $z_i$s and $x_j$s are unknowns.

Prove that LP1 has a feasible solution if and only if LP2 has an optimal value =0.

There is a trivial initial basic feasible solution for LP2, which is $z_1 = b_1, z_2 = b_2, \ldots, z_k = b_k$ and $x_1 = x_2 = \cdots = x_n = 0$.

7. Prove that the following algorithm gives 2-approximation for minimum size vertex cover. That is, the set $S$ output by the algorithm is a vertex cover and its size is at most twice of the optimal vertex cover.

---

$S \leftarrow$ empty set.

While the graph is non-empty

choose an edge $(u, v)$ and put both its endpoints in $S$

delete $u$ and $v$ and all their incident edges from the graph

delete isolated vertices

---

Observe that the edges chosen during the algorithm form a matching in the given graph. Prove that this is an 1/2-approximation for maximum matching. That is, the matching obtained has size at least half of the maximum size matching.

8. Consider the following (approximation) algorithm for minimum size vertex cover. Construct examples to show that the approximation factor is not bounded by any constant, that is, the approximation factor increases with the input size.

---

$S \leftarrow$ empty set.

While the graph is non-empty

choose a vertex $u$ with the highest degree and put it in $S$

delete $u$ and all its incident edges from the graph

delete isolated vertices

---

9. **Maximum weight matching:** Given a graph with edge weights, the goal is to find a matching (set of disjoint edges) with maximum total weight. Write an integer linear program for the maximum weight matching problem. Now, remove the integer constraint, that is, variables are allowed to take any real value. We get a linear program. Find an example (a graph with edge weights), where the optimal value of the linear program is higher than the weight of the maximum weight matching. Interestingly, if the graph is bipartite then the two values are always equal.

10. Recall the greedy algorithm for minimum makespan problem. Prove that the analysis for 2-approximation we did was tight. That is, find an example where the makespan given by the greedy algorithm is roughly twice of the optimal makespan.

11. Consider a variant of the greedy algorithm, where we go over the job in decreasing order of processing times. Prove that this variant gives a 3/2-approximate solution.

    Hint: consider load on any processor. Split it into two parts: the last job assigned to that processor and the remaining jobs assigned to that processor. The total load of the remaining jobs is upper bounded by the average load per processor, because it must have been the minimum load among all processors at that time. Show that the last job has processing time at most 1/2 of the optimal makespan (assuming the processor was assigned at least two jobs).

12. Do you think your analysis above is tight? Do you see an example, where the solution obtained is indeed 3/2 times the optimal?