Algorithms (CS2443) : Problem Set 2

Department of Computer Science and Engineering
IIT Hyderabad

Please read the following comment before you work on the problems.

This is a set of questions that you should solve to master the contents taught in the class. *You don't have to submit the solutions*! But you should still solve them to understand the material, and also because these problems or their variants have a devilish way of finding their way in exams/quizzes!

We suggest to try to solve them from scratch under exam conditions—by yourself, *without* your notes, *without* the internet, and if possible, even without a cheat sheet.

If you find yourself getting stuck on a problem, try to figure out *why* you're stuck. Do you understand the problem statement? Are you stuck on choosing the right high-level approach, or are you stuck on the details? Are you solving the right (recursive) subproblems?

Discussing problems with other people (in your study groups, with me, or on Google Classroom) and/or looking up old solutions can be *extremely* helpful, but **only after** you have (1) made a good-faith effort to solve the problem on your own, and (2) you have either a candidate solution or some idea about where you are getting stuck.

When you do discuss problems with other people, remember that your goal is not merely to "understand" the solution to any particular problem, but to become more comfortable with solving a certain type of problem on your own. Identify specific steps that you find problematic, read more about those steps, and focus your practice on those steps.

---

1. Suppose you are given an array $A[1, \ldots, n]$ of numbers which may be positive, negative, or zero, and which are **not** necessarily integers.

   (a) Describe and analyze an algorithm that finds the largest sum of elements in a contiguous subarray $A[i, \ldots, j]$.

   (b) Suppose A is a *circular* array. In this setting, a "contiguous subarray" can be either an interval $A[i, \ldots, j]$ or a suffix followed by a prefix $A[i, \ldots, n] \cdot A[1, \ldots, j]$. Describe and analyze an algorithm that finds a contiguous subarray of A with the largest sum.

For example, given the array $[-4, 12, -7, 0, 14, -7, 5]$ as input, your first algorithm should return 19 (which is given by $A[2, \ldots, 5]$), and your second algorithm should return 20 (which is given by $A[5, 6, 7] \cdot A[1, 2]$). Given the one-element array $[-374]$ as input, both algorithms should return 0. (The empty interval is still an interval!) For the sake of analysis, assume that comparing, adding, or multiplying any pair of numbers takes $O(1)$ time.

2. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.

   Having never taken an algorithms class, Elmo follows the obvious greedy strategy–when it's his turn, Elmo *always* takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely hates it when grown-ups let him win.)

   (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do not follow the same greedy strategy as Elmo.

   (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

   (c) When Elmo was four, he used an even simpler strategy–on his turn, he always chose his next card uniformly at random. That is, if there was more than one card left on his turn, he would take the leftmost card with probability 1/2, and the rightmost card with probability 1/2. Describe an algorithm to determine, given the initial sequence of cards, the maximum *expected* number of points you can collect playing against four-year-old-Elmo.

   (d) Five years later, thirteen-year-old Elmo has become a *much* stronger player. Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against a *perfect* opponent.

3. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the list of n songs that the judges will play during the contest, in chronological order. Yesssssssssss!

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will

be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k+Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1, \ldots, n]$ and $Wait[1, \ldots, n]$.

4. The new swap-puzzle game *Candy Swap Saga XIII* involves n cute animals numbered from 1 to n. Each animal holds one of three types of candy: circus peanuts, Heath bars, and Cioccolateria Gardini chocolate truffles. You also have a candy in your hand; at the start of the game, you have a circus peanut.

   To earn points, you visit each of the animals in order from 1 to n. For each animal, you can either keep the candy in your hand or exchange it with the candy the animal is holding.

   - If you swap your candy for another candy of the *same* type, you earn one point.
   - If you swap your candy for a candy of a *different* type, you lose one point. (Yes, your score can be negative.)
   - If you visit an animal and decide not to swap candy, your score does not change.

   You must visit the animals in order, and once you visit an animal, you can never visit it again.

   Describe and analyze an efficient algorithm to compute your maximum possible score. Your input is an array $C[1, \ldots, n]$, where $C[i]$ is the type of candy that the i-th animal is holding.

5. Suppose we are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, and all $2n$ endpoints are distinct.

   (a) Describe and analyze an algorithm to compute the largest subset of L in which no pair of segments intersects.

   (b) Describe and analyze an algorithm to compute the largest subset of L in which **every** pair of segments intersects.

   Now suppose we are given a set L of n line segments in the plane, where both endpoints of each segment lie on the unit circle $x^2 + y^2 = 1$, and all $2n$ endpoints are distinct.

   (c) Describe and analyze an algorithm to compute the largest subset of L in which no pair of segments intersects.

   (d) Describe and analyze an algorithm to compute the largest subset of L in which **every** pair of segments intersects.

6. Suppose you are given an $m \times n$ bitmap as an array $M[1 \ldots n, 1 \ldots n]$ of 0s and 1s. A *solid block* in M is a subarray of the form $M[i \ldots i', j \ldots j']$ in which all bits are equal. A solid block is square if it has the same number of rows and columns.

   (a) Describe an algorithm to find the maximum area of a solid square block in M in $O(n^2)$ time.

   (b) Describe an algorithm to find the maximum area of a solid block in M in $O(n^3)$ time.

   (c) Describe an algorithm to find the maximum area of a solid block in M in $O(n^2 \log n)$ time. [Hint: Divide and conquer.]

   (d) Describe an algorithm to find the maximum area of a solid block in M in $O(n^2)$ time.

7. Suppose you are given an array $M[1 \ldots n, 1 \ldots n]$ of numbers, which may be positive, negative, or zero, and which are not necessarily integers. Describe an algorithm to find the largest sum of elements in any rectangular subarray of the form $M[i \ldots i', j \ldots j']$. For full credit, your algorithm should run in $O(n^3)$ time. [Hint: See Problem 1.]

8. A **basic arithmetic expression** is composed of characters from the set $\{1, +, \times\}$ and parentheses, where $+, \times$ are binary operations. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$(((1+1)+(1+1))+((1+1)+(1+1)))+(((1+1)+(1+1))+((1+1)+(1+1)))$$

$$((1+1)\times((1+1)+(1+(1+1))))+((1+1)\times(1+1))$$

$$(1+1)\times((1+(1+1))+((1+1)+(1+1)))$$

$$(1+1)\times(((1+(1+1))\times(1+1))+1)$$

   Describe and analyze an algorithm to compute, given an integer n as input, the minimum number of 1s in a basic arithmetic expression whose value is equal to n. For example, when $n = 14$, your algorithm should return 8, for the final expression above. The running time of your algorithm should be bounded by a small polynomial function of n.

9. Suppose you are given a sequence of integers separated by + and - signs; for example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7$$

   You can change the value of this expression by adding parentheses in different places. For example:

$$((((((1+3)-2)-5)+1)-6)+7) = -1$$

$$(1+(3-(2-5)))+((1-6)+7) = 9$$

$$((1+(3-2))-(5+1))-(6+7) = -17$$

Describe and analyze an algorithm to compute, given a list of integers separated by + and - signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$.

10. After graduating from Sham-Poobanana University, you decide to interview for a position at the Wall Street bank ***Long Live Boole***. The managing director of the bank, Eloob Egroeg, poses a 'solve-or-die' problems to each new employee, which they must solve within 24 hours. Those who fail to solve the problem are fired immediately!

    Entering the bank for the first time, you notice that the employee offices are organized in a straight row, with a large T or F printed on the door of each office. Furthermore, between each adjacent pair of offices, there is a board marked by one of the symbols $\wedge$, $\vee$, or $\oplus$. When you ask about these arcane symbols, Eloob confirms that T and F represent the Boolean values True and False, and the symbols on the boards represent the standard Boolean operators AND, OR, and XOR. He also explains that these letters and symbols describe whether certain combinations of employees can work together successfully. At the start of any new project, Eloob hierarchically clusters his employees by adding parentheses to the sequence of symbols, to obtain an unambiguous Boolean expression. The project is successful if this parenthesized Boolean expression evaluates to T.

    For example, if the bank has three employees, and the sequence of symbols on and between their doors is $T \wedge F \oplus T$, then both $((T \wedge F) \oplus T)$ and $(T \wedge (F \oplus T))$ are successful parenthesization scheme. However, if the list of door symbols is $F \wedge T \oplus F$, there is no way to add parentheses to make the project successful.

    Eloob finally poses your solve-or-die interview question: Describe an algorithm to decide whether a given sequence of symbols can be parenthesized so that the resulting Boolean expression evaluates to T. Your input is an array $S[0, \ldots, 2n]$, where $S[i] \in \{T, F\}$ when i is even, and $S[i] \in \{\vee, \wedge, \oplus\}$ when i is odd.

11. An **independent set** in a graph is a subset of the vertices with no edges between them. Finding the largest independent set in an arbitrary graph is extremely hard; in fact, this is one of the canonical NP-complete problems which you may encounter if you take the complexity course next semester! But in some special classes of graphs, we can find largest independent sets quickly.

    Describe and analyze an algorithm to compute the largest independent set in $O(n)$ time when the input is a *tree* with n nodes.

12. The greedy algorithm for Interval Scheduling that we described in the class is not the only greedy strategy we could have tried. For each of the following alternative greedy strategies, either prove that the resulting algorithm always constructs an optimal schedule, or describe a small example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a

manner that is completely out of your control). [Hint: Three of these algorithms are actually correct.]

(a) Choose an interval x that *ends last*, discard intervals that conflict with x, and recurse.

(b) Choose an interval x that *starts first*, discard all intervals that conflict with x, and recurse.

(c) Choose an interval x that *starts last*, discard all intervals that conflict with x, and recurse.

(d) Choose an interval x with *shortest duration*, discard all intervals that conflict with x, and recurse.

(e) Choose an interval x that *conflicts with the fewest other intervals*, discard all intervals that conflict with x, and recurse.

(f) If no intervals conflict, choose them all. Otherwise, discard an interval with the *longest duration* and recurse.

(g) If no intervals conflict, choose them all. Otherwise, discard an interval that conflicts *with the most other intervals* and recurse.

(h) Let I_1 be the interval with the *earliest start time*, and let I_2 be the interval with the *second earliest start time*.

- If I_1 and I_2 are disjoint, choose I_1 and recurse on everything but I_1.
- If I_1 completely contains I_2, discard I_1 and recurse.
- Otherwise, discard I_2 and recurse.

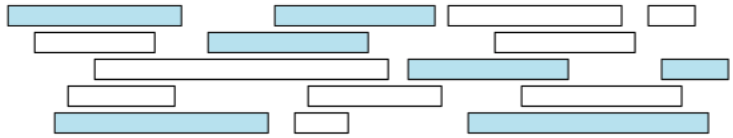
(i) If any interval I_1 completely contains another interval, discard I_1 and recurse. Otherwise, choose the interval I_2 that *ends last*, discard all intervals that conflict with I_2, and recurse.

13. Now consider a weighted version of the interval scheduling problem, where each interval has a certain *weight*. Your goal is now to choose a set of non-overlapping intervals that give you the largest possible weight, given arrays of start times, end times, and weights as input.

- Prove that the greedy algorithm we saw for the interval scheduling problem – Choose the interval that ends first and recurse – does *not* always return an optimal schedule.
- Describe and analyze an algorithm that always computes an optimal schedule. [Hint: Your algorithm will not be greedy. Think DP. A greedy algorithm for this problem is an open problem!]

14. Let X be a set of n closed intervals on the real line. We say that a subset of intervals $Y \subseteq X$ *covers* X if the union of all intervals in Y is equal to the union of all intervals in X. The *size* of a cover is just the number of intervals.

Give an efficient algorithm to compute the *smallest* cover of X. Assume that your input consists of two arrays $L[1, \ldots, n]$ and $R[1, \ldots, n]$, representing the left and right endpoints of the intervals in X.



A set of intervals, with a cover (shaded) of size 7.

15. Let X be a set of n intervals on the real line. We say that a set P of points *stabs* X if every interval in X contains at least one point in P (see Fig. 1). Describe and analyze an efficient algorithm to compute the smallest set of points that stabs X. Assume that your input consists of two arrays $L[1, \ldots, n]$ and $R[1, \ldots, n]$, representing the left and right endpoints of the intervals in X.

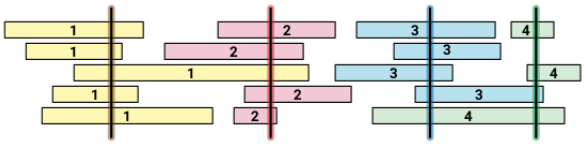


Figure 1: A set of intervals stabbed by four points (shown here as vertical segments).

16. Let X be a set of n intervals on the real line. A *proper coloring* of X assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe and analyze an efficient algorithm to compute the minimum number of colors needed to properly color X. Assume that your input consists of two arrays $L[1, \ldots, n]$ and $R[1, \ldots, n]$, representing the left and right endpoints of the intervals in X.

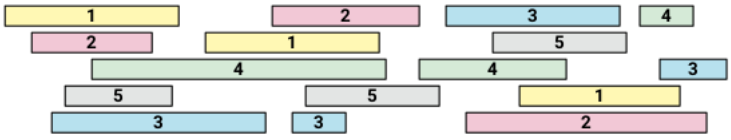


Figure 2: A proper coloring of a set of intervals using five colors.

17. For every integer $n \geq 2$,

    (a) Find frequencies for a set of n letters, say $\{a_1, a_2, \ldots, a_n\}$, whose Huffman code tree has depth $n - 1$.

    (b) Find frequencies for a set of n letters whose Huffman code tree has depth $n - 1$ and the largest frequency is as small as possible.

18. (a) Prove that it is possible for the Gale-Shapley algorithm to perform $\Omega(n^2)$ offers before termination. (You need to describe both a suitable input and a sequence of $\Omega(n^2)$ valid offers.)

    (b) Describe for any integer n a set of preferences for n doctors and n hospitals that forces the Gale-Shapley algorithm to execute $\Omega(n^2)$ rounds, *no matter which valid proposal is made in each round.*

19. Describe and analyze an efficient algorithm to determine whether a given set of men and women preferences has a *unique* stable matching.

20. Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \cdots < c[n]$. (In the U.S., for example, n = 6 and the values are 1, 5, 10, 25, 50 and 100 cents.) Your beloved and benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.

    (a) In the United States, there is a simple greedy algorithm that always results in the smallest number of coins: subtract the largest coin and recursively give change for the remainder. El Generalissimo does not approve of American capitalist greed. Show that there is a set of coin values for which the greedy algorithm does not always give the smallest possible of coins.

    (b) Now suppose El Generalissimo decides to impose a currency system where the coin denominations are consecutive powers $b^0, b^1, b^2, \ldots, b^k$ of some integer $b \geq 2$. Prove that despite El Generalissimo's disapproval, the greedy algorithm described in part (a) does make optimal change in this currency system.

    (c) Describe and analyze an efficient algorithm to determine, given a target amount T and a sorted array $c[1, \ldots, n]$ of coin denominations, the smallest number of coins needed to make T cents in change. Assume that $c[1] = 1$, so that it is possible to make change for any amount T.

21. Suppose you are given an array $A[1, \ldots, n]$ of integers, each of which may be positive, negative, or zero. A contiguous subarray $A[i, \ldots, j]$ is called a **positive-interval** if the sum of its entries is greater than zero. Give an efficient algorithm to compute the minimum number of positive-intervals that cover every positive entry in A. For example, given the following array as input, your algorithm should output 3. If every entry in the input array is negative, your algorithm should output 0.

3	−5	7	−4	1	−8	3	−7	5	−9	5	−2	4

    The three positive-intervals in the above array are $A[1, 2, 3, 4, 5]$, $A[7, 8, 9]$, and $A[11, 12, 13]$.

22. Consider the following process. At all times you have a single positive integer x, which is initially equal to 1. In each step, you can either *increment* x or *double* x. Your goal is to produce a target value n. For example, you can produce the integer 10 in four steps as follows:

$$1 \xrightarrow{+1} 2 \xrightarrow{\times 2} 4 \xrightarrow{+1} 5 \xrightarrow{\times 2} 10.$$

Obviously you can produce any integer n using exactly $n-1$ increments, but for almost all values of n, this is horribly inefficient. Give an efficient algorithm to compute the minimum number of steps required to produce any given integer n.

23. Suppose we have n skiers with heights given in an array $P[1, \ldots, n]$, and n skis with heights given in an array $S[1, \ldots, n]$. Describe an efficient algorithm to assign a ski to each skier, so that the average difference between the height of a skier and her assigned ski is as small as possible. The algorithm should compute a permutation σ such that the expression

$$\frac{1}{n} \sum_{i=1}^{n} |P[i] - S[\sigma(i)]|$$

is as small as possible.

24. Suppose we are given two arrays $C[1, \ldots, n]$ and $R[1, \ldots, n]$ of positive integers. An $n \times n$ matrix of 0s and 1s agrees with R and C if, for every index i, the i-th row contains $R[i]$ 1s, and the i-th column contains $C[i]$ 1s. Describe and analyze an algorithm that either constructs a matrix that agrees with R and C, or correctly reports that no such matrix exists.

25. Alice wants to throw a party and she is trying to decide who to invite. She has n people to choose from, and she knows which pairs of these people know each other. She wants to invite as many people as possible, subject to two constraints:

    - For each guest, there should be at least five other guests that they already know.
    - For each guest, there should be at least five other guests that they don't already know.

    Describe and analyze an algorithm that computes the largest possible number of guests Alice can invite, given a list of n people and the list of pairs who know each other.