#484.	17.26	Solution 1: Start with just a simple algorithm comparing all documents to all other documents. How would you compute the similarity of two documents as fast as possible?
#485.	17.5	It doesn't really matter which letter or number it is. You can simplify this problem to just having an array of As and Bs. You would then be looking for the longest subarray with an equal number of As and Bs.
#486.	17.11	Consider first the algorithm for finding the closest distance if you will run the algorithm only once. You should be able to do this in O(N) time, where N is the number of words in the document.
#487.	16.20	Can you recursively try all possibilities?
#488.	17.9	Be clear about what this problem is asking for. It's asking for the kth smallest number in the form $3^a * 5^b * 7^c$.
#489.	16.2	Think about what the best conceivable runtime is for this problem. If your solution matches the best conceivable runtime, then you probably can't do any better.
#490.	16.10	Solution 1: Try using a hash table, or an array that maps from a birth year to how many people are alive in that year.
#491.	16.14	Sometimes, a brute force is a pretty good solution. Can you try all possible lines?
#492.	16.1	Try picturing the two numbers, a and b, on a number line.
#493.	17.7	The core part of the problem is to group names into the various spellings. From there, figuring out the frequencies is relatively easy.
#494.	17.3	If you haven't already, solve 17.2 on page 186.
#495.	17.16	There are recursive and iterative solutions to this problem, but it's probably easier to start with the recursive solution.
#496.	17.13	Try a recursive approach.
#497.	16.3	Infinite lines will almost always intersect—unless they're parallel. Parallel lines might still "intercept"—if they're the same lines. What does this mean for line segments?
#498.	17.26	Solution 1: To compute the similarity of two documents, try reorganizing the data in some way. Sorting? Using another data structure?
#499.	17.15	If we wanted to know just the longest word made up of other words in the list, then we could iterate over all words, from longest to shortest, checking if each could be made up of other words. To check this, we split the string in all possible locations.
#500.	17.25	Can you find a word rectangle of a specific length and width? What if you just tried all options?
#501.	17.11	Adapt your algorithm for one execution of the algorithm for repeated executions. What is the slow part? Can you optimize it?
#502.	16.8	Try thinking about the number in terms of chunks of three digits.
#503.	17.19	Start with the first part: Finding the missing number if only one number is missing.

#504.	17.16	Recursive solution: You have two choices at each appointment (take the appointment or reject the appointment). As a brute force approach, you can recurse through all possibilities. Note, though, that if you take request `i`, your recursive algorithm should skip request `i + 1`.
#505.	16.23	Be very careful that your solution actually returns each value from 0 through 6 with equal probability.
#506.	17.22	Start with a brute force, recursive solution. Just create all words that are one edit away, check if they are in the dictionary, and then attempt that path.
#507.	16.10	Solution 2: What if you sorted the years? What would you sort by?
#508.	17.9	What does a brute force solution to get the kth smallest value for $3^a * 5^b * 7^c$ look like?
#509.	17.12	Try a recursive approach.
#510.	17.26	Solution 1: You should be able to get an `O(A+B)` algorithm to compute the similarity of two documents.
#511.	17.24	The brute force solution requires us to continuously compute the sums of each matrix. Can we optimize this?
#512.	17.7	One thing to try is maintaining a mapping of each name to its "true" spelling. You would also need to map from a true spelling to all the synonyms. Sometimes, you might need to merge two different groups of names. Play around with this algorithm to see if you can get it to work. Then see if you can simplify/optimize it.
#513.	16.7	If `k` were 1 when `a > b` and 0 otherwise, then you could return `a*k + b*(not k)`. But how do you create `k`?
#514.	16.10	Solution 2: Do you actually need to match the birth years and death years? Does it matter when a specific person died, or do you just need a list of the years of deaths?
#515.	17.5	Start with a brute force solution.
#516.	17.16	Recursive solution: You can optimize this approach through memoization. What is the runtime of this approach?
#517.	16.3	How can we find the intersection between two lines? If two line segments intercept, then this must be at the same point as their "infinite" extensions. Is this intersection point within both lines?
#518.	17.26	Solution 1: What is the relationship between the intersection and the union? Can you compute one from the other?
#519.	17.20	Recall that the median means the number for which half the numbers are larger and half the numbers are smaller.
#520.	16.14	You can't truly try all possible lines in the world—that's infinite. But you know that a "best" line must intersect at least two points. Can you connect each pair of points? Can you check if each line is indeed the best line?
#521.	16.26	Can we just process the expression from left to right? Why might this fail?
#522.	17.10	Start with a brute force solution. Can you just check each value to see if it's the majority element?

#523.	16.10	Solution 2: Observe that people are "fungible." It doesn't matter who was born and when they died. All you need is a list of birth years and death years. This might make the question of how you sort the list of people easier.
#524.	16.25	First scope the problem. What are the features you would want?
#525.	17.24	Can you do any sort of precomputation to make computing the sum of a submatrix $O(1)$?
#526.	17.16	Recursive solution: The runtime of your memoization approach should be $O(N)$, with $O(N)$ space.
#527.	16.3	Think carefully about how to handle the case of line segments that have the same slope and y-intercept.
#528.	16.13	To cut two squares in half, a line must go through the middle of both squares.
#529.	16.14	You should be able to get to an $O(N^2)$ solution.
#530.	17.14	Consider thinking about reorganizing the data in some way or using additional data structures.
#531.	16.17	Picture the array as alternating sequences of positive and negative numbers. Observe that we would never include just part of a positive sequence or part of a negative sequence.
#532.	16.10	Solution 2: Try creating a sorted list of births and a sorted list of deaths. Can you iterate through both, tracking the number of people alive at any one time?
#533.	16.22	Option #2: Think about how an `ArrayList` works. Can you use an `ArrayList` for this?
#534.	17.26	Solution 1: To understand the relationship between the union and the intersection of two sets, consider a Venn diagram (a diagram where one circle overlaps another circle).
#535.	17.22	Once you have a brute force solution, try to find a faster way of getting all valid words that are one edit away. You don't want to create all strings that are one edit away when the vast majority of them are not valid dictionary words.
#536.	16.2	Can you use a hash table to optimize the repeated case?
#537.	17.7	An easier way of taking the above approach is to have each name map to a list of alternate spellings. What should happen when a name in one group is set equal to a name in another group?
#538.	17.11	You could build a lookup table that maps from a word to a list of the locations where each word appears. How then could you find the closest two locations?
#539.	17.24	What if you precomputed the sum of the submatrix starting at the top left corner and continuing to each cell? How long would it take you to compute this? If you did this, could you then get the sum of an arbitrary submatrix in $O(1)$ time?
#540.	16.22	Option #2: It's not impossible to use an `ArrayList`, but it would be tedious. Perhaps it would be easier to build your own, but specialized for matrices.
#541.	16.10	Solution 3: Each birth adds one person and each death removes a person. Try writing an example of a list of people (with birth and death years) and then re-formatting this into a list of each year and a +1 for a birth and a -1 for a death.

#542. 17.16 Iterative solution: Take the recursive solution and investigate it more. Can you implement a similar strategy iteratively?

#543. 17.15 Extend the earlier idea to multiple words. Can we just break each word up in all possible ways?

#544. 17.1 You can think about binary addition as iterating through the number, bit by bit, adding two bits, and then carrying over the one if necessary. You could also think about it as grouping the operations. What if you first added each of the bits (without carrying any overflow)? After that, you can handle the overflow.

#545. 16.21 Do some math here or play around with some examples. What does this pair need to look like? What can you say about their values?

#546. 17.20 Note that you have to store all the elements you've seen. Even the smallest of the first 100 elements could become the median. You can't just toss very low or very high elements.

#547. 17.26 Solution 2: It's tempting to try to think of minor optimizations—for example, keeping track of the min and max elements in each array. You could then figure out quickly, in specific cases, if two arrays don't overlap. The problem with that (and other optimizations along these lines) is that you still need to compare all documents to all other documents. It doesn't leverage the fact that the similarity is sparse. Given that we have a lot of documents, we really need to not compare all documents to all other documents (even if that comparison is very fast). All such solutions will be $O(D^2)$, where D is the number of documents. We shouldn't compare all documents to all other documents.

#548. 16.24 Start with a brute force solution. What is the runtime? What is the best conceivable runtime for this problem?

#549. 16.10 Solution 3: What if you created an array of years and how the population changed in each year? Could you then find the year with the highest population?

#550. 17.9 In looking for the kth smallest value of $3^a * 5^b * 7^c$, we know that a, b, and c will be less than or equal to k. Can you generate all such numbers?

#551. 16.17 Observe that if you have a sequence of values which have a negative sum, those will never start or end a sequence. (They could be present in a sequence if they connected two other sequences.)

#552. 17.14 Can you sort the numbers?

#553. 16.16 We can think about the array as divided into three subarrays: `LEFT`, `MIDDLE`, `RIGHT`. `LEFT` and `RIGHT` are both sorted. The `MIDDLE` elements are in an arbitrary order. We need to expand `MIDDLE` until we could sort those elements and then have the entire array sorted.

#554. 17.16 Iterative solution: It's probably easiest to start with the end of the array and work backwards.

#555. 17.26 Solution 2: If we can't compare all documents to all other documents, then we need to dive down and start looking at things at the element level. Consider a naive solution and see if you can extend that to multiple documents.

#556. 17.22 To quickly get the valid words that are one edit away, try to group the words in the dictionary in a useful way. Observe that all words in the form `b_ll` (such as `bill`, `ball`, `bell`, and `bull`) will be one edit away. However, those aren't the only words that are one edit away from `bill`.

#557. 16.21 When you move a value a from array A to array B, then A's sum decreases by a and B's sum increases by a. What happens when you swap two values? What would be needed to swap two values and get the same sum?

#558. 17.11 If you had a list of the occurrences of each word, then you are really looking for a pair of values within two arrays (one value for each array) with the smallest difference. This could be a fairly similar algorithm to your initial algorithm.

#559. 16.22 Option #2: One approach is to just double the size of the array when the ant wanders to an edge. How will you handle the ant wandering into negative coordinates, though? Arrays can't have negative indices.

#560. 16.13 Given a line (slope and y-intercept), can you find where it intersects another line?

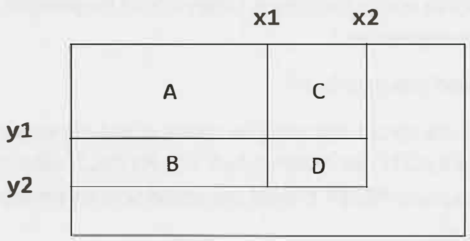
#561. 17.26 Solution 2: One way to think about this is that we need to be able to very quickly pull a list of all documents with some similarity to a specific document. (Again, we should not do this by saying "look at all documents and quickly eliminate the dissimilar documents." That will be at least $O(D^2)$.)

#562. 17.16 Iterative solution: Observe that you would never skip three appointments in a row. Why would you? You would always be able to take the middle booking.

#563. 16.14 Have you tried using a hash table?

#564. 16.21 If you swap two values, a and b, then the sum of A becomes `sumA - a + b` and the sum of B becomes `sumB - b + a`. These sums need to be equal.

#565. 17.24 If you can precompute the sum from the top left corner to each cell, you can use this to compute the sum of an arbitrary submatrix in `O(1)` time. Picture a particular submatrix. The full, precomputed sum will include this submatrix, an array immediately above it (C), and array to the left (B), and an area to the top and left (A). How can you compute the sum of just D?



#566. 17.10 Consider the brute force solution. We pick an element and then validate if it's the majority element by counting the number of matching and non-matching elements. Suppose, for the first element, the first few checks reveal seven non-matching elements and three matching elements. Is it necessary to keep checking this element?

#567. 16.17 Start from the beginning of the array. As that subsequence gets larger, it stays as the best subsequence. Once it becomes negative, though, it's useless.

#568.	17.16	Iterative solution: If you take appointment `i`, you will never take appointment `i + 1`, but you will always take appointment `i + 2` or `i + 3`.
#569.	17.26	Solution 2: Building off the earlier hint, we can ask what defines the list of documents with some similarity to a document like {13, 16, 21, 3}. What attributes does that list have? How would we gather all documents like that?
#570.	16.22	Option #2: Observe that nothing in the problem stipulates that the label for the coordinates must remain the same. Can you move the ant and all cells into positive coordinates? In other words, what would happen if, whenever you needed to grow the array in a negative direction, you relabeled all the indices such that they were still positive?
#571.	16.21	You are looking for values a and b where `sumA - a + b = sumB - b + a`. Do the math to work out what this means for a and b's values.
#572.	16.9	Approach these one by one, starting with subtraction. Once you've completed one function, you can use it to implement the others.
#573.	17.6	Start with a brute force solution.
#574.	16.23	Start with a brute force solution. How many times does it call `rand5()` in the worst case?
#575.	17.20	Another way to think about this is: Can you maintain the bottom half of elements and the top half of elements?
#576.	16.10	Solution 3: Be careful with the little details in this problem. Does your algorithm/code handle a person who dies in the same year that they are born? This person should be counted as one person in the population count.
#577.	17.26	Solution 2: The list of documents similar to {13, 16, 21, 3} includes all documents with a 13, 16, 21, and 3. How can we efficiently find this list? Remember that we'll be doing this for many documents, so some precomputing can make sense.
#578.	17.16	Iterative solution: Use an example and work backwards. You can easily find the optimal solution for the subarrays $\{r_n\}$, $\{r_{n-1}, r_n\}$, $\{r_{n-2}, \ldots, r_n\}$. How would you use those to quickly find the optimal solution for $\{r_{n-3}, \ldots, r_n\}$?
#579.	17.2	Suppose you had a method `shuffle` that worked on decks up to `n - 1` elements. Could you use this method to implement a new `shuffle` method that works on decks up to n elements?
#580.	17.22	Create a mapping from a wildcard form (like `b_ll`) to all words in that form. Then, when you want to find all words that are one edit away from `bill`, you can look up `_ill`, `b_ll`, `bi_l`, and `bil_` in the mapping.
#581.	17.24	The sum of just D will be `sum(A&B&C&D) - sum(A&B) - sum(A&C) + sum(A)`.
#582.	17.17	Can you use a trie?
#583.	16.21	If we do the math, we are looking for a pair of values such that `a - b = (sumA - sumB) / 2`. The problem then reduces to looking for a pair of values with a particular difference.
#584.	17.26	Solution 2: Try building a hash table from each word to the documents that contain this word. This will allow us to easily find all documents with some similarity to {13, 16, 21, 3}.
#585.	16.5	How does a zero get into the result of `n!`? What does it mean?

#586. 17.7 If each name maps to a list of its alternate spellings, you might have to update a lot of lists when you set X and Y as synonyms. If X is a synonym of {A, B, C}, and Y is a synonym of {D, E, F} then you would need to add {Y, D, E, F} to A's synonym list, B's synonym list, C's synonym list, and X's synonym list. Ditto for {Y, D, E, F}. Can we make this faster?

#587. 17.16 Iterative solution: If you take an appointment, you can't take the next appointment, but you can take anything after that. Therefore, `optimal(`r_i`, ..., `r_n`) = max(`r_i` + optimal(`r_{i+2}`, ..., `r_n`), optimal(`r_{i+1}`, ..., `r_n`))`. You can solve this iteratively by working backwards.

#588. 16.8 Have you considered negative numbers? Does your solution work for values like 100,030,000?

#589. 17.15 When you get recursive algorithms that are very inefficient, try looking for repeated subproblems.

#590. 17.19 Part 1: If you have to find the missing number in O(1) space and O(N) time, then you can do a only constant number of passes through the array and can store only a few variables.

#591. 17.9 Look at the list of all values for $3^a * 5^b * 7^c$. Observe that each value in the list will be 3*(some previous value), 5*(some previous value), or 7*(some previous value).

#592. 16.21 A brute force solution is to just look through all pairs of values to find one with the right difference. This will probably look like an outer loop through A with an inner loop through B. For each value, compute the difference and compare it to what we're looking for. Can we be more specific here, though? Given a value in A and a target difference, do we know the exact value of the element within B we're looking for?

#593. 17.14 What about using a heap or tree of some sort?

#594. 16.17 If we tracked the running sum, we should reset it as soon as the subsequence becomes negative. We would never add a negative sequence to the beginning or end of another subsequence.

#595. 17.24 With precomputation, you should be able to get a runtime of $O(N^4)$. Can you make this even faster?

#596. 17.3 Try this recursively. Suppose you had an algorithm to get a subset of size m from n - 1 elements. Could you develop an algorithm to get a subset of size m from n elements?

#597. 16.24 Can we make this faster with a hash table?

#598. 17.22 Your previous algorithm probably resembles a depth-first search. Can you make this faster?

#599. 16.22 Option #3: Another thing to think about is whether you even need a grid to implement this. What information do you actually need in the problem?

#600. 16.9 Subtraction: Would a negate function (which converts a positive integer to negative) help? Can you implement this using the add operator?

#601. 17.1 Focus on just one of the steps above. If you "forgot" to carry the ones, what would the add operation look like?

#602. 16.21 What the brute force really does is look for a value within B which equals `a - target`. How can you more quickly find this element? What approaches help us quickly find out if an element exists within an array?

#603. 17.26 Solution 2: Once you have a way of easily finding the documents similar to a particular document, you can go through and just compute the similarity to those documents using a simple algorithm. Can you make this faster? Specifically, can you compute the similarity directly from the hash table?

#604. 17.10 The majority element will not necessarily look like the majority element at first. It is possible, for example, to have the majority element appear in the first element of the array and then not appear again for the next eight elements. However, in those cases, the majority element will appear later in the array (in fact, many times later on in the array). It's not necessarily critical to continue checking a specific instance of an element for majority status once it's already looking "unlikely."

#605. 17.7 Instead, X, A, B, and C should map to the same instance of the set `{X, A, B, C}`. Y, D, E, and F should map to the same instance of `{Y, D, E, F}`. When we set X and Y as synonyms, we can then just copy one of the sets into the other (e.g., add `{Y, D, E, F}` to `{X, A, B, C}`). How else do we change the hash table?

#606. 16.21 We can use a hash table here. We can also try sorting. Both help us locate elements more quickly.

#607. 17.16 Iterative solution: If you're careful about what data you really need, you should be able to solve this in `O(n)` time and `O(1)` additional space.

#608. 17.12 Think about it this way: If you had methods called `convertLeft` and `convertRight` (which would convert left and right subtrees to doubly linked lists), could you put those together to convert the whole tree to a doubly linked list?

#609. 17.19 Part 1: What if you added up all the values in the array? Could you then figure out the missing number?

#610. 17.4 How long would it take you to figure out the least significant bit of the missing number?

#611. 17.26 Solution 2: Imagine you are looking up the documents similar to {1, 4, 6} by using a hash table that maps from a word to documents. The same document ID appears multiple times when doing this lookup. What does that indicate?

#612. 17.6 Rather than counting the number of twos in each number, think about digit by digit. That is, count the number of twos in the first digit (for each number), then the number of twos in the second digit (for each number), then the number of twos in the third digit (for each number), and so on.

#613. 16.9 Multiply: it's easy enough to implement `multiply` using add. But how do you handle negative numbers?

#614. 16.17 You can solve this in `O(N)` time and `O(1)` space.

#615. 17.24 Suppose this was just a single array. How could we compute the subarray with the largest sum? See 16.17 for a solution to this.

#616. 16.22 Option #3: All you actually need is some way of looking up if a cell is white or black (and of course the position of the ant). Can you just keep a list of all the white cells?

#617.	17.17	One solution is to insert every suffix of the larger string into the trie. For example, if the word is `dogs`, the suffixes would be `dogs`, `ogs`, `gs`, and `s`. How would this help you solve the problem? What is the runtime here?
#618.	17.22	A breadth-first search will often be faster than a depth-first search—not necessarily in the worst case, but in many cases. Why? Can you do something even faster than this?
#619.	17.5	What if you just started from the beginning, counting the number of As and the number of Bs you've seen so far? (Try making a table of the array and the number of As and Bs thus far.)
#620.	17.10	Note also that the majority element must be the majority element for some subarray and that no subarray can have multiple majority elements.
#621.	17.24	Suppose I just wanted you to find the maximum submatrix starting at row `r1` and ending at row `r2`, how could you most efficiently do this? (See the prior hint.) If I now wanted you find the maximum subarray from `r1` to `(r2+2)`, could you do this efficiently?
#622.	17.9	Since each number is 3, 5, or 7 times a previous value in the list, we could just check all possible values and pick the next one that hasn't been seen yet. This will result in a lot of duplicated work. How can we avoid this?
#623.	17.13	Can you just try all possibilities? What might that look like?
#624.	16.26	Multiplication and division are higher priority operations. In an expression like `3*4 + 5*9/2 + 3`, the multiplication and division parts need to be grouped together.
#625.	17.14	If you picked an arbitrary element, how long would it take you to figure out the rank of this element (the number of elements bigger or smaller than it)?
#626.	17.19	Part 2: We're now looking for two missing numbers, which we will call a and b. The approach from part 1 will tell us the sum of a and b, but it won't actually tell us a and b. What other calculations could we do?
#627.	16.22	Option #3: You could consider keeping a hash set of all the white cells. How will you be able to print the whole grid, though?
#628.	17.1	The adding step alone would convert 1 + 1 -> 0, 1 + 0 -> 1, 0 + 1 -> 1, 0 + 0 -> 0. How do you do this without the + sign?
#629.	17.21	What role does the tallest bar in the histogram play?
#630.	16.25	What data structure would be most useful for the lookups? What data structure would be most useful to know and maintain the order of items?
#631.	16.18	Start with a brute force approach. Can you try all possibilities for a and b?
#632.	16.6	What if you sorted the arrays?
#633.	17.11	Can you just iterate through both arrays with two pointers? You should be able to do it in `O(A+B)` time, where A and B are the sizes of the two arrays.
#634.	17.2	You could build this algorithm recursively by swapping the nth element for any of the elements before it. What would this look like iteratively?
#635.	16.21	What if the sum of A is 11 and the sum of B is 8? Can there be a pair with the right difference? Check that your solution handles this situation appropriately.

#636. 17.26 Solution 3: There's an alternative solution. Consider taking all of the words from all of the documents, throwing them into one giant list, and sorting this list. Assume you could still know which document each word came from. How could you track the similar pairs?

#637. 16.23 Make a table indicating how each possible sequence of calls to `rand5()` would map to the result of `rand7()`. For example, if you were implementing `rand3()` with `(rand2() + rand2()) % 3`, then the table would look like the below. Analyze this table. What can it tell you?

```
1st    2nd    Result
0      0      0
0      1      1
1      0      1
1      1      2
```

#638. 17.8 This problem asks us to find the longest sequence of pairs you can build such that both sides of the pair are constantly increasing. What if you needed only one side of the pair to increase?

#639. 16.15 Try first creating an array with the frequency that each item occurs.

#640. 17.21 Picture the tallest bar, and then the next tallest bar on the left and the next tallest bar on the right. The water will fill the area between those. Can you calculate that area? What do you do about the rest?

#641. 17.6 Is there a faster way of calculating how many twos are in a particular digit across a range of numbers? Observe that roughly $\frac{1}{10}$ th of any digit should be a 2—but only roughly. How do you make that more exact?

#642. 17.1 You can do the add step with an XOR.

#643. 16.18 Observe that one of the substrings, either a or b, must start at the beginning of the string. That cuts down the number of possibilities.

#644. 16.24 What if the array were sorted?

#645. 17.18 Start with a brute force solution.

#646. 17.12 Once you have a basic idea for a recursive algorithm, you might get stuck on this: sometimes your recursive algorithm needs to return the start of the linked list, and sometimes it needs to return the end. There are multiple ways of solving this issue. Brainstorm some of them.

#647. 17.14 If you picked an arbitrary element, you would, on average, wind up with an element around the 50th percentile mark (half the elements above it and half the elements below). What if you did this repeatedly?

#648. 16.9 Divide: If you're trying to compute, where $x = a/b$, remember that $a = bx$. Can you find the closest value for x? Remember that this is integer division and x should be an integer.

#649. 17.19 Part 2: There are a lot of different calculations we could try. For example, we could multiply all the numbers, but that will only lead us to the product of a and b.

#650. 17.10 Try this: Given an element, start checking if this is the start of a subarray for which it's the majority element. Once it's become "unlikely" (appears less than half the time), start checking at the next element (the element after the subarray).