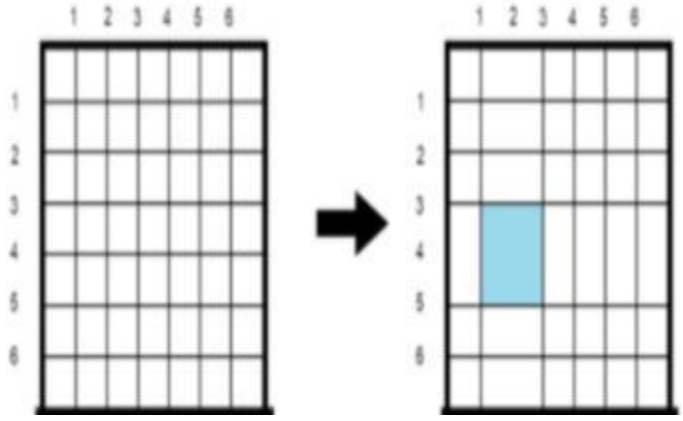
Contents

Storage Optimization

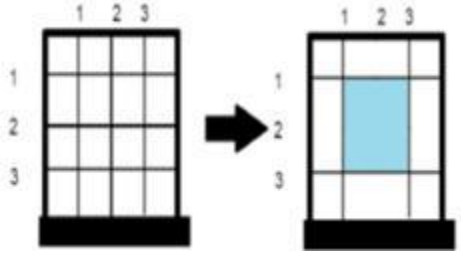
Amazon is experimenting with a flexible storage system for their warehouses. The storage unit consists of a shelving system which is one meter deep with removable vertical and horizontal separators. When all separators are installed, each storage space is one cubic meter (1' x 1' x 1'). Determine the volume of the largest space when a series of horizontal and vertical separators are removed.

Example
n = 6
m = 6
h = [4]
v = [2]
Consider the diagram below. The left image depicts the initial storage unit with n = 6 horizontal and m = 6 vertical separators, where the volume of the largest storage space is 1 x 1 x 1. The right image depicts that unit after the fourth horizontal and second vertical separators are removed. The maximum storage volume for that unit is then 2 x 2 x 1 = 4 cubic meters:



Sample Case 0
Sample Input 0
STDIN Function
3 -> n = 3
3 -> m = 3
1 -> h[] size x = 1
2 -> h = [2]
1 -> v[] size y = 1

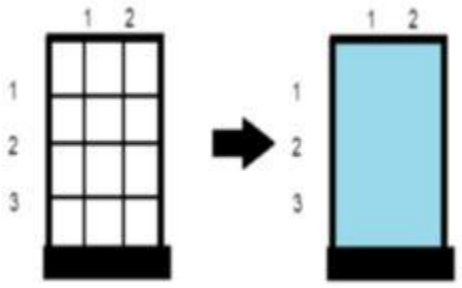
2 -> v = [2]
Sample Output 0
4
Explanation 0
There are n = m = 3 separators in the vertical and horizontal directions. Separators to remove are h = [2] and v = [2] so the unit looks like this:



Return the volume of the biggest space, 4, as the answer.

Sample Case 1
Sample Input 1
STDIN Function
2 -> n = 2
2 -> m = 2
1 -> h[] size x = 1
1 -> h = [1]
1 -> v[] size y = 1
2 -> v = [2]
Sample Output 1
4
Explanation 1
There are 2 vertical and two horizontal separators initially. After removing the two separators, h = [1] and v = [2], the top-right cell will be the largest storage space at 4 cubic meters.

Sample Case 2
Sample Input 2
STDIN Function
3 -> n = 3
2 -> m = 2
3 -> h[] size x = 3
1 -> h = [1, 2, 3]
2
3
2 -> v[] size y = 3
1 -> v = [1, 2]
2
Sample Output 2
12
Explanation 2
Initially there are n = 3 horizontal and m = 2 vertical separators. Remove separators h = [1, 2, 3] and v = [1, 2] so the unit looks like this:



The volume of the biggest storage space is 12

```
def storage_optimization(n: int, m: int, h: List[int], v: List[int]) -> int:
    max_h = 0
    max_w = 0

    h_ptr = 0
    v_ptr = 0

    prev = 0
    for hc in range(1, n+2):
        if hc != h[h_ptr]:
            max_h = max(max_h, hc - prev)
            prev = hc
        else:
            if h_ptr < len(h) - 1:
                h_ptr += 1

    prev = 0
    for vc in range(1,m+2):
        if vc != v[v_ptr]:
            max_w = max(max_w, vc - prev)
            prev = vc
        else:
            if v_ptr < len(v) - 1:
                v_ptr += 1

    return max_h * max_w
```

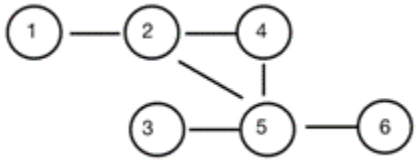
Shopping Patterns

https://aonecode.com/amazon-online-assessment-shopping-patterns
LC 1761

Amazon is trying to understand customer shopping patterns and offer items that are regularly bought together to new customers. Each item that has been bought together can be represented as an undirected graph where edges join often bundled products. A group of *n* products is uniquely numbered from 1 of *product_nodes*. A *trio* is defined as a group of three related products that all connected by an edge. Trios are scored by counting the number of related products outside of the trio, this is referred as a *product sum*.
Given product relation data, determine the minimum product sum for all trios of related products in the group. If no such trio exists, return -1.
Example
products_nodes = 6
products_edges = 6

products_from = [1,2,2,3,4,5]
products_to = [2,4,5,5,5,6]

Product	Related Products
1	2
2	1, 4, 5
3	5
4	2, 5
5	2, 3, 4, 6
6	5



A graph of n = 6 products where the only trio of *related products is (2, 4, 5).*
The product scores based on the graph above are:

Product	Outside Products	Which Products Are Outside
2	1	1
4	0	
5	2	3, 6

In the diagram above, the total product score is 1 + 0 + 2 = 3 for the trio (2, 4, 5).
Function Description
Complete the function getMinScore in the editor below.
getMinScore has the following parameter(s):
int products_nodes: the total number of products
int products_edges the total number of edges representing related products
int products_from[products_nodes]: each element is a node of one side of an edge.
int products_to[products edges]: each products_to[i] is a node connected to products_from[i]
Returns:
int: the minimum product sum for all trios of related products in the group. If no such trio exists, return -1.
Constraints

- *1 <= products_nodes <= 500*
- *1 <= products_edges <= min(500, (products_nodes * (products_nodes - 1)) / 2)*
- *1 <= products_from[i], products to[i] <= products_nodes*
- *products_from[i] != products_to[i]*

Sample Case 0
STDIN Funtion
------- ----------
5 6 -> products_nodes = 5 products_edges = 6
1 2 -> products_from[0] = 1 products_to[0] = 2
1 3 -> products_from[1] = 1 products_to[1] = 3
2 3 -> products_from[2] = 2 products_to[2] = 3
2 4 -> products_from[3] = 2 products_to[3] = 4
3 4 -> products_from[4] = 3 products_to[4] = 4
4 5 -> products_from[5] = 4 products_to[5] = 5
Sample Output
2
Explanation
There are two possible trios: {1,2,3} and {2,3,4}
The score for {1,2,3} is 0 + 1 + 1 = 2.
The score for {2,3,4} is 1 + 1 + 1 = 3.
Return 2.

```
class Shoppingpattens(object):
    def solution(self, n, edges):
        """
        :type n: int
        :type edges: List[List[int]]
        :rtype: int
        """
        graph = defaultdict(set)
        for a, b in edges:
            graph[a].add(b)
            graph[b].add(a)

        d = {n:len(graph[n]) for n in graph}

        res = float('inf')
        for a in graph:
            for b in graph[a]:
                for c in graph[a] & graph[b]:
                    res = min(res, d[a]+d[b]+d[c]-6)
                    graph[c].discard(a)
                graph[b].discard(b)
        if res == float('inf'):
            return -1
        return res
```

Shopping Options

A customer wants to buy a pair of jeans, a pair of shoes, a skirt, and a top but has a limited budget in dollars. Given different pricing options for each product, determine how many options our customer has to buy 1 of each product. You cannot spend more money than the budgeted amount.

Example
priceOfJeans = [2, 3]
priceOfShoes = [4]
priceOfSkirts = [2, 3]
priceOfTops = [1, 2]
budgeted = 10

The customer must buy shoes for 4 dollars since there is only one option. This leaves 6 dollars to spend on the other 3 items. Combinations of prices paid for jeans, skirts, and tops respectively that add up to 6 dollars or less are [2, 2, 2], [2, 2, 1], [3, 2, 1], [2, 3, 1]. There are 4 ways the customer can purchase all 4 items.

Function Description

Complete the getNumberOfOptions function in the editor below. The function must return an integer which represents the number of options present to buy the four items.

getNumberOfOptions has 5 parameters:
int[] priceOfJeans: An integer array, which contains the prices of the pairs of jeans available.
int[] priceOfShoes: An integer array, which contains the prices of the pairs of shoes available.
int[] priceOfSkirts: An integer array, which contains the prices of the skirts available.
int[] priceOfTops: An integer array, which contains the prices of the tops available.
int dollars: the total number of dollars available to shop with.

Constraints

1 ≤ a, b, c, d ≤ 103
1 ≤ dollars ≤ 109
1 ≤ price of each item ≤ 109
Note: a, b, c and d are the sizes of the four price arrays

```
import bisect

class ShoppingOptions(object):

    def solution(self, priceOfJeans, priceOfShoes, priceOfSkirts, priceOfTops,
dollars):

        pair1=[]

        pair2=[]

        for a in priceOfJeans:

            for b in priceOfShoes:

                pair = a + b

                if pair < dollars:

                    pair1.append(pair)


        for a in priceOfSkirts:

            for b in priceOfTops:

                pair = a + b

                if pair < dollars:

                    pair2.append(pair)


        if len(pair1) > len(pair2):
```

```
            pair1, pair2 = pair2, pair1


        pair2.sort()
        res = 0
        for p1 in pair1:
            p2 = dollars - p1
            idx = bisect.bisect_right(pair2,p2)
            if idx!=-1:
                res += idx
        return res
    def test(self):
        assert self.solution([2, 3], [4], [2, 3], [1, 2], 10) == 4
        assert self.solution([2, 3], [4], [2, 3], [1, 2], 9) == 1
        assert self.solution([6], [1, 1, 1, 1], [4, 5, 6], [1], 12) == 4
        assert self.solution([6], [1, 1, 1, 1], [4, 5, 6], [1], 13) == 8
        assert self.solution([6], [1, 1, 1, 1], [4, 5, 6], [1], 14) == 12
        assert self.solution([100], [1, 1, 1, 1], [4, 5, 6], [1], 99) == 0
        assert self.solution([1], [1], [1], [1], 4) == 1
        assert self.solution([1], [1], [1], [1], 3) == 0
```

Optimizating Box Weight

An Amazon Fulfillment Associate has a set of items that need to be packed into two boxes. Given an integer array of the item weights (arr) to be packed, divide the item weights into two subsets, A and B, for packing into the associated boxes, while respecting the following conditions:
The intersection of A and B is null.
The union A and B is equal to the original array.
The number of elements in subset Ais minimal.
The sum of A's weights is greater than the sum of B's weights.
Return the subset A in increasing order where the sum of A's weights is greater than the sum of B's weights. If more than one subset A exists, return the one with the maximal total weight.

Input Format For Custom Testing
STDIN Function

6 -> arr[] size n = 6
5 -> arr[] = [5, 3, 2, 4, 1, 2]
3
2
4
1
2

Sample Output
4
5

Explanation
n = 6
arr = [5, 3, 2, 4, 1, 2]
The subset of A that satisfies the conditions is [4, 5]
A is minimal (size 2)
Sum(A) = (4 + 5) = 9 > Sum(B) = (1 + 2 + 2 + 3) = 8
The intersection of A and B is null and their union is equal to arr.
The subset A with the maximal sum is [4, 5].

```
class OptimizatingBoxWeight(object):
    def solution(self, arr):
        arr.sort(reverse=True)
        total =sum(arr)
        res =[]
        sumA = 0
        for i in range(0,len(arr)):
            res.append(arr[i])
            sumA +=arr[i]
            sumB = total - sumA
            if sumA > sumB:
                break
        return res[::-1]
    def test(self):
        res = self.solution([5, 3, 2, 4, 1, 2])
        print(res)
```

Optimal Utilization – Prime Air time

https://leetcode.com/discuss/interview-question/373202

Given 2 lists `a` and `b`. Each element is a pair of integers where the first integer represents the unique id and the second integer represents a value. Your task is to find an element from `a` and an element form `b` such that the sum of their values is less or equal to `target` and as close

to `target` as possible. Return a list of ids of selected elements. If no pair is possible, return an empty list.

Example 1:

```
Input:
a = [[1, 2], [2, 4], [3, 6]]
b = [[1, 2]]
target = 7

Output: [[2, 1]]

Explanation:
There are only three combinations [1, 1], [2, 1], and [3, 1], which have a
total sum of 4, 6 and 8, respectively.
Since 6 is the largest sum that does not exceed 7, [2, 1] is the optimal pair.
```

Example 2:

```
Input:
a = [[1, 3], [2, 5], [3, 7], [4, 10]]
b = [[1, 2], [2, 3], [3, 4], [4, 5]]
target = 10

Output: [[2, 4], [3, 2]]

Explanation:
There are two pairs possible. Element with id = 2 from the list `a` has
a value 5, and element with id = 4 from the list `b` also has a value 5.
Combined, they add up to 10. Similarily, element with id = 3 from `a` has
a value 7, and element with id = 2 from `b` has a value 3.
These also add up to 10. Therefore, the optimal pairs are [2, 4] and [3, 2].
```

Example 3:

```
Input:
a = [[1, 8], [2, 7], [3, 14]]
b = [[1, 5], [2, 10], [3, 14]]
target = 20

Output: [[3, 1]]
```

Example 4:

```
Input:
a = [[1, 8], [2, 15], [3, 9]]
b = [[1, 8], [2, 11], [3, 12]]
target = 20

Output: [[1, 3], [3, 2]]
```

O(MlogM + NlogN) and two-pointer traversal is O(M + N), the final complexity can be regarded as O(KlogK) where K is the longest input array.

```
class MaxShipping(object):
    def maxShippingDist(self,A, B, maxDist):
        if not A or not A[0] :return []
        if not B or not B[0]: return []
        res = []
        target = 0
```

```
        A.sort(key=lambda x:x[1])
        B.sort(key=lambda x:x[1],reverse=True)
        m,n=len(A),len(B)
        i,j=0,0
        target=0
        while i<m and j <n:
            theSum=A[i][1]+B[j][1]
            if theSum>maxDist:
                j+=1
            elif theSum<maxDist:
                if theSum>target:
                    target=theSum
                i+=1
            else:
                target=maxDist
                break;

        i,j=0,0
        res=[]
        while i<m and j <n:
            theSum=A[i][1]+B[j][1]
            if theSum==target:
                res.append([A[i][0],B[j][0]])
                i,j=i+1,j+1
            elif theSum<target:
                i+=1
            else:
                j+=1
        return res

    def test(self):
        list1 = [[1, 8], [2, 15], [3, 9]]
        list2 = [[1, 8], [2, 11], [3, 12]]
        maxDist = 20
        res=self.maxShippingDist(list1, list2, maxDist)
        pause=1
```

Cloud Front Caching

AWS CloudFront wants to build an algo to measure the efficiency of its caching network. The network is represented as a number of nodes and a list of connected pairs. The efficiency of this network can be estimated by first summing the cost of each isolated set of nodes where each individual node has a cost of 1. To account for the increase in efficiency as more nodes are connected, update the cost of each isolated set to be the ceiling of the square root of the original cost and return the final sum of all costs.

Example:
n = 10 nodes
edges = [[1 2] , [1 3] , [2 4] , [3 5] , [7 8]]

There are 2 isloated sets with more than one node {1,2,3,4,5} and {7,8}. The ceilings of their square roots are:
5^1/2 = 2.236 and ceil(2.236) = 3
2^1/2 = 1.414 and ceil(1.414) = 2

The other three isolated nodes are separate and the square root of their weights is 1^1/2 = 1 respectively.
The sum is 3+2+(3*1) = 8

Function Description
Complete the function *connectedSum* in the editor below

connectedSum has the following parameter(s):
int n: the number of nodes
str edges[m]: an array of strings that consist of a space-separated integer pain that denotes two connected nodes, p and q

Returns:
int: an integer that denotes the sum of the values calculated

Constraints:

- 2 <= n <= 10^5
- 1 <= m <=10^5
- 1 <= p,q <= n
- p != n

Sample Input 0
n = 4 nodes
edges[] size m = 2
edges[] = [[1 2], [1 4]]

Sample Output 0
3

Explanation 0
The values to sum are:

1. Set {1,2,4}: c =ceil(sqrt(3)) = 2

2. Set {3}: c = ceil(sqrt(1)) = 1

2+1=3

Sample Input 1
n = 8 nodes
edges[] size m = 4
edges[] = [[8 1], [5 8], [7 3], [8 6]]

Sample Output 1
6

Explanation 1
The values to sum for each group are:

1. Set {2}: c =ceil(sqrt(1)) = 1
2. Set {4}: c = ceil(sqrt(1)) = 1
3. Set {1,5,6,8}: c = ceil(sqrt(4)) = 2
4. Set {3,7}: c = ceil(sqrt(2)) = 2

1+1+2+2 = 6

```
import math
import collections

class CloudFront(object):
    def solution(self,n, arr):
        return self.doDFS(n,arr)

    def doBFS(self,n,arr):
        def bfs(node):
            dq=collections.deque([node])
            visited[node] = True
            cnt = 0
            while dq:
                node = dq.popleft()
                cnt+=1
                for nextNode in graph[node]:
                    if not visited[nextNode]:
                        dq.append(nextNode)
                        visited[nextNode] = True
            return cnt

        visited = [False] * (n + 1)
```

```
        graph = collections.defaultdict(set)
        for a, b in arr:
            graph[a].add(b)
            graph[b].add(a)
        cost = 0
        for i in range(1, n+1):
            if not visited[i]:
                cnt = bfs(i)
                cost += math.ceil(math.sqrt(cnt))
        return cost


    def doDFS(self, n, arr):
        def dfs(node):
            cnt = 1
            visited[node] = True
            for next in graph[node]:
                if not visited[next]:
                    cnt += dfs(next)
            return cnt

        visited = [False] * (n + 1)
        graph = collections.defaultdict(set)
        for a, b in arr:
            graph[a].add(b)
            graph[b].add(a)
        cost = 0
        for i in range(1, n+1):
            if not visited[i]:
                cnt = dfs(i)
                cost += math.ceil(math.sqrt(cnt))
        return cost

    def test(self):
        res= self.solution(10,[[1, 2] , [1, 3] , [2, 4] , [3, 5] , [7,
8]])
        print(res)
        res= self.solution(4,[[1, 2], [1, 4]])
        print(res)
        res= self.solution(8,[[8, 1], [5, 8], [7, 3], [8, 6]])
        print(res)
```

Turnstile

A university has exactly one turnstile. It can be used either as an exit or an entrance. Unfortunately, sometimes many people want to pass through the turnstile and their directions can be different. The ith person comes to the turnstile at time[i] and wants to

either exit the university if direction[i] = 1 or enter the university if direction[i] = 0. People form 2 queues, one to exit and one to enter. They are ordered by the time when they came to the turnstile and, if the times are equal, by their indices.

If some person wants to enter the university and another person wants to leave the university at the same moment, there are three cases:
If in the previous second the turnstile was not used (maybe it was used before, but not at the previous second), then the person who wants to leave goes first.
If in the previous second the turnstile was used as an exit, then the person who wants to leave goes first.
If in the previous second the turnstile was used as an entrance, then the person who wants to enter goes first.
Passing through the turnstile takes 1 second.

For each person, find the time when they will pass through the turnstile.

Input
arrTime, an array of n integers where the value at index i is the time in seconds when the ith person will come
direction, a list of integers where the value at indexi is the direction of the ith person.

Output
an array of integers where the value at index i is the time when the ith person will pass the turnstile.

Constraints
1 <= n <= 10^5
0 <= arrTime[i] <= 10^9 for 0<= i <=n-1
arrTime[i] <= arrTime[i+1] for 0 <= i <= n - 2
0 <= direction[i] <= 1 for 0 <= o <= n - 1

Example1
Input:
n = 4
arrTime = [1, 1, 2, 6]
direction = [0, 1, 1, 0]

Output:
[3,1,2,6]

Explanation:
At time 1, person 0 and 1 want to pass through the turnstile. Person 0 wants to enter the store and person 1 wants to leave the store. The turnstile was not used in the previous second, so the priority is on the side of the person 1
At time 2, person 0 and 2 want to pass through the turnstile. Person 2 wants to leave the store and at the previous second the turnstile was used as an exit, so the person 2 passes through the turnstile.
At time 3, person 0 passes through the turnstile.

At time 6, person 3 passes through the turnstile.

Example2

Input:
numPersons = 5
arrTime = [1,2,2,4,4]
direction = [0, 1, 0, 0, 1]

Output:
[1, 3, 2, 5, 4]

Explanation:
At time 1, person 0 passes through the turnstile (enters).
At time 2, persons 1 (exit) and 2 (enter) want to pass through the turnstile, and person 2 passes through the turnstile because their direction is equal to the direction at the previous second.
At time 3. person 1 passes through the turnstile (exit).
At time 4, persons 3 (enter) and 4 (exit) want to pass through the turnstile. Person 4 passes through the turnstile because at the previous second the turnstile was used to exit.
At time 5, person 3 passes through the turnstile.

```
class Turnstile(object):
    #Time Complexity: O(n), Space Complexity: O(n) where n = size(time)
    def solution(self, arrTime, dir):
        enter, exit = [], []
        res = [0] * len(arrTime)
        for i, t in enumerate(arrTime):
            if dir[i] == 1:
                exit.append([t, i])
            else:
                enter.append([t, i])

        time, lastTurn = 0, -1 # time is 0 at the beginning and -1
                                    # indicates nothing happened at pri
or time
        while exit or enter:
            # Process the exit queue if and only if following condition
s are satisfied
            # If exit queue is not empty and the person at the front of
 the queue can go out based on his time stamp
            # and ( Nothing happened at last time stamp i.e. nobody mov
ed in or out so lastTurn will be -1 in this case
            # or, somebody moved out at last time stamp, in this case l
astTurn will be 1
            # or, nobody is there in the entrance queue
            # or, at last time stamp somebody got in but the person at
the front of the queue can't go in due to their timestamp
            if exit and exit[0][0] <= time and \
            (lastTurn !=0 or not enter or (lastTurn == 0 and enter[0][0
] > time)):
```

```
                res[exit[0][1]] = time
                lastTurn = 1
                exit.pop(0)
            elif enter and enter[0][0] <= time:
                res[enter[0][1]] = time
                lastTurn = 0
                enter.pop(0)
            else:
                lastTurn = -1

            time += 1

        return res
    def test(self):
        res= self.solution([0,0,1,5], [0,1,1,0])
        print (res)
        assert res == [2,0,1,5]
        res= self.solution([1,2,4], [0,1,1])
        print (res)
        assert res == [1,2,4]
        res= self.solution([1,1], [1,1])
        print (res)
        assert res == [1,2]
        res= self.solution([1,1,3,3,4,5,6,7,7], [1,1,0,0,0,1,1,1,1])
        print (res)
        assert res ==  [1,2,3,4,5,6,7,8,9]
```

1328. Break a Palindrome

Given a palindromic string of lowercase English letters `palindrome`, replace **exactly one** character with any lowercase English letter so that the resulting string is **not** a palindrome and that it is the **lexicographically smallest** one possible.
Return *the resulting string. If there is no way to replace a character to make it not a palindrome, return an* ***empty string***.
A string `a` is lexicographically smaller than a string `b` (of the same length) if in the first position where `a` and `b` differ, `a` has a character strictly smaller than the corresponding character in `b`. For example, `"abcc"` is lexicographically smaller than `"abcd"` because the first position they differ is at the fourth character, and `'c'` is smaller than `'d'`.

Example 1:

```
Input: palindrome = "abccba"
Output: "aaccba"
Explanation: There are many ways to make "abccba" not a palindrome,
such as "zbccba", "aaccba", and "abacba".
Of all the ways, "aaccba" is the lexicographically smallest.
```

Example 2:

```
Input: palindrome = "a"
Output: ""
Explanation: There is no way to replace a single character to make "a"
not a palindrome, so return an empty string.
```

Example 3:

```
Input: palindrome = "aa"
Output: "ab"
```

Example 4:

```
Input: palindrome = "aba"
Output: "abb"
```

Constraints:

- `1 <= palindrome.length <= 1000`
- `palindrome` consists of only lowercase English letters.

$$Solution$$

```
    def breakPalindrome(self, s: str) -> str:
        """
Check half of the string,
replace a non 'a' character to 'a'.

If only one character, return empty string.
Otherwise repalce the last character to 'b'


Complexity
Time O(N)
Space O(N)
        """
        for i in range(len(s) // 2):
            if s[i] != 'a':
                return s[:i] + 'a' + s[i + 1:]
        return s[:-1] + 'b' if s[:-1] else ''
```

Fetch Items To Display - Ranking Products

A search engine website wants to implement a new feature that allows their users to sort their search results.

Each search result consists of a URL, a timestamp, and a relevance score. Given an array of results, the name of the column to sort by, the sort order (ascending or descending), the page number, and size of each page, implement a function that returns a list of results.

Input

`sortColumn`: a number representing the column to sort by: `0 = URL`, `1 = timestamp`, `2 = relevance`

`sortOrder`: a number representing the sort order: `0 = ascending`, `1 = descending`

`pageSize`: the number of results that is required to be displayed on a single page

`pageIndex`: the page number, starting from `0`

`results`: a map of `URL` strings to tuples representing the `(relevance, timestamp)`

Output

Return a list of URLs to be displayed.

Note

`pageSize` is never zero, and is always less than the number of results.

Examples

Example 1:

Input:

sortColumn = `1`

sortOrder = `0`

pageSize = `2`

pageIndex = `1`

results = `[["foo.com", 10, 15], ["bar.com", 3, 4]. ["baz.com", 17, 8]]`

Output: `["baz"]`

Explanation:

There are `3 results`.

Sort them by `timestamp (sortColumn = 1)` in ascending order `results = [["bar.com", 3, 4], ["foo.com", 10, 15], ["baz.com", 17, 8]]`.

Display up to `2 results` on each page.

The page `0` contains `2` results `["bar.com", "foo.com"]` and page `1` contains only `1` result `["baz.com"]`.

Therefore, the output is `["baz.com"]`.

```
class FetchItemsToDisplay(object):
    #sort_column: int, sort_order: int, results_per_page: int, page_ind
ex: int, results: Dict[str, Tuple[int, int]])
    def solution(self,URLS, sort_column, sort_order, results_per_page,
page_index):
        ordered = []
        for name in URLS:
            ordered.append((name,URLS[name][0],URLS[name][1]))
        ordered.sort(key=lambda x: x[sort_column], reverse=(sort_order
== False))
        start_index = results_per_page * page_index
        return [name for name, _, _ in ordered[start_index:start_index
+ results_per_page]]
def test(self):

    #  sort_column = int(input())
    #  sort_order = int(input())
    #  results_per_page = int(input())
```

```
    #  page_index = int(input())
    #  results_length = int(input())
    #  results = {
    #      n: (int(r), int(p))
    #      for _ in range(results_length)
    #      for n, r, p in [input().split()]
    #  }
    #  res = fetch_results_to_display(sort_column, sort_order, results_per_page, page_index, results)
    #  print(' '.join(res))

        res = self.solution({'p1':(10,5),'p2':(3,3),'p3':(17,4),'p4':(9
,4),'p5':(1,5)},1,False,3,1)
        print (res)
        res = self.solution({"foo.com":(10, 15), "bar.com":(3, 4),"baz.
com":(17, 8)},1,False,2,1)
        print (res)
```

Find Related Products

Amazon Online Assessment Feeds 2020

Find Related Products ⭐⭐

This question is based on the product recommendation system on Am
Every time you open the a product page on Amazon you can see a se
“People who viewed this also viewed”. Now given a product relationsh
represented as a graph(adjacent list), find out the largest connected
component on this graph.

Notice the graph is transitive.

For example:

Input:

[[“product1”, “product2”, “product3”]

[“product5”, “product2”]

[“product6”, “product7”]

[“product8”, “product7”]]

Output:

[“product1”, “product2”, “product3”, “product5”]

Explanation:

First we need to process the input and build the graph like this:

```
class FindRelatedProducts(object):
    def solution(self, items):
        def dfs(node):
            cnt = 1
            path =[node]
            visited.add(node)
            for next in graph[node]:
                if next not in visited:
                    nextcnt, nextpath = dfs(next)
                    cnt+=nextcnt
                    path.extend(nextpath)
            return cnt, path

        visited =set()
        graph = collections.defaultdict(set)
        for arr in items:
            for i in range(len(arr)-1):
                for j in range(i+1, len(arr)):
                    a , b = arr[i], arr[j]
                    graph[a].add(b)
                    graph[b].add(a)
        maxCnt = 0
        res=[]
        for p in graph:
            if p not in visited:
                cnt,path = dfs(p)
                if maxCnt < cnt:
                    maxCnt = cnt
                    res = list(path)
        return sorted(res)

    def test(self):
        res= self.solution([['p1','p2','p3'],['p5','p2'],['p6','p7'],['
p8','p7']])
        print(res)
        res= self.solution([['product1', 'product2'], ['product3', 'pro
duct4'], ['product5', 'product6'], ['product1', 'product3', 'product5']
])
        print(res)
```

Debt Records

An international organization is investigating debt across countries. Given a list of records representing amounts of money owed between each country, find the country with the largest negative balance.

Return the list consisting of the string "No countries have debt." if all countries zero out their owed amounts.

Input

`debts`: an array consisting of `borrower country` string, `lender country` string, `amount` number triplets, each representing a debt record.

Output

A list of countries with the largest debt. If there are multiple countries with the same maximum debt amount, sort them alphabetically.

Return a list containing the string `"No countries have debt."` if there is no debt.

Example

Borrower	Lender	Amount
USA	Canada	2
Canada	USA	2
Mexico	USA	5
Canada	Mexico	7
USA	Canada	4
USA	Mexico	4

Explanation:

For `USA`:

The `first`, `fifth`, and `sixth` entries decrease the balance because they are a `borrower`.

The `second` and `third` entries increase because they are a lender.

Their balance is `(2 + 5) - (2 + 4 + 4) = 7 - 10 = -3`.

For `Canada`:

They are a lender in `first` and `fifth` entries and a borrower in the `second` and `fourth` entries.

Their balance is `(2 + 4) - (2 + 7) = 6 - 9 = -3`.

For `Mexico`:

They are a borrower in the `third` entry and a lender in the `fourth` and `sixth` entries.

Thus, `Mexico's` balance is `(7 + 4) - 5 = 11 - 6 = 5`.

Here `USA` and `Canada` both have the balance of `-3`, which is the minimum net balance among all countries.

```
class DebtRecords(object):
    def solution(self,debts):
        balance ={}
        for borrower,lender,amount in debts:
            balance[borrower] = balance.get(borrower, 0) - int(amount)
            balance[lender] = balance.get(lender,0) + int(amount)
        maxdebt = min(balance.values())
        res=[]
        if maxdebt>=0:
            return res
        for country in balance:
            if  maxdebt == balance[country]:
                res.append(country)
        res.sort()
        return res
```

Unique Device(File) Names

You are asked to build a function to ensure uni
filename already exists in the system, an intege
the end of the filename to make it unique. The i
incremented by 1 for each new file with an exis
Given a list of filenames, write an algorithm to
in the order given.

Input

The input to the function/method consists of tw
num, an integer representing the number of file
filenames, a list of strings representing the filen

Output

Return a list of strings representing the filenam

Constraints

1 <= num <= 10^4

1 <= length of filenames[i] >= 20

0 <= i < num

```
class UniqueFileNames(object):
    def solution(self, names):
        nameCnt = collections.defaultdict(int)
        res=[]
        for name in names:
            if name not in nameCnt:
                res.append(name)
                nameCnt[name] = 1
            else:
                res.append(name+str(nameCnt[name]))
                nameCnt[name] +=1
        return res
    def test(self):
        res = self.solution(["system","access","access","system","acces
s","access"])
        print (res)
```

Labeling System

Given a string, construct a new string by rearranging the original string and deleting characters as needed. Return the alphabetically largest string that can be constructed respecting a limit as to how many consecutive characters can be the same.
Example:
s='bacc'
k=2
The largest string, alphabetically, is *'cccba'* but it is not allowed because it uses the character *'c'* more than 2 times consecutively. Therefore, the answer is *'ccbca'*.
Function Description
Complete the function *getLargestString* in the editor below.
getLargestString has the following parameters:
string *s[n]*: the original string
int *k*: the maximum number of identical consecutive characters the new string can have
Returns:
string: the alphabetically largest string that can be constructed that has no more than k identical consecutive characters
Constraints

- 1<= *n* <= 10^5
- 1<= *k* <= 10^3
- The string *s* contains only lowercase English letters.

Input Format For Custom Testing
Sample Case 0

Sample Input

STDIN Function

---

zzzazz --> string s = 'zzzazz'

2 --> k = 2

Sample Output

zzazz

Explanation

One '*z*' must be removed so that no more than 2 consecutive characters are the same.

```
class LabelingSystem(object):
    def solution(self, s, k):
        chrCounts = collections.Counter(s)
        chrs = [[c,chrCounts[c]] for c in chrCounts]
        chrs.sort(reverse= True)
        i , n  = 0 , len(chrs)
        res=[]
        while i < n:
            if chrs[i][1] > k:
                res.append(chrs[i][0]*k)
                chrs[i][1] -= k
                j = i+1
                while j<n and chrs[j][1] <= 0:
                    j+=1
                if j < n and chrs[j][1] > 0:
                    res.append(chrs[j][0])
                    chrs[j][1] -= 1
                else:
                    break
            elif chrs[i][1] > 0:
                res.append(chrs[i][0]*chrs[i][1])
                chrs[i][1] = 0
            else :
                i+=1
        return "".join(res)
    def test(self):
        res = self.solution("zzzzzzxxxzzaabbazza", 3)
        print(res)
        assert res == 'zzzxzzzxzzzxzbbaaa'
        res = self.solution("zzzazz",2)
        print(res)
        assert res =='zzazz'
        res = self.solution("baccc",2)
        print(res)
        assert res == 'ccbca'
```

Throttling Gateway

A planetarium has multiple entrances. Only the special Entrance X has cable cars carrying visitors into the planetarium while other entrances have walking tunnels. Everyone visiting the planetarium prioritizes entering from Entrance X.
An empty cable car arrives every minute at Entrance X and takes at most 3 passengers. For safety and better user experience, Entrance X has the following constraints:
The number of visitors on a cable car in any given minute cannot exceed 3.
The number of visitors going though Entrance X in any given 10-minute period cannot exceed 20.
A ten-minute period includes all visitors arriving from any time max(1, T-9) to T (inclusive of both) for any valid time T.
The number of visitors in any given hour cannot exceed 60. Similar to above, 1 hour is from max(1, T-59) to T.
Any visitor that exceeds any of the above limits will be assigned to other entrances instantly.
Given the times at which different visitors arrive sorted ascending, write an algorithm to find how many people will be assigned to other entrances.

Input
The input to the function consists of two arguments:
num, an integer representing the total number of visitors at X;
arriveTime, a list of integers representing the times of various visitor arrivals.
Output
Return an integer representing the total number of visitors NOT entering through Entrance X.

Constraints
1 <= num <= 10^6
1 <= arriveTime[i] <= 10^9
0 <= i< num

Note
Even if a visitor is assigned to other entrances, he/she is still considered for future calculations. Although, if a visitor is to be re-assign due to multiple constraints, he/she is still counted only once.

Example
Input:
num = 27
arriveTime = [1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7,7,7,7, 11, 11, 11, 11]
Output:
7

Explanation:
Visitor at 1 - Taken.
Visitor at 1 - Taken.
Visitor at 1 - Taken.
Visitor at 1 - Re-assigned. At most 3 visitors are allowed in one minute.
No visitor will be re-assigned till 6 as all comes at an allowed rate of 3 visitors per minute and the 10-minute clause is also not violated.

Visitor at 7 - Taken. The total number of visitors has reached 20 now.
Visitor at 7 - Re-assigned. At most 20 visitors are allowed in ten minutes.
Visitor at 7 - Re-assigned. At most 20 visitors are allowed in ten minutes.
Visitor at 7 - Re-assigned. At most 20 visitors are allowed in ten minutes. Note that the 1-minute limit is also violated here.
Visitor at 11 - Taken. The 10-minute window has now become 2 to 11. Hence the total number of visitors in this window is 20 now.
Visitor at 11 - Re-assigned. At most 20 visitors are allowed in ten minute s.
Visitor at 11 - Re-assigned. At most 20 visitors are allowed in ten minutes.
Visitor at 11 - Re-assigned. At most 20 visitors are allowed in ten minute s. Also, at most 3 visitors are allowed per minute.
Hence, a total of 7 visitors are re-assigned.

```
class ThrottlingGateway(object):
    def solution(self,time):
        n = len(time)
        res = 0
        for i in range(n):
            if i>2 and time[i]==time[i-3]:
                res+=1
            elif i>19 and time[i] - time[i-20] < 10:
                res+=1
            elif i > 59 and time[i] - time[i-60] < 60:
                res+=1
        return res

    def solution2(self,time):
        res = 0
        # this is to keep track of any of the element that is already d
ropped due to any of 3 limit violation.
        dropped = {}
        for i in range(len(time)):
            if i > 2 and time[i] == time[i-3]:
                if time[i] not in dropped or dropped[time[i]] != i:
                    dropped[time[i]] = i
                    res += 1

            elif i > 19 and time[i] - time[i-20] < 10:
                if time[i] not in dropped or dropped[time[i]] != i:
                    dropped[time[i]] = i
                    res += 1

            elif i > 59 and time[i] - time[i-60] < 60:
                if time[i] not in dropped or dropped[time[i]] != i:
                    dropped[time[i]] = i
                    res += 1
        return res
    def test(self):
        res = self.solution([1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5,
5, 5, 6, 6, 6, 7,7,7,7, 11, 11, 11, 11])
```

```
        print(res)
        res = self.solution([1, 1, 1, 1, 2])
        print(res)
        res = self.solution([1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5,
5, 5, 6, 6, 6, 7,7])
        print(res)
```

Nearest Cities

Given a list of points, find the nearest points that shares either an x or a y coordinate with the queried point.
The distance is denoted on a Euclidean plane: the difference in x plus the difference in y.
Input
numOfPoints, an integer representing the number of points;
points, a list of strings representing the names of each point [i];
xCoordinates, a list of integers representing the X coordinates of each point[i];
yCoordinates, a list of integers representing the Y coordinates of each point[i];
numOfQueriedPoints, an integer representing the number of points queried;
queriedPoints, a list of strings representing the names of the queried points.
Output
Return a list of strings representing the name of the nearest points that shares either an x or a y coordinate with the queried point.
Example 1:
Input:
numOfPoints = 3
points = ["p1","p2","p3"]
xCoordinates = [30, 20, 10]
yCoordinates = [30, 20, 30]
numOfQueriedPoints = 3
queriedPoints = ["p3", "p2", "p1"]
Output:
["p1", NONE, "p3"]
Example 2:
Input:
numOfPoints = 5
points = ["p1", "p2","p3", "p4", "p5"]
xCoordinates = [10, 20, 30, 40, 50]
yCoordinates = [10, 20, 30, 40, 50]
numOfQueriedPoints = 5
queriedPoints = ["p1", "p2", "p3", "p4", "p5"]
Output
[NONE, NONE, NONE, NONE, NONE]

```
class NearestCities(object):
    def solution(self, points, xCoord, yCoord, queriedPoints):
        n = len(points)
```

```
        mpx= collections.defaultdict(set)
        mpy= collections.defaultdict(set)
        coord = collections.defaultdict(list)
        for i in range(n):
            p, x, y = points[i], xCoord[i],yCoord[i]
            coord[p]=[x, y]
            mpx[x].add(p)
            mpy[y].add(p)
        res=[]
        for p in queriedPoints:
            dest='NONE'
            x , y = coord[p]
            dist = float('inf')

            for p_x in mpx[x]:
                if p_x != p:
                    if dist > abs(coord[p_x][1] - y):
                        dist = abs(coord[p_x][1] - y)
                        dest = p_x
            for p_y in mpy[y]:
                if p_y != p:
                    if dist > abs(coord[p_y][0]-x):
                        dist = abs(coord[p_y][0]-x)
                        dest = p_y
            res.append(dest)
        return res
    def test(self):
        res = self.solution(["a", "b", "c", "d","e"],[50, 60, 100, 200,
 300],[50, 60, 50, 200, 50], ["a", "b","c", "d", "e"])
        print(res)
        res = self.solution(["p1","p2","p3"],[30, 20, 10],[30, 20, 30],
 ["p3", "p2", "p1"])
        print(res)
        res = self.solution( ["p1", "p2","p3", "p4", "p5"],[10, 20, 30,
 40, 50], [10, 20, 30, 40, 50], ["p1", "p2", "p3", "p4", "p5"])
        print(res)
        res = self.solution(["green", "yellow", "red", "blue", "grey",
"pink"],[10, 20, 15, 30, 10, 15],[30, 25, 30, 40, 25, 25], ["grey", "bl
ue", "red", "pink"])
        print(res)
```

Schedule Deliveries - Earliest Time To Complete Deliveries

Give a list of piers each with 4 receiving docks, and a list of intake delivery times, return the earliest time to complete all deliveries.

Example

Input:

numOfPiers = 2
pierOpenTime = [7, 9]
deliveryTime = [7,6,3,4,1,1,2,0]

Output:
14

Explanation:
Assign the deliveries [2, 1, 6, 7] to pier 0 which opens at time 7.
The finishing time for pier 0 : 2 + 7 = 9, 1 + 7 = 8, 6 + 7 = 13, 7 + 7 = 14.
Assign the deliveries [3, 1, 4, 0] to pier 1 which opens at time 9.
The finishing time for pier 1 : 12, 10, 13, 9
The max finishing times is 14, which the earliest possible finish time.

```
class ScheduleDeliveries(object):
    def solution(self, n, open_times, delivery_time_cost):

        hq =[]
        for t in open_times:
            heapq.heappush(hq,t)
        delivery_time_cost.sort(reverse = True)
        res = 0
        m = len(delivery_time_cost)
        for i in range(0,m,4):
            t = heapq.heappop(hq)
            t += delivery_time_cost[i]
            res = max(t, res)
            heapq.heappush(hq,t)

        return res

    def test(self):
        res= self.solution(2,[8,10],[2,2,3,1,8,7,4,5])
        print(res)
        res= self.solution(2,[7,9], [7,6,3,4,1,1,2,0])
        print(res)
```

Multiprocessor System- Schedule Tasks

A processor is able to perform a certain number of tasks per hour.
Find the minimum number of hours required to schedule all the tasks, given that after each hour, the capacity of the processor that was last scheduled cuts in half.
For example, if a processor of capacity `5` is able to perform `5` tasks in the first hour, and in the second hour it is only eligible to perform `5/2 = 2` (rounded down) tasks.
Input

`capacity` = an array of numbers representing the capacity of each processor

`tasks` = the number of tasks to be completed

Output

Calculate the minimum number of hours required to complete all tasks.

Constraints

The number of tasks is an integer always greater than `1`. The capacity of all processors is also at least `1`. It is guaranteed that there is always sufficient capacity to complete all the tasks.

Examples

Example 1:

Input: `capacity = [3,1,7,2,4]`, `tasks = 15`

Output: `4`

Explanation:

In the first hour, assign the set of tasks to the processor with capacity `7`.
Once completed, the capacity of this processor drops to `7/2 = 3`, and the new state becomes `capacity = [3,1,3,2,4]`, and `tasks = 8`.
In the next hour, assign tasks to the processor with capacity `4`.
Then, `capacity = [3,1,3,2,2]`, and `tasks = 4`.
In the third hour, assign the next batch of tasks to processor with capacity `3`.
Then, `capacity = [1,1,3,2,2]`, and `tasks = 1`.
Assign the last task to processor with capacity `1` in the forth hour.
Therefore, it took `4` hours to complete all the tasks.

```
class ScheduleTasks(object):
    def solution(self, workers, task):
        hq = []
        for w in workers:
            heapq.heappush(hq, -w)
        res = 0
        while task > 0:
            power = -heapq.heappop(hq)
            task -=power
            power //= 2
            heapq.heappush(hq, -power)
            res+=1
        return res
    def test(self):
        res = self.solution([4,2,8,3,5],19)
        print (res)
```

Winning Sequence - Maximum Bounded Array

Given the lower and upper bound of a range of integers, find the largest "mountain array". A mountain array is defined as in the Peak of mountain array problem, i.e. An array that

- has at least 3 elements
- let's call the element with the largest value the "peak", with index `k`. The array elements monotonically increase from the first element to `A[k]`, and then monotonically decreases from `A[k + 1]` to the last element of the array. Thus creating a "mountain" of numbers.

If more tham one valid mountain arrays can be built from a given range of integers, the largest array is the one with the maximum values starting from the left side. For example, `[6, 7, 6, 5]` is larger than `[5, 6, 7, 5]` because first value is larger in the first array.

Return the largest mountain array satisfying the constraints, or -1 if it's not possible.

Examples

Example 1:

Input: `num = 4`, `lowerEnd = 3`, `upperEnd = 10`

Output: `[9 10 9 8]`

Example 2:

Input: `num = 5`, `lowerEnd = 1`, `upperEnd = 3`

Output: `[1 2 3 2 1]`

```
class WinningSequence(object):
    def solution(self, n, low, hi):
        dq=collections.deque([hi])
        n-=1
        num = hi
        while n>0:
            num -= 1
            if num < low:
                return []
            dq.append(num)
            n-=1
            if n>0:
                dq.appendleft(num)
            n-=1
        return list(dq)
```

```
    def test(self):
        res= self.solution(4,10,12)
        print(res)
        res= self.solution(5,1,3)
        print(res)
        res= self.solution(4,3,10)
        print(res)
```

1335. Minimum Difficulty of a Job Schedule

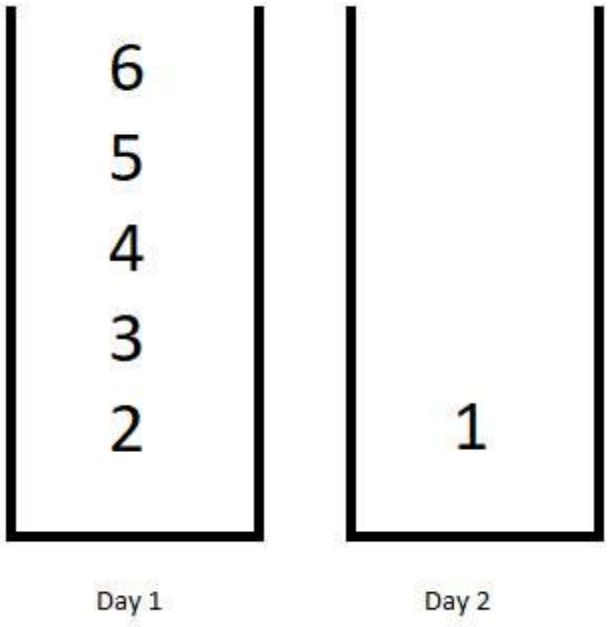
You want to schedule a list of jobs in `d` days. Jobs are dependent (i.e To work on the `i-th` job, you have to finish all the jobs `j` where `0 <= j < i`).

You have to finish **at least** one task every day. The difficulty of a job schedule is the sum of difficulties of each day of the `d` days. The difficulty of a day is the maximum difficulty of a job done in that day.

Given an array of integers `jobDifficulty` and an integer `d`. The difficulty of the `i-th` job is `jobDifficulty[i]`.

Return *the minimum difficulty* of a job schedule. If you cannot find a schedule for the jobs return **-1**.

Example 1:




```
Input: jobDifficulty = [6,5,4,3,2,1], d = 2
Output: 7
```

Explanation: First day you can finish the first 5 jobs, total difficulty = 6.
Second day you can finish the last job, total difficulty = 1.
The difficulty of the schedule = 6 + 1 = 7

Example 2:

Input: jobDifficulty = [9,9,9], d = 4
Output: -1
Explanation: If you finish a job per day you will still have a free day. you cannot find a schedule for the given jobs.

Example 3:

Input: jobDifficulty = [1,1,1], d = 3
Output: 3
Explanation: The schedule is one job per day. total difficulty will be 3.

Example 4:

Input: jobDifficulty = [7,1,7,1,7,1], d = 3
Output: 15

Example 5:

Input: jobDifficulty = [11,111,22,222,33,333,44,444], d = 6
Output: 843

Constraints:

- 1 <= jobDifficulty.length <= 300
- 0 <= jobDifficulty[i] <= 1000
- 1 <= d <= 10

$$Solution$$

```
class MinimumDifficultyofaJobSchedule:
    def minDifficulty(self, A: List[int], d: int) -> int:
        n = len(A)
        if n < d: return -1
        dp =[[float('inf')] *(n+1) for _ in range(d+1)]
        for i in range(d+1):
            dp[i][n] = 0

        for k in range(1,d+1):
            for i in range(n-k+1):
                maxd = 0
                for j in range(i, n-k+1):
                    maxd = max(maxd,A[j])
                    dp[k][i] =  min(dp[k][i], maxd + dp[k-1][j+1])
        print(dp)
        return dp[d][0]
```

Maximum Disk Space Available - The Max Of Minima

A user wants to store a file in a data center, but requests it to be replicated across each machine in a block. A block is defined as a continuous set of machines, starting from the first machine, with each block being next to one another and fixed in size. For example, if the block size is defined as 3, the first block is composed of machines 1 to 3, the second block is composed of machines 2 to 5, and so on.
Find the largest possible file the user can store in a data center, given a block size.
Input

`freeSpace`: a list of numbers representing the free space available in each machine of the data center

`blockSize`: a number representing the size of each block

Output

A number representing the amount of free space that the emptiest block in the data center has. The free space within a given block is the minimum free space of all the machines in it.

Constraints

The size of the block is always smaller than the number of machines in the `freeSpace` list. `freeSpace` values are never zero.

Examples

Example 1:

Input:

freeSpace = `[8,2,4,5]`

blockSize = `2`

Output: `4`

Explanation:

In this data center, the subarrays representing the free space of each block of size `2` are `[8,2]`, `[2,4]`, and `[4,5]`. The minimum available space of each blocks is `2`, `2`, and `4`. The maximum of these values is `4`. Therefore, the answer is `4`.

Complexity

Both time complexity and `O(n)` space complexity must be around `O(n)`.

```
class MaxOfMinima_MaximumDiskSpaceAvailable(object):
```

```
    def solution(self, nums, k):
        dq=collections.deque()
        n=len(nums)
        res=0
        for i in range(n):
            if dq and dq[0]==i-k:
                dq.popleft()
            while dq and nums[dq[-1]]>nums[i]:
                dq.pop()
            dq.append(i)
            if i>=k-1:
                res = max(res,nums[dq[0]] )
        return res
    def test(self):
        res = self.solution([8,2,4],2)
        print (res)
        res = self.solution([1, 6, 7, 4, 8, 11, 9],3)
        print (res)
        res = self.solution([3, 18, 40, 5, 5, 5, 7, 2],4)
        print (res)
        res = self.solution([62,64,77,75,71,60,79,75],4)
        print (res)
```

Transaction Logs

Your Amazonian team is responsible for maintaining a monetary transaction service. The transactions are tracked in a log file.
A log file is provided as a string array where each entry represents a transaction to service. Each transaction consists of:

- sender_user_id, Unique identifier for the user that initiated the transaction. It consists of only digits with at most 9 digits.
- recipient_user_id: Unique identifier for the user that is receiving the transaction. It consists of only digits with at most 9 digits.
- amount_of_transaction: The amount of the transaction. It consists of only digits with at most 9 digits.

The values are separated by a space. For example, "sender_user_id recipient_user_id amount_of_transaction".
Users that perform an excessive amount of transactions might be abusing the service so you have been tasked to identify the users that have a number of transactions over a threshold. The list of user ids should be ordered in ascending numeric value.
Example
logs = ["88 99 200", "88 99 300", "99 32 100", "12 12 15"]
threshold = 2
The transactions count for each user, regardless of role are:

ID	Transactions
99	3

88 2
12 1
32 1
There are two users with at least threshold = 2 transactions: 99 and 88. In ascending order, the return arra is ['88', '99'].

```
class TransactionLogs(object)
    def solution(logs, threshold):
        """
        :type logs: List[str]
        :type threshold: int
        :rtype: List[str]
        """
        mp = collections.defaultdict(int)
        for s in logs:
            x= s.split()
            sender =x[0]
            receiver = x[1]
            if sender == receiver :
                mp[sender]+=1
            else:
                mp[sender]+=1
                mp[receiver]+=1
        q=[]
        for user in mp:
            if mp[user]>=threshold:
                q.append(user)
        q.sort()
        return q
```

Packaging Automation

The Fulfillment Center consists of a packaging bay where orders are automatically packaged in groups(n). The first group can only have 1 item and all the subsequent groups can have one item more than the previous group. Given a list of items on groups, perform certain operations in order to satisfy the constraints required by packaging automation.
The conditions are as follows:
-The first group must contain 1 item only.
-For all other groups, the difference between the number of items in adjacent groups must not be greater than 1. In other words, for 1<=i<n, arr[i]-arr[i-1]<=1

To accomplish this, the following operations are available:
- Rearrange the groups in any way.
- Reduce any group to any number that is at least 1
Write an algorithm to find the maximum number of items that can be packaged in the last group with the conditions in place.

Input
The function/method consists of two arguments:
numGroups, an integer representing the number of groups(n);
arr, a list of integers representing the number of items in each group

Output
Return an integer representing the maximum items that can be packaged for the final group of the list given the conditions above.

Example1:
Input:
[3,1,3,4]
Output:
4
Explanation:
Subtract 1 from the first group making the list [2, 1, 3, 4]. Rearrange the list into [1, 2, 3, 4]. The final maximum of items that can be packaged in the last group is 4.
Example2:
Input:
[1,3,2,2]
Output:
3
Example3:
Input:
[1,1,1,1]
Output:
1
Example4:
Input:
[3,2,3,5]
Output:
4

```
    def solution(self, arr):
        arr.sort()
        for i in range(len(arr)):
            if i ==0:
                arr[i] = 1
            else:
                if arr[i] > arr[i-1]+1:
                    arr[i] = arr[i-1]+1
        return arr[-1]
    def test(self):
        res= self.solution([3,1,3,4])
        print(res)
        res= self.solution([1,3,2,2])
        print(res)
        res= self.solution([1,1,1,1])
        print(res)
        res= self.solution([3,2,3,5])
        print(res)
```

Rover Control

A Mars rover is directed to move within a square matrix. It accepts a sequence of commands to move in any of the four directions from each cell: [UP, DOWN, LEFT or RIGHT]. The rover starts from cell 0. and may not move diagonally or outside of the boundary.

Each cell in the matrix has a position equal to:
(row * size) + column
where row and column are zero-indexed, size = row length of the matrix.

Return the final position of the rover after all moves.

Example
n = 4
commands = [RIGHT, UP, DOWN, LEFT, DOWN, DOWN]

The rover path is shown below.

```
0  1  2  3
4  5  6  7
8  9  10 11
12 13 14 15
```

RIGHT: Rover moves to position 1
UP: Position unchanged, as the move would take the rover out of the boundary.
DOWN: Rover moves to Position 5.
LEFT: Rover moves to position 4
DOWN: Rover moves to position 8
DOWN: The rover ends up in position 12.

The function returns 12.

Function Description
Complete the function roverMove in the editor below.
roverMove has the following parameter(s):
int n: the size of the square matrix
string cmds[m]: the commands

Returns
int: the label of the cell the rover occupies after executing all commands

Constraints
2 <= n <= 20
1 <= |cmds| <= 20

Input Format For Custom Testing

Input from stdin will be processed as follows and passed to the function.
The first line contains an integer, n, denoting the size of the square matrix.
The next line contains an integer, m, the number of commands to follow.
Each of the next m lines contains a command string, cmds[i].

Sample Input :
STDIN Function
------- ----------
4 → n = 4
5 → cmds [] size m = 5
RIGHT → cmds = ['RIGHT', 'DOWN', 'LEFT', 'LEFT', 'DOWN']
DOWN
LEFT
LEFT
DOWN

Sample Output:
8

Explanation:
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

Rover starts at position 0
RIGHT → pos 1
DOWN → pos 5
LEFT → pos 4
LEFT → pos 4, No effect
DOWN → pos 8

```
class RoverControl(object):
    def solution(self, A, cmds):
        dir = {'RIGHT':(0,1), 'DOWN':(1,0), 'LEFT':(0,-1), 'UP':(-1,0)}
        m, n = len(A),len(A[0])
        row,col = 0, 0
        for cmd in cmds:
            dr,dc = dir[cmd]
            newRow =row+dr
            newCol =col+dc
            if newRow<0 or newRow >= m or newCol<0 or newCol>=n:
                continue
            row = newRow
            col = newCol
        return A[row][col]
    def test(self):
        res =self.solution([[0,1,2,3],[4,5,6,7],[8,9,10,11],[12,13,14,1
5]], ['RIGHT', 'UP', 'DOWN', 'LEFT', 'DOWN', 'DOWN'])
        print(res)
```

```
        res =self.solution([[0,1,2,3],[4,5,6,7],[8,9,10,11],[12,13,14,1
5]], ['RIGHT', 'DOWN', 'LEFT', 'LEFT', 'DOWN'])
        print(res)
```

Items In Containers

A company would like to know how much inventory exists in their closed inventory compartments. Given a string s
consisting of items as "*" and closed compartments as an open and close "|", an array of starting indices startIndices, and an array of ending indices endIndices, determine the number of items in closed compartments within the substring between the two indices, inclusive.

- An item is represented as an asterisk ('*' = ascii decimal 42)
- A compartment is represented as a pair of pipes that may or may not have items between them ('|' = ascii decimal 124).

Example

s = '|**|*|*'
startIndices = [1, 1]
endIndices = [5, 6]
The string has a total of 2 closed compartments, one with 2 items and one with 1 item.
For the first pair of indices, (1, 5), the substring is '|**|*'. There are 2 items in a compartment.
For the second pair of indices, (1, 6), the substring is '|**|*|' and there are 2 + 1 = 3 items in compartments.
Both of the answers are returned in an array, [2, 3].

Function Description

Complete the numberOfItems function in the editor below. The function must return an integer array that contains the results for each of the startIndices[i] and endIndices[i] pairs.
numberOfItems has three parameters:

- s: A string to evaluate
- startIndices: An integer array, the starting indices.
- endIndices: An integer array, the ending indices.

Constraints

1 ≤ m, n ≤ 10^5
1 ≤ startIndices[i] ≤ endIndices[i] ≤ n
Each character of s is either '*' or '|'

Input Format For Custom Testing

The first line contains a string, s.
The next line contains an integer, n, the number of elements in startIndices.
Each line i of the n subsequent lines (where 1 ≤ i ≤ n) contains an integer, startIndices[i].
The next line repeats the integer, n, the number of elements in endIndices.
Each line i of the n subsequent lines (where 1 ≤ i ≤ n) contains an integer, endIndices[i].

Sample Case 0

STDIN	Function
||	s = "*|*|"
1	startIndices[] size n = 1
1	startIndices = 1

1	endIndices[] size n = 1
3	endIndices = 3

Sample Output

0

Explanation

s = '*|*|'
n = 1
startIndices = [1]
n = 1
startIndices = [3]
The substring from index = 1 to index = 3 is '|'. There is no compartments in this string.

Sample Case 1

STDIN	Function
||*	s = "*|*|*|"
1	startIndices[] size n = 1
1	startIndices = 1
1	endIndices[] size n = 1
6	endIndices = 6

Sample Output

2

Explanation

s = '*|*|*|'
n = 1
startIndices = [1]
n = 1
startIndices = [1]
The substring from index = 1 to index = 6 is '||*|'. There are two compartments in this string at (index = 2, index = 4) and (index = 4, index = 6). There are 2 items between these compartments.

```
class  ItemsInContainers(object):
    #The prefixSum only applies to Pipe Characters
    #For any * elements. Two situations
    # 1. Start index -> * wants to match to its next Pipe
    # 2. End Index -> we want to match to to its previous Pipe
    #
    #'|**|*|*'
    # 0123456
    #
    #
    def solution(self,ss, ranges):
        n = len(ss)
        Idx =[0] * n
        Chars=[]
        starCount = 0
        res = []
        for i in range(n):
```

```
        Idx[i] = len(Chars)
        if ss[i] == '|':
            if not Chars:
                if ss[0]!='|' : starCount = 0
                Chars.append(starCount)
            else:
                Chars.append(starCount + Chars[-1])
            starCount = 0
        else:
            starCount += 1
    for begin, end in ranges:
        if begin < 0  or begin>= n or end <0  or end >=n:
            res.append(-1)
            continue
        beginIdx = Idx[begin]
        endIdx = Idx[end]
        if ss[end]!='|':
            endIdx -=1
        if endIdx > beginIdx :
            res.append(Chars[endIdx] - Chars[beginIdx])
        else:
            res.append(0)
    return res
```

Five-Star Sellers

Third-party companies that sell their products online are able to analyze the customer reviews for their products in real time. Imagine that there is creating a category called "five-star sellers" that will only display products sold by companies whose average percentage of five-star reviews per-product is at or above a certain threshold. Given the number of five-star and total reviews for each product a company sells, as well as the threshold percentage, what is the **minimum number of additional fivestar reviews** the company needs to become a five-star seller?

For example, let's say there are 3 products (n = 3) where productRatings = [[4,4], [1,2], [3, 6]], and the percentage ratings Threshold = 77. The first number for each product in productRatings denotes the number of fivestar reviews, and the second denotes the number of total reviews. Here is how we can get the seller to reach the threshold with the minimum number of additional five-star reviews:

- Before we add more five-star reviews, the percentage for this seller is ((4 / 4) + (1/2) + (3/6))/3 = 66.66%
- If we add a five-star review to the second product, the percentage rises to ((4 / 4) + (2/3) +(3/6))/3 = 72.22%
- If we add another five-star review to the second product, the percentage rises to ((4 / 4) + (3/4) + (3/6))/3 = 75.00%
- If we add a five-star review to the third product, the percentage rises to ((4/4) + (3/4) + (4/7))/3 = 77.38%

At this point, the threshold of 77% has been met. Therefore, **the answer is 3** because that is the minimum number of additional five-star reviews the company needs to become a five-star seller.

Function Description

Complete the function fiveStarReviews in the editor below.

fiveStarReviews has the following parameters:

int productRatings[n][2]: a 2-dimensional array of integers where the ith element contains two values, the first one denoting fivestar[i] and the second denoting total[i]

int ratingsThreshold: the threshold percentage, which is the average percentage of five-star reviews the products need for the company to be considered a five-star seller

Returns:

int: the minimum number of additional five-star reviews the company needs to meet the threshold ratingsThreshold

Constraints

- 1<=n<=200
- 0 <= fivestar<total<=100
- 1<=ratingsThreshold<100
- The array productRatings contains only non-negative integers.

```
class FiveStarSellers(object):
    def solution(self, n, arr, threshold):
        def increase(rate, total):
            return (rate+1) / (total+1) - rate / total
        current = 0

        hq = []
        for rate, total in arr:
            current += rate / total
            heapq.heappush(hq, (-increase(rate, total), rate , total))
        res = 0
        while current/n < threshold / 100.0:
            res+=1
            inc , rate, total = heapq.heappop(hq)
            current = current - inc
            rate, total = rate+1, total+1
            heapq.heappush(hq,(-increase(rate, total), rate , total))
        return res

    def test(self):
        res = self.solution(3, [[4,4], [1,2], [3, 6]], 77)
        print (res)
```

Maximize Profit

There are total N sellers. Each sellers has arr[i] items. Every time an i
sold the seller will raise the price by 1. And your profit on any item is
equals to the number of items the seller has left. Your job is to buy K
from N sellers and make the highest profit.

Input
N, an integer representing the number of sellers;
arr, a list of long integers representing the supply of the ith seller;
K, a long integer representing the number of items to be ordered.

Output
Return a long integer representing the highest profit that can be generated.

Constraints
1 <= N <= 100,000
1 <= arr[i] <= 100,000
0 <= i < N
1 <= K <= sum of arr

Example1:
Input:
N = 2
arr = [3, 4]
K = 6

Output:
15

Explanation:
Here seller one has 3 items. The final prices are [3, 2, 1].
Here seller two has 4 items. The final prices are [4, 3, 2, 1]
The highest profit is 4 + 2 * 3 + 2 * 2 + 1 = 15

Example2:

```
class MaximizeProfit(object):
    def solution(self,n,arr,k):
        hq= []
        for x in arr:
            heapq.heappush(hq,-x)
        res = 0
        for _ in range(k):

             x = heapq.heappop(hq)
             res -=x
             x+=1
             heapq.heappush(hq,x)
        return res

    def test(self):
        res= self.solution(2, [3,4],6)
        print (res)
        res= self.solution(5, [3,5,7,10,6],20)
        print (res)
```

Algorithm Swap

You're a new Amazon Software Development Engineer (SDE). You're reading through your team's code and find an old sorting algorithm. The following algorithm is used to sort an array of distinct n integers:

For the input array arr of size n do:
Try to find the smallest pair of indices 0 <= i < j <= n-1 such that arr[i] > arr[j]. Here smallest means usual alphabetical ordering of pairs, i.e. (i1, j1) < (i2, j2) if and only if i1 < i2 or (i1 = i2 and j1 <j2).
If there is no such pair, stop.
Otherwise, swap a[i] and a[j] and repeat finding the next pair.
The algorithm seems to be correct, but the question is how efficient is it? Write a function that returns the number of swaps performed by the above algorithm.

For example, if the initial array is [5,1,4,2], then the algorithm first picks pair (5,1) and swaps it to produce array [1,5,4,2]. Next, it picks pair (5,4) and swaps it to produce array [1,4,5,2]. Next, pair (4,2) is picked and swapped to produce array [1,2,5,4], and finally, pair (5,4) is swapped to produce the final sorted array [1,2,4,5], so the number of swaps performed is 4.

Function Description

Complete the function howManySwaps in the editor below. The function should return an integer that denotes the number of swaps performed by the proposed algorithm on the input array.

The function has the following parameter(s):
arr: integer array of size n with all unique elements

Constraints
1 <= n <= 10^5
1 <= arr[i] <= 10^9
all elements of arr are unique

Input Format Format for Custom Testing
Input from stdin will be processed as follows and passed to the function.
In the first line, there is a single integer n.
In the i-th of the next n lines, there is a single integer arr[i].

Sample Case
Sample Input
3
7
1
2

Sample Output
2

Explanation
There are 3 elements in the array, 7, 1 and 2 respectively.

Initially, there are two pairs of indices i < j for which a[i] > a[j]. These pairs are (0, 1) and (0, 2). Since (0, 1) is smaller of them, the algorithm swaps elements a[0] and a[1]. The resulting array is [1, 7, 2].
Next, in the second iteration there is only a single pair of indices i < j for which a[i> a[j]. This pair is (1, 2) and the algorithm swaps a[1] with a[2]. The resulting array is [1, 2, 7]. After that, the algorithm tries to find the next pair of indices to swap but since there is none, the algorithm stops.
The number of swaps it performed is 2.

```
class AlgorithmSwap(object):

    def solution(self,A):
        #self.cnt = 0
        #self.mergeSort(A, 0, len(A)-1)
        #print (A)
        #return self.cnt
        def merge(A, l,r):
            if r-l<2:return 0
            mid=l+(r-l)//2
            count=merge(A,l,mid)+merge(A,mid,r)
```

```
            i,j=l,mid
            tmp=0
            while i<mid and j < r:
                if A[i]<=A[j]:
                    i+=1
                else:
                    j+=1
                    tmp+=mid-i
            A[l:r]=sorted(A[l:mid]+A[mid:r])
            return tmp+count
        n=len(A)
        cnt=merge(A,0,n)
        return cnt


    def test(self):
        res =self.solution([5,1,4,2])
        print (res)
```

1041. Robot Bounded In Circle

On an infinite plane, a robot initially stands at `(0, 0)` and faces north. The robot can receive one of three instructions:

- `"G"`: go straight 1 unit;
- `"L"`: turn 90 degrees to the left;
- `"R"`: turn 90 degrees to the right.

The robot performs the `instructions` given in order, and repeats them forever. Return `true` if and only if there exists a circle in the plane such that the robot never leaves the circle.

Example 1:

```
Input: instructions = "GGLLGG"
Output: true
Explanation: The robot moves from (0,0) to (0,2), turns 180 degrees,
and then returns to (0,0).
When repeating these instructions, the robot remains in the circle of
radius 2 centered at the origin.
```

Example 2:

```
Input: instructions = "GG"
Output: false
Explanation: The robot moves north indefinitely.
```

Example 3:

```
Input: instructions = "GL"
```

```
Output: true
Explanation: The robot moves from (0, 0) -> (0, 1) -> (-1, 1) -> (-1, 0)
-> (0, 0) -> ...
```

Constraints:

`1 <= instructions.length <= 100`
`instructions[i]` is `'G'`, `'L'` or, `'R'`.

Intuition

Let chopper help explain.

Starting at the origin and face north `(0,1)`,
after one sequence of `instructions`,

1. if chopper return to the origin, he is obvious in an circle.
2. if chopper finishes with face not towards north,
   it will get back to the initial status in another one or three sequences.

Explanation

`(x,y)` is the location of chopper.
`d[i]` is the direction he is facing.
`i = (i + 1) % 4` will turn right
`i = (i + 3) % 4` will turn left
Check the final status after `instructions`.
Complexity
Time `O(N)`
Space `O(1)`

```
    def isRobotBounded(self, instructions):
        """
        :type instructions: str
        :rtype: bool
        """
        x,y=0,0
        s=instructions*4
        dx,dy=0,1
        for c in s:
            if c=='G':
                x,y=x+dx,y+dy
            elif c=='L':
                dx,dy=-dy,dx
            elif c=="R":
                dx,dy=dy,-dx

        return (x, y) == (0, 0) or (dx, dy) != (0,1)
```

Cutoff Ranks

A group of work friends at Amazon is playing a competitive video game together. During the game, each player receives a certain amount of points based on their performance. At the end of a round, players who achieve at least a cutoff rank get to "level up" their character, gaining increased abilities for them. Given the scores of the players at the end of the round, how many players will be able to level up their character?

Note that players with equal scores will have equal ranks, but the player with the next lowest score will be ranked based on the position within the list of all players' scores. For example, if there are four players, and three players tie for first place, their ranks would be 1,1,1, and 4. Also, no player with a score of O can level up, no matter what their rank.

Write an algorithm that returns the count of players able to level up their character.

Input

The input to the function/method consists of three arguments:

cutOffRank, an integer representing the cutoff rank for levelin up the player's character;

num, an integer representing the total number of scores;

scores, a list of integers representing the scores of the players.

Output

Return an integer representing the number of players who will be able to level up their characters at the end of the round.

Constraints

1 <= num <= 10^5

0 <= scores[i] <= 100

0 <= i < num

cutOffRank <= num

Examples

Example 1:

Input:

cutOffRank = 3

num= 4

scores=[100, 50, 50, 25]

Output:

3

Explanation:

There are num= 4 players, where the cutOffRank is 3 and scores = [100, 50,50, 25]. These players' ranks are [1, 2, 2, 4]. Because the players need to have a rank of at least 3 to level up their characters, only the first three players will be able to do so.

So, the output is 3.

Example 2:

Input:

cutOffRank = 4

num=5

scores=[2,2,3,4,5]

Output:

5

Explanation:

In order, the players achieve the ranks [4,4,3,2,1]. Since the cutOffRank is 4, all 5 players will be able to level up their characters.

So, the output is 5.

```
class CutOffRankS
    def countLevelUpPlayers(cutOffRank, num, scores):
        n = num
        mp = collections.Counter(scores)
        q = [(score, mp[score]) for score in mp ]
        q.sort(reverse = True)
        ranks = []
        rank = 1
        for score, cnt in q:
            ranks.extend([rank]*cnt)
            rank+=cnt
        idx = bisect.bisect_right(ranks,cutOffRank)
        return idx
```

Utilization Checks

A new Amazon-developed scaling computing system checks the average utilization of the computing system every second while it monitors. It implements an autoscale policy to add or reduce instances depending on the current load as described below. Once an action of adding or reducing the number of instances is performed, the system will stop monitoring for 10 seconds. During that time, the number of instances does not change.

- Average utilization < 25%: An action is instantiated to reduce the number of instances by half if the number of instances is greater than 1 (take the ceiling if the number is not an integer). If the number of instances is 1, take no action.
- 25% <= Average utilization <= 60%: Take no action.
- Average utilization > 60%: An action is instantiated to double the number of instances if the doubled value does not exceed 2* 10^8. If the number of instances exceeds this limit upon doubling, perform no action.

Given an array of integers that represent the average utilization at each second, determine the number of instances at the end of the time frame.

Example

instances = 2

averageUtil = [25, 23, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 76, 80]

- At second 1, the average utilization averageUtil[0] = 25 <= 25. Take no action.
- At second 2, the average utilization averageUtil[1] = 23 < 25. Reduce the number of instances by half, 2/2 = 1.
- Since an action was taken, the system will stop checking for 10 seconds, from averageUtil[2] to averageUtil[11].
- At averageUtil[12] = 76, 76 > 60 so the number of instances is doubled from 1 to 2.
- There are no more readings to consider and 2 is the final answer.

Function Description

Complete the function finallnstances in the editor below.

finallnstances has the following parameter(s):

int instances: an integer that represents the original number of instances running

int averageUtil[n]: an array of integers that represents the average utilization at each second of the time frame

Returns:
int: an integer that represents the final number of instances running.

Contraints

- 1 <= instances <= 10^5
- 1 <= n < 10^5
- 0 <= averageUtil[i] <= 100

Input Format For Custom Testing
Sample Case 0
Sample Input
STDIN Function
------- ----------
1 -> instances = 1
3 -> averageUtil[] size n = 3
5 -> averageUtil = [5, 10, 80]
10
80
Sample Output
2
Explanation

- At second 1. averageUtil[0] =5<25. The number of instances is 1. so take no action.
- At second 2. averageUtil[1] = 10 < 25. The number of instances is 1, so take no action.
- At second 3. averageUtil[2] = 80 > 60. An action is instantiated to double the number of instances from 1 to 2.

There are no more readings to consider and 2 is the final answer.
Sample Case 1
Sample Input
STDIN Function
------- ----------
5 -> instances = 5
6 -> averageUtil[] size n = 6
30 -> averageUtil = [30, 5, 4, 8, 19, 89]
5
4
8
19
89
Sample Output
3
Explanation

- At second 1,25 <= averageUtil[0] = 30 <= 60, so take no action.
- At second 2, averageUtil[1] = 5<25, so an action is instantiated to halve the number of instances to ceil(5/2) = 3.
- The system will stop checking for 10 seconds, so from averageUtil[2] through averageUtil[5] no actions will be taken.

There are no more readings to consider and 3 is the final answer.

```
class UtilizationChecks(object):
```

```
    def solution(self, instances, averageUtil):
        n = len(averageUtil)
        i = 0
        while i < n:
            if averageUtil[i] < 25 :
                if instances > 1:
                    instances = math.ceil(instances/2)
                    i+=10

            elif averageUtil[i] > 60:
                if instances < 2*10**8:
                    instances *= 2
                    #instances =  min(instances * 2 , 2*10**8)
                    i+=10

            i+=1
        return instances

    def test(self):
        res = self.solution(2,[25, 23, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 7
6, 80])
        print (res)
        #res = self.solution(5, [30, 5, 4, 8, 19, 89])
        #print (res)
        res = self.solution(1, [30, 15, 18, 18, 19, 89,  15 ,  18 ,  18
 ,  19 ,  89 ,  15 ,  18 ,  18 ,  19 ,  89 ,  15 ,  18 ,  18 ,  19 ,  8
9 ,  15 ,  18 ,  18 ,  19 ,  89 ])
        print (res)
```

1102. Path With Maximum Minimum Value (Medium)

Given a matrix of integers `A` with `R` rows and `C` columns, find the **maximum** score of a path starting at `[0,0]` and ending at `[R-1,C-1]`.
The *score* of a path is the **minimum** value in that path. For example, the value of the path 8 → 4 → 5 → 9 is 4.
A *path* moves some number of times from one visited cell to any neighbouring unvisited cell in one of the 4 cardinal directions (north, east, west, south).

Example 1:

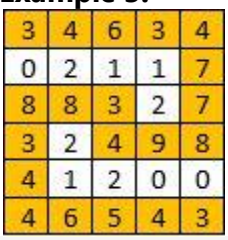
5	4	5
1	2	6
7	4	6

```
Input: [[5,4,5],[1,2,6],[7,4,6]]
Output: 4
Explanation:
The path with the maximum score is highlighted in yellow.
```

Example 2:

```
Input: [[2,2,1,2,2,2],[1,2,2,2,1,2]]
Output: 2
```

Example 3:



```
Input: [[3,4,6,3,4],[0,2,1,1,7],[8,8,3,2,7],[3,2,4,9,8],[4,1,2,0,0],[4,
6,5,4,3]]
Output: 3
```

Note:

1. 1 <= R, C <= 100
2. 0 <= A[i][j] <= 10^9

```
    def maximumMinimumPath(self, A):
        """
        :type A: List[List[int]]
        :rtype: int
        """
        """
BFS Dijkstra algorithm: use a priority queue to choose the next step
with the maximum value. Keep track of the mininum value along the
path.
        """


        pq = []
        # negate element to simulate max heap
        heappush(pq, (-A[0][0], 0, 0))
        m, n, maxScore = len(A), len(A[0]), A[0][0]
        visited = {(0,0)}
        while len(pq) != 0:
            val, i, j = heappop(pq)
            maxScore = min(maxScore, -val)
            if i == m - 1 and j == n - 1:
                break
            for dx , dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
                ni, nj = dx + i, dy + j
                if 0 <= ni < m and 0 <= nj <
n and (ni,nj) not in visited:
                    heappush(pq, (-A[ni][nj], ni, nj))
                    visited.add((ni,nj))
        return maxScore
```

```
#-------------

    def maximumMinimumPath(self, A):
        """
        :type A: List[List[int]]
        :rtype: int
        """
        """
We want to find a path from (0,0) to (n-1, m-1) w/ max lower bound. So
we just visit the cell in the order from largest to smallest, and use
UF to connect all the visited cells. Once we make (0,0) and (n-1, m-1)
connected, we know we get a path with max lower bound, which is just
the value of the last visited cell.

sort all the points in a descending order
union the point with the explored points until start and end has the
same parent
Time: O(MN log MN)
Space: O(MN)
        """

        class UnionFind(object):
            def __init__(self,n):
                self.parent = range(n)

            def find(self,p):
                while self.parent[p] != p:
                    self.parent[p] = self.parent[self.parent[p]]
                    p=self.parent[p]
                return p

            def union(self,p,q):
                rootp = self.find(p)
                rootq = self.find(q)
                if rootp != rootq:
                    self.parent[rootp] = rootq
            def connected(self,p,q):
                rootp = self.find(p)
                rootq = self.find(q)
                return rootp == rootq

        m, n =len(A), len(A[0])

        uf = UnionFind(m*n)
        q = [(A[i][j],i,j) for j in range(n) for i in range(m)]
        q.sort(reverse = True)

        visited=set()
        for _, i , j in q:
```

```
                visited.add((i, j))
                for dr, dc in [(0, -1), (0, 1), (1, 0), (-1, 0)]:
                    r, c = i + dr, j + dc
                    if 0 <= r < m and 0 <= c < n and (r, c) in visited:
                        uf.union(r * n + c, i * n + j)
                if uf.connected(0, m*n-1):
                    return A[i][j]
        return -1
```

150. Evaluate Reverse Polish Notation (Medium)

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, /. Each operand may be an integer or another expression.

Note:

- Division between two integers should truncate toward zero.
- The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation.

Example 1:

```
Input: ["2", "1", "+", "3", "*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: ["4", "13", "5", "/", "+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Example 3:

```
Input: ["10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+",
 "5", "+"]
Output: 22
Explanation:
  ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
```

```
= (0 + 17) + 5
= 17 + 5
= 22
```

```
    def evalRPN(self, tokens):
        """
        :type tokens: List[str]
        :rtype: int
        """
        q=[]
        for s in tokens:
            if s in "+-*/":
                b=q.pop()
                a=q.pop()
                if s=="+" : q.append(a+b)
                if s=="-" : q.append(a-b)
                if s=="*" : q.append(a*b)
                if s=="/" : q.append(int(operator.truediv(a, b)))
            else:
                q.append(int(s))

        return (q[0])
```

Split String Into Unique Primes

https://www.geeksforgeeks.org/count-of-ways-to-split-a-given-number-into-prime-segments/