

## Algorithms Stanford



## Week 1

Introduction, Union find and analyse of Algorithms.

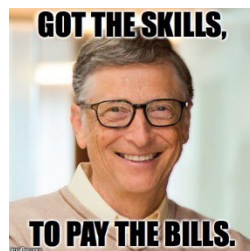
Introduction

topic	data structures and algorithms
data types	stack, queue, bag, union-find, priority queue
sorting	quicksort, mergesort, heapsort
searching	BST, red-black BST, hash table
graphs	BFS, DFS, Prim, Kruskal, Dijkstra
strings	radix sorts, tries, KMP, regexps, data compression
advanced	B-tree, suffix array, maxflow

part 1

part 2

*Reason to study algorithms*



1. Fun and profit.



2. Power AI
3. Better code

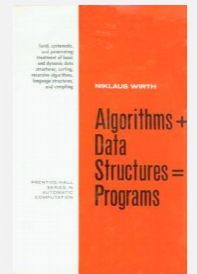


4. Brain stimulation

([Maths build brain muscles](#))

## 5. Become best programmer.

*“Algorithms + Data Structures = Programs.” — Niklaus Wirth*

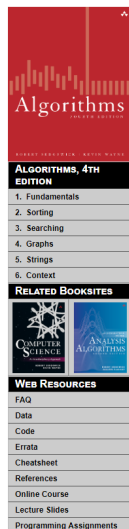


## Extra Materials

### Book



## Book web page



### ALGORITHMS, 4TH EDITION

essential information that  
every serious programmer  
needs to know about  
algorithms and data structures

**Textbook.** The textbook *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne [ [Amazon](#) · [Pearson](#) · [InformIT](#) ] surveys the most important algorithms and data structures in use today. We motivate each algorithm that we address by examining its impact on applications to science, engineering, and industry. The textbook is organized into six chapters:

- *Chapter 1: Fundamentals* introduces a scientific and engineering basis for comparing algorithms and making predictions. It also includes our programming model.
- *Chapter 2: Sorting* considers several classic sorting algorithms, including insertion sort, mergesort, and quicksort. It also features a binary heap implementation of a priority queue.
- *Chapter 3: Searching* describes several classic symbol-table implementations, including binary search trees, red-black trees, and hash tables.
- *Chapter 4: Graphs* surveys the most important graph-processing problems, including depth-first search, breadth-first search, minimum spanning trees, and shortest paths.
- *Chapter 5: Strings* investigates specialized algorithms for string processing, including radix sorting, substring search, tries, regular expressions, and data compression.
- *Chapter 6: Context* highlights connections to systems programming, scientific computing, commercial applications, operations research, and intractability.

**Booksite.** Reading a book and surfing the web are two different activities. This booksite is intended for your use while online (for example, while programming and while browsing the web); the textbook is for your use when initially learning new material and when reinforcing your understanding of that material (for example, when reviewing for an exam). The booksite consists of the following elements.

- *Excerpts.* A condensed version of the text narrative, for reference while online.
- *Java code.* The algorithms and clients [ [algs4](#) · [github](#) ] in this textbook.
- *Exercise solutions.* Solutions to selected exercises.

#### For students:

- *Java.* Here are instructions for setting up an IntelliJ-based Java programming environment for [Mac OS X](#), [Windows](#), and [Linux](#).
- *Lecture videos.* The *deluxe edition* includes professionally produced lecture videos.
- *Online course.* You can take our free Coursera MOOCs *Algorithms, Part I* and *Algorithms, Part II*.

#### For instructors:

For more information, see the [Instructor's Guide](#) or contact your Pearson representative.

## Union Find

the number of components. Thus, we articulate the following API:

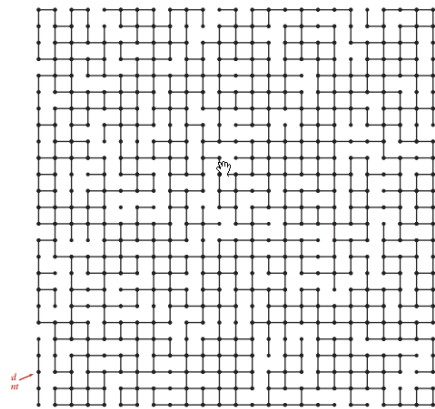
```
public class UF
```

<code>UF(int N)</code>	<i>initialize N sites with integer names (0 to N-1)</i>
<code>void union(int p, int q)</code>	<i>add connection between p and q</i>
<code>int find(int p)</code>	<i>component identifier for p (0 to N-1)</i>
<code>boolean connected(int p, int q)</code>	<i>return true if p and q are in the same component</i>
<code>int count()</code>	<i>number of components</i>

Union-find API

### Problem

The problem is when the big variety of object



Applications involved in in connectivity problem

1. Pixels in digital photos
2. Computers in a networks
3. Friends in social network
4. Transistors in a computer chip
5. Elements in mathematical set
6. Variables in Fortran program
7. Metallic seites in composite system

### Quick Find

It is very simple to implement

```

6 references | 0 changes | 0 authors, 0 changes
public override int Find(int p){
    return Id[p];
}

2 references | 0 changes | 0 authors, 0 changes
public override void Union(int p, int q){
    int pId = Find(p);
    int qId = Find(q);

    if (pId == qId)
        return;

    for (int i = 0; i < Id.Length; i++){
        if (Id[i] == pId)
            Id[i] = qId;
    }

    UnionCount--;
}

```

Analyze

So the algorithm is very slow

**Cost model.** Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	N	N	1

order of growth of number of array accesses

To Union all elements it will take to run Union Operation N times what will be  $O(N^2)$  what is super slow.

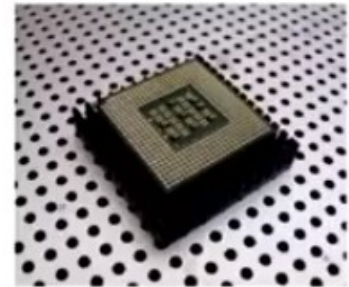
And again quadratic algorithm is too slow to be performed. So we need to find a better way.

## Quadratic algorithms do not scale

Rough standard (for now).

- $10^9$  operations per second.
- $10^9$  words of main memory.
- Touch all words in approximately 1 second.

a truism (roughly)  
since 1950!

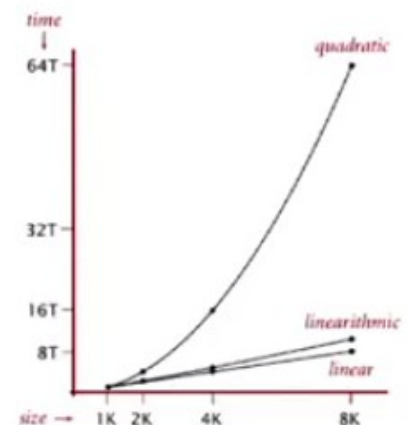


Ex. Huge problem for quick-find.

- $10^9$  union commands on  $10^9$  objects.
- Quick-find takes more than  $10^{18}$  operations.
- 30+ years of computer time!

Quadratic algorithms don't scale with technology.

- New computer may be 10x as fast.
- But, has 10x as much memory  $\Rightarrow$  want to solve a problem that is 10x as big.
- With quadratic algorithm, takes 10x as long!



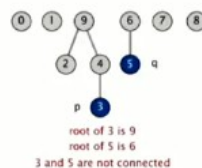
### QuickUnion

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[...id[i]...]]`.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9

find. Check if `p` and `q` have the same root.



Implementation

```

9 references | 0 changes | 0 authors, 0 changes
public override int Find(int p){
    while (Id[p]!=p)
        p = Id[p];
    return p;
}

3 references | 0 changes | 0 authors, 0 changes
public override void Union(int p, int q){
    int pRoot = Find(p);
    int qRoot = Find(q);

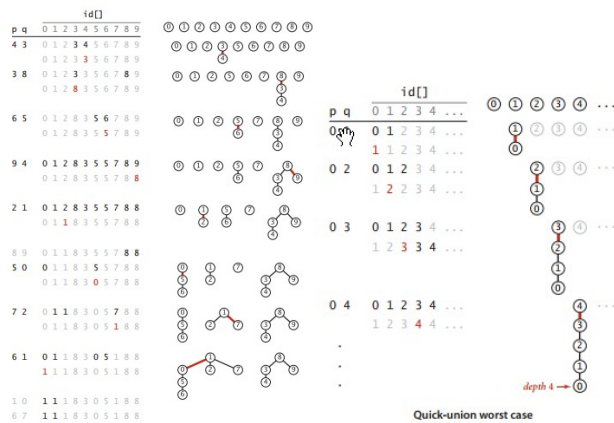
    Id[pRoot] = qRoot;

    UnionCount--;
}

```

## Analyze

So in perfect world we have those trees that speed up find operation.

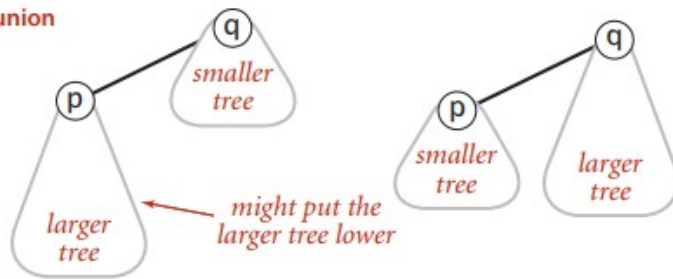


We

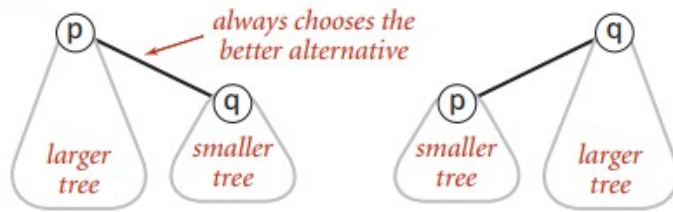
But some times we will have it linear tree. So it would be too slow to use Quick Union

## Weighted Quick Union

### quick-union



### weighted



### Weighted quick-union

```
public class QuickUnionWeighted:AbstractUnionFind{
    private int[] _size;

    public QuickUnionWeighted(int count) : base(count){
        _size = Enumerable.Repeat(1, count).ToArray();
    }

    public override int Find(int p){
        while (Id[p] != p)
            p = Id[p];
        return p;
    }

    public override void Union(int p, int q){
        int rootP = Find(p);
        int rootQ = Find(q);

        if (rootQ == rootP)
            return;

        if (_size[rootP] < _size[rootQ]){
            Id[rootP] = rootQ;
            _size[rootQ] += _size[rootP];
        }
        else{
            Id[rootQ] = rootP;
            _size[rootP] += _size[rootQ];
        }
    }
}
```

In that way our tree can maximum can be

Analyze

$\lg N$

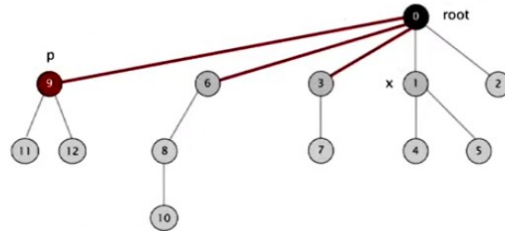
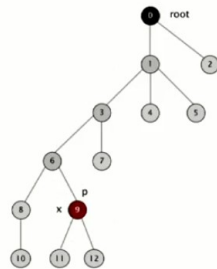


algorithm	initialize	union	connected
quick-find	$N$	$N$	1
quick-union	$N$	$N^\dagger$	$N$
weighted QU	$N$	$\lg N^\dagger$	$\lg N$

† includes cost of finding roots

### Path Compression

Quick union with path compression. Just after computing the root of  $p$ , set the id of each examined node to point to that root.



1 Line of Code!!!!

```
private int root(int i)
{
    while (i != id[i])
    {
        id[i] = id[id[i]];
        i = id[i];
    }
    return i;
}
```

```
12 references | 0 changes | 0 authors, 0 changes
public override int Find(int p){
    while (Id[p] != p){
        Id[p] = Id[Id[p]];
        p = Id[p];
    }

    return p;
}
```

## Analyze

N	$\lg^* N$
1	0
2	1
4	2
16	3
65536	4
$2^{65536}$	5

iterate log function

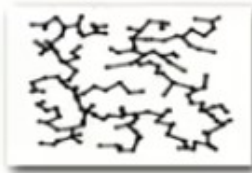
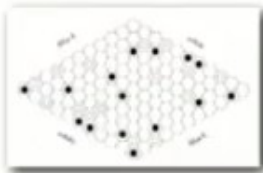
Path Compression is close to be linear.

## Analyze

algorithm	order of growth for $N$ sites (worst case)		
	constructor	union	find
<i>quick-find</i>	$N$	$N$	1
<i>quick-union</i>	$N$	tree height	tree height
<i>weighted quick-union</i>	$N$	$\lg N$	$\lg N$
<i>weighted quick-union with path compression</i>	$N$	very, very nearly, but not quite 1 (amortized) (see EXERCISE 1.5.13)	
<i>impossible</i>	$N$	1	1

## Application

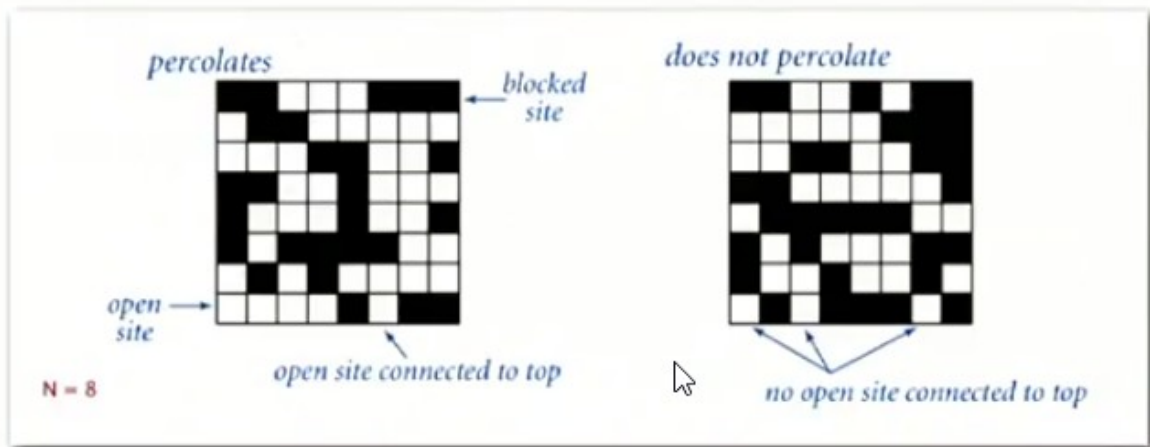
- Percolation.
- Games (Go, Hex).
- ✓ Dynamic connectivity.
  - Least common ancestor.
  - Equivalence of finite state automata.
  - Hoshen-Kopelman algorithm in physics.
  - Hinley-Milner polymorphic type inference.
  - Kruskal's minimum spanning tree algorithm.
  - Compiling equivalence statements in Fortran.
  - Morphological attribute openings and closings.
  - Matlab's `bwlabel()` function in image processing.



## Percolation

A model for many physical systems:

- $N$ -by- $N$  grid of sites.
- Each site is open with probability  $p$  (or blocked with probability  $1 - p$ ).
- System **percolates** iff top and bottom are connected by open sites.



Likelihood of percolation

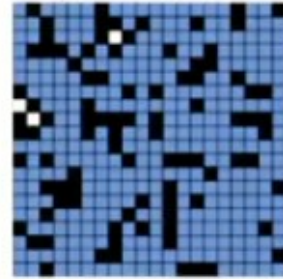
Depends on site vacancy probability  $p$ .



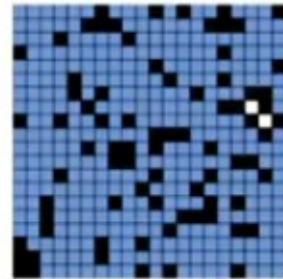
$p$  low (0.4)  
does not percolate



$p$  medium (0.6)  
percolates?



$p$  high (0.8)  
percolates

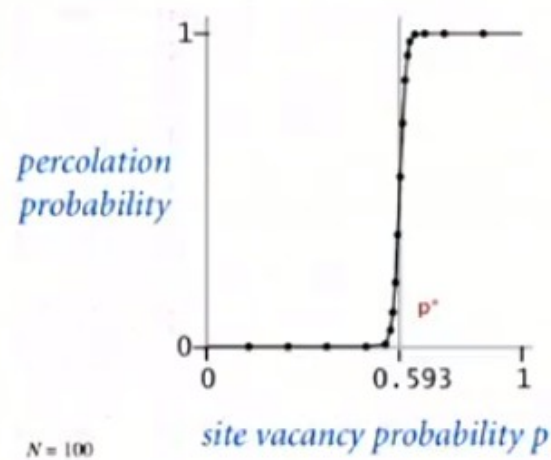


## Percolation phase transition

When  $N$  is large, theory guarantees a sharp threshold  $p^*$ .

- $p > p^*$ : almost certainly percolates.
- $p < p^*$ : almost certainly does not percolate.

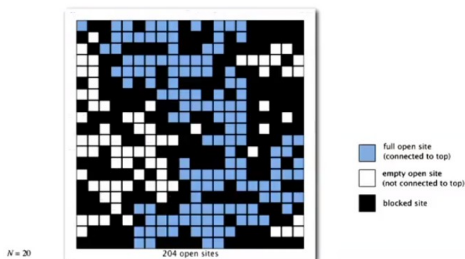
Q. What is the value of  $p^*$  ?



## Monte Carlo simulation

### Monte Carlo simulation

- Initialize  $N$ -by- $N$  whole grid to be blocked.
- Declare random sites open until top connected to bottom.
- Vacancy percentage estimates  $p^*$ .



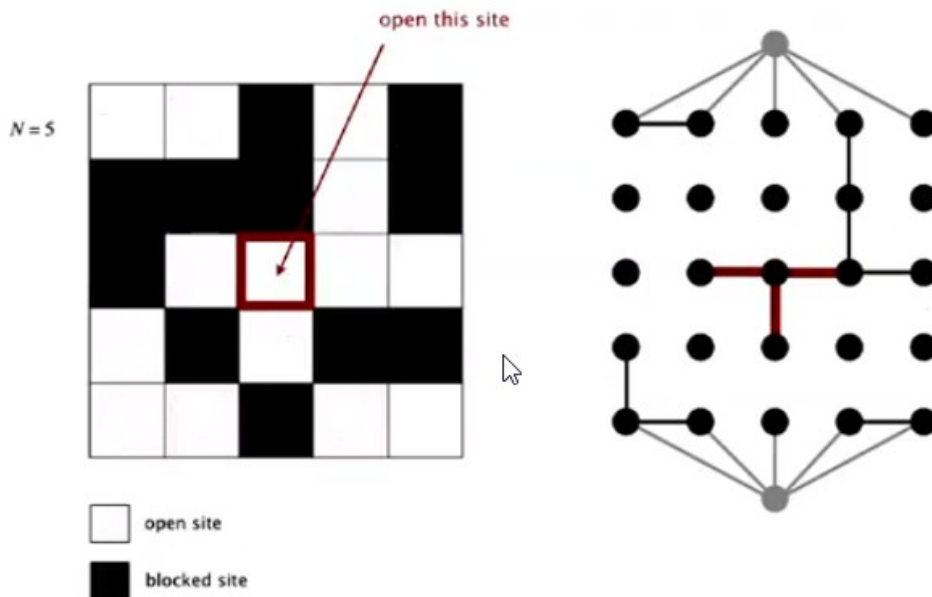
There is Monte Carlo simulation where we open block by block to see if there is percolation or not

## Dynamic connectivity solution to estimate percolation threshold

Q. How to model opening a new site?

A. Connect newly opened site to all of its adjacent open sites.

up to 4 calls to union()



60

Here is the way to represent the Percolation problem using Quick Union way.

I can do it on my computer and run it infinite amount of times and get result for my self.

Codility

**codility** for Programmers

So codility have this lesson sequence that is used for TopTal but I would like to get max score to apply for toptal.

[Lesson 1 : Iteration](#)

For, while, foreach

Basically it talks about iterations.

[Binary Gap](#)

The task is basically to made while cycle together with while operation.

So it is very basic and siple

## Lesson 2: Arrays

Reading material focus on basic arrays operations like element value changing and array length determination. It focus on reversing the array

### CyclicRotation

The code is very basic just shift the array for count of elements.

```
6 references | Yuriy Paramonov, 1 day ago | 1 author, 1 change
public static int[] TaskSolution(int[] A, int K){
    int[] result = new int[A.Length];

    for (int i = 0; i < A.Length; i++){
        K %= A.Length;
        int newIndex = K - i > 0 ? A.Length - K + i : i - K;
        result[i] = A[newIndex];
    }
    return result;
}
```

### Hidden trick

So the trick to the task was in one non describe detail

Number to shift K can be longer then array length

In such case for shifting our array we need to remove extra length by preforming

$K \% = A.Length;$

And also you need to done task in  $O(N)$  time.

### OddConcurrencyInArray

So all you need is mark number that alredy be counted.

And then check all the checked number and find one that wasn't dubble checked.

### Hidden trick

The number can be to big so you need to go smart to mark it. You can't use as mutch space as you would like so you need to find way to count them on minimum space so I used trick with binary operations where each bit represent one number



And then using this technique I achieved enough in total to get what I need.

```
1 reference | 0 changes | 0 authors, 0 changes
36 public static bool NumberIsCheck(int a, int number)
37 {
38     int shifted = 1 << number - 1;
39     return (a & shifted) == shifted;
40 }
41
1 reference | 0 changes | 0 authors, 0 changes
42 public static int MarkNumber( int a, int number)
43 {
44     int shifted = 1 << number - 1;
45     return a ^ shifted;
46 }
47
```

### Lesson 3: Time complexity

Now it basically talking about the  $O(n)$

About types I already well-known  $O(n)$   $O(1)$   $O(1)$  ,  $O(1)$  .

The good example case in the end in task where you need count

Arithmetical progression  $1+2+3+4+\dots+N$  instead of using

You can do it very bad:

#### 3.7: Slow solution — time complexity $O(n^2)$ .

```
1 def slow_solution(n):
2     result = 0
3     for i in xrange(n):
4         for j in xrange(i + 1):
5             result += 1
6     return result
```

Bad:

#### 3.8: Fast solution — time complexity $O(n)$ .

```
1 def fast_solution(n):
2     result = 0
3     for i in xrange(n):
4         result += (i + 1)
5     return result
```

Or basically remember that it was in math and it's basic arithmetical progression and you can easily apply to get result.

### 3.9: Model solution — time complexity $O(1)$ .

```
1 def model_solution(n):  
2     result = n * (n + 1) // 2  
3     return result
```

#### FrogJump

So the solution is just count the count of jumps in arithmetical way instead of any iteration.

#### Hidden Trick

So there is always can be situation when you have or not have extra jump

But the calculation is based on division so you need manually perform operation if there is some rest after division then add one extra jump because frog needs to jump one more time to throw that distance.

#### PermMissingElement

The task was simply to find missing element using the lowest possible algorithm complexity.

#### Hidden trick

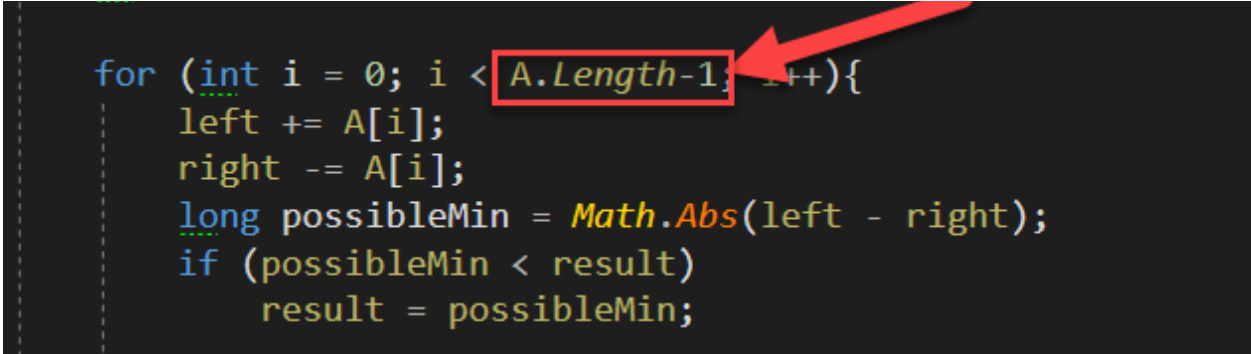
1. Using algebraic progression
2. Using longs to get total sum of elements.

#### TypeEquilibrium

So the task was kind of tricky. So the solution was to make sum of all elements.

#### Hidden trick

1. Use longs for calculation sum
2. Pay attention to the minimum numbers
3. Attention on iteration of two slices it counts too much elements but last shouldn't be counted



```
for (int i = 0; i < A.Length-1; i++){  
    left += A[i];  
    right -= A[i];  
    long possibleMin = Math.Abs(left - right);  
    if (possibleMin < result)  
        result = possibleMin;
```

So if I have the three number

{100, 5, -100}

We should have 2 slices

$P[0] = |100 - -95| = 195$

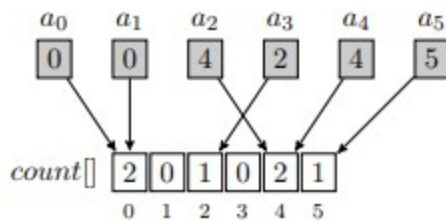
$P[1] = |105 - -100| = 205$

But if we let got all iteration of array what 3 then we will have one extra elemtn

$P[2] = [5 - 0] = 5$

That would be mistake and it would not be seen in positive number but in negative it would be issaly missanderstand.

#### Lesson 4 : Counting Elements



So here we have system where each element is counted and count index stores the count of elements with such number But for such operation we need first of all know the max element.

And there are were interesting example with swaping one element

Where obvious solution is  $O()$  but my the smart moves we come to the length  $O(n+m)$ .

Very smart and interesting example.

#### FrogRiverOne

So the trick was in arithmetical progression Using counting technique in mix.

##### Hidden trick

1. A lot of test
2. Using longs for total sum
3. Mixed technique of counting.

#### Perm Check

So the task was just detect is there any missing element or not.

It was really basic if you think about it but. There is one major thing I should get noted about.

##### Hidden trick

1. Mix counting technique

- N is an integer within the range [1..100,000],
- each element of array A is an integer within the range [1..1,000,000,000].

2. IN the task description is clue that our integer can be bigger than our array. So we just check is array max bigger then array length.
3. Our our main integer could be smaller then array length.

### Missing integer

So I need to find minimal positive inateger but ther could minus number just minus number and just queue of numbers.

### Hidden trick

Unit testing. Unit testing for ever evething.

### Max Counters

It's easy you just do old trick

### Hidden trick

It's sounds like you should made  $O(N*M)$  by description, but you should actually made  $O(N+M)$

## Lesson 5: Prefix sum

So we have been give the some array and we create new array containing the prefix sums of all possible combinations of first array.

	$a_0$	$a_1$	$a_2$	...	$a_{n-1}$
$p_0 = 0$	$p_1 = a_0$	$p_2 = a_0 + a_1$	$p_3 = a_0 + a_1 + a_2$	...	$p_n = a_0 + a_1 + \dots + a_{n-1}$

It allow us to count the

### 5.1: Counting prefix sums — $O(n)$ .

```
def prefix_sums(A):
    n = len(A)
    P = [0] * (n + 1)
    for k in xrange(1, n + 1):
        P[k] = P[k - 1] + A[k - 1]
    return P
```

### 5.2: Total of one slice — $O(1)$ .

```
1 def count_total(P, x, y):
2     return P[y + 1] - P[x]
```

### Mushroom picker

So there is this exercise where you need to count how many mushrooms can pick the mushroom picker.

About the guy who pick the mushrooms.

2	3	7	5	1	3	9
0	1	2	3	4	5	6

The exercise is kind of complex so I would like to sort of rewrite it.

### Passing Cars

Simple task where we just need to use the prefix sum technique.

### Genomic range query

More fun task But the basic is the same all you need to do is made multiple prefix sums.

```
// A, C, G and T have impact factors of 1, 2, 3 and 4,
int[] prefixA = new int[S.Length + 1];
int[] prefixC = new int[S.Length + 1];
int[] prefixG = new int[S.Length + 1];

for (int i = 1; i < S.Length+1; i++){
    prefixA[i] = prefixA[i - 1] + (S[i-1] == 'A' ? 1 : 0);
    prefixC[i] = prefixC[i - 1] + (S[i-1] == 'C' ? 1 : 0);
    prefixG[i] = prefixG[i - 1] + (S[i-1] == 'G' ? 1 : 0);
}
```

I would say not that hard.

### Minimal average of two slice

Task is hard or sort of and from the first attempt I get only 60%

## Candidate Report: Anonymous

Test Name:

[SUMMARY](#) [TIMELINE](#)

Test Score

60 out of 100 points

60%

Tasks in Test



MinAvgTwoSlice  
Submitted in: C#

Time Spent ⓘ

59 min

Task Score

60%

### TASKS DETAILS

MEDIUM

#### 1. MinAvgTwoSlice

Find the minimal average of any slice containing at least two elements.

Task Score

60%

Correctness

60%

Performance

60%

Then I found that guy who has kind of hobby of solving codility and Hacker rank tasks.

MartinKysel.com  
CODE MADE HUMAN

[Home](#) [Codility Solutions](#) [HackerRank Solutions](#) [About](#)



CODE MADE HUMAN

Prove by mathematical that any sequence can be made out of sub-sequence of 2 or 3 elements.

And even mathematical prove that:

**Problem.** See [https://codility.com/programmers/task/min\\_avg\\_two\\_slice/](https://codility.com/programmers/task/min_avg_two_slice/)

Martin Kysel claimed that we only need to check slices of size 2 and 3 as the min average should be in the sub-slices.<sup>1</sup> In this note, I just want to give a formal proof for this observation.

Without loss of generality, consider a non-empty array  $A = A_1, A_2, \dots, A_N$  and an integer  $1 < i < N$ . We consider the average of  $A$  and its two sub-arrays  $A_1, \dots, A_i$  and  $A_{i+1}, \dots, A_N$ . Let

$$B = \frac{A_1 + \dots + A_N}{N} \quad C = \frac{A_1 + \dots + A_i}{i} \quad D = \frac{A_{i+1} + \dots + A_N}{N - i}.$$

We claim that either  $C \leq B$  or  $D \leq B$  ( $\star$ ). Indeed, assume the opposite, that is,  $B < C$  and  $B < D$ . We have that:

$$\begin{aligned} B < C &\Leftrightarrow \frac{A_1 + \dots + A_N}{N} < \frac{A_1 + \dots + A_i}{i} \\ &\Leftrightarrow i \times (A_1 + \dots + A_N) < N \times (A_1 + \dots + A_i) \\ &\Leftrightarrow i \times (A_{i+1} + \dots + A_N) < (N - i) \times (A_1 + \dots + A_i) \\ &\Leftrightarrow \frac{A_{i+1} + \dots + A_N}{N - i} < \frac{A_1 + \dots + A_i}{i} \\ &\Leftrightarrow D < C \end{aligned}$$

Similarly,  $B < D \Leftrightarrow C < D$ .

Then  $B < C$  and  $B < D$  means  $D < C$  and  $C < D$ , a contradiction! Therefore, claim ( $\star$ ) holds. What follows is simple, since a slice of size bigger than 3 is composed of sub-slices of size 2 or 3.

What is sort of awesome.

## CountDiv

So the task was to count the div between number find the amount of number dividebla by K between A and B. The solution is so simple that I would even not write the description.

```
8 references | 0 changes | 0 authors, 0 changes
public static int TaskSolution(int A, int B, int K){
    int result = (B / K) - (A / K);

    if (A % K == 0)
        result++;

    return result;
}
```

## Lesson 6: Sorting

So whole idea of the lesson goes around our opportunitie to sort things and we have this idea of using counter of elemnts as sorting method.

finally, just count the number of distinct pairs in adjacent cells.

#### 6.4: The number of distinct values — $O(n \log n)$ .

```
1 def distinct(A):
2     n = len(A)
3     A.sort()
4     result = 1
5     for k in xrange(1, n):
6         if A[k] != A[k - 1]:
7             result += 1
8     return result
```

In the example task we need to find count of unique values in the array N.

And what w

Distinct

Basically the same task as it was in the beginning but array can be empty so just notice that detail but otherwise normal sorting

MaxProductOfThree

Task was to find max multiplication of 3 elements in the array the hidden trick was that there could be minus number. If the biggest are two last first numbers

So it can be  $A[0] * A[1] * A[N]$  or  $A[N] * A[N-1] * A[N-2]$  if the biggest are positive numbers.

Easy task.

Triangle

So the thing was to find triangle. The solution was easy just sort it. And look for 3 elements one after other. If there such sequence then we have triangle one trick in task description

- N is an integer within the range [0..100,000];
- each element of array A is an integer within the range [-2,147,483,648, 2,147,483,647].

The sum of 2 sides can be sum of 2 int.MaxValue what is more then int so we need to use long.

But meanwhile it is simple task.

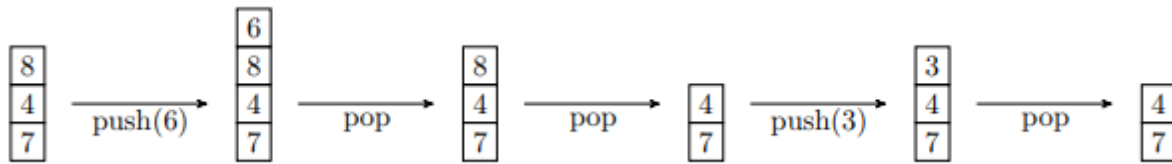
## Lesson 7: Stack and Queues

```
Stack<int> stack = new Stack<int>();
Queue<int> queue = new Queue<int>();
```

- in C#

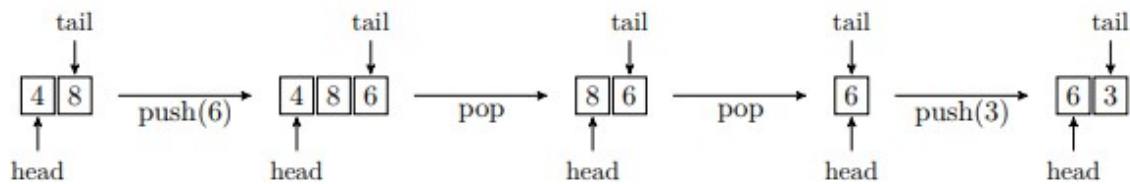


## Stack



## FIFO(First In First Out)

## Queue

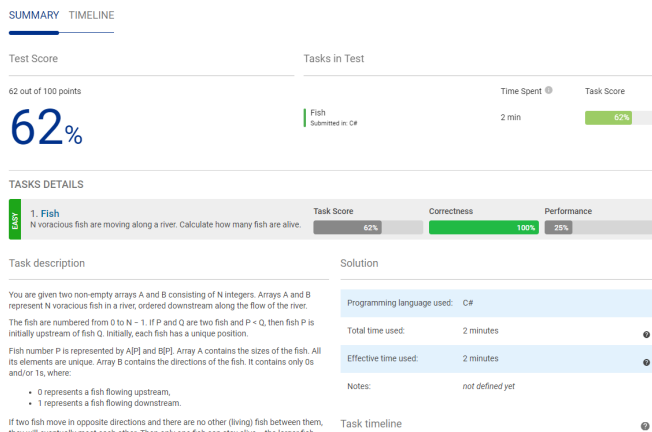


## LIFO (Last In First Out)

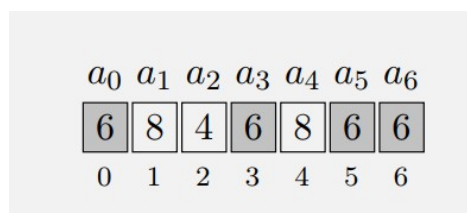
## Fish

I found solution with Stack but currently it is 62%

I should look for other solution. Tomorrow or maybe next time.



## Lesson 8: Leader



Basically saying we looking for element that occurs more then times. In given sequence A.

And there is basically two solutions:

$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$
4	6	6	6	6	8	8
0	1	2	3	4	5	6

1. Is simply sorting all elements and then choosing central one Because in order to be leader who should occur more then more then times

4	<del>6</del>	6	6	6	<del>8</del>	<del>8</del>
$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$
4	6	6	6	6	8	8

2. Smarter one using stack. We just add each element to stack and check if we have same element on the top of our stack the same we plan to currently add to the stack .

But in such cause it's not guaranty to be the leade it's only nown to be candidate for leader role.

So we need to check is value occurs more then time.

If yes then we have our leader else we need to return -1.

And after we recicle throw the array to check is our leader correct.

## Dominator

Test Name:

Summary Timeline

Test Score

100 out of 100 points

**100%**

Tasks in Test

Dominator  
Summed in C#

Time Spent 8 min

Task Score 100%

TASKS DETAILS

**1. Dominator**  
Find an index of an array such that its value occurs at more than half of indices in the array.

Task Score 100% Correctness 100% Performance 100%

Task description

An array A consisting of N integers is given. The dominator of array A is the value that occurs in more than half of the elements of A.

For example, consider array A such that

A[0] = 3 A[1] = 4 A[2] = 3  
A[3] = 2 A[4] = 3 A[5] = -1  
A[6] = 3 A[7] = 3

The dominator of A is 3 because it occurs in 5 out of 8 elements of A (namely in those with indices 0, 2, 4, 6 and 7) and 5 is more than a half of 8.

Write a function

```
class Solution { public int solution(int[] A); }
```

Solution

Programming language used: C#

Total time used: 8 minutes

Effective time used: 8 minutes

Notes: not defined yet

Task timeline

It was simply the copy of the explanation task but with one small difference. It returns us the index instead of number itself .

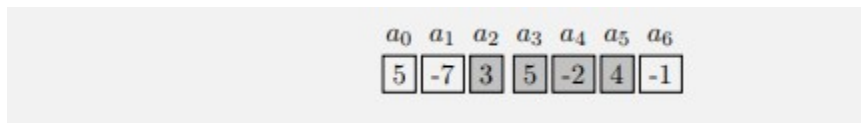
Very easy task. Just perfect to get back

## EquiLeader

I believe I found solution for the Equi Leader it is sort of obvious we just use 2 stacks instead of one.

The screenshot shows a coding platform interface with a dark theme. At the top, there are tabs for 'Summary' and 'Timeline'. Below this, the 'Test Score' section displays '44 out of 100 points' and a large '44%' in yellow. To the right, the 'Tasks in Test' section shows 'EquiLeader' submitted in 'C#' with a 'Time Spent' of '5 min' and a 'Task Score' of '44%'. Below this, the 'TASKS DETAILS' section shows '1. EquiLeader' with a description: 'Find the index S such that the leaders of the sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N - 1] are the same.' To the right of the description are three progress bars for 'Task Score' (44%), 'Correctness' (60%), and 'Performance' (25%). At the bottom, there are tabs for 'Task description' and 'Solution'.

## Lesson 9: Max Slice



So for given sequency we have covered in grey the maximum sum.

So In word there is sort of way of inserting next b

```
9.4: Maximal slice —  $O(n)$ .  
1 def golden_max_slice(A):  
2     max_ending = max_slice = 0  
3     for a in A:  
4         max_ending = max(0, max_ending + a)  
5         max_slice = max(max_slice, max_ending)  
6     return max_slice
```

Okay so it's sort of interesting I get max slic by using this simplet implementation in the world.

But it's still workable.

Okay so I run out of time but the concept is with me.

## Lesson 10: Prime and composite number

Prime number – the one that can be only divided by itself and by one

Composite number – has more than two divisors.

2	3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	---	----	----	----	----	----

White – prime numbers

Gray – composite numbers.

### Finding the number of divisors of n

This turns out to be an application of the multiplication principle for counting things.

For example, suppose we want to count (or find all of) the divisors of  $n = 144$ .

Begin by forming the prime factorization of 144:

$$144 = 2^4 \cdot 3^2.$$

So any divisor of 144 must be a product of some number of 2's (between 0 and 4) and some number of 3's (between 0 and 2). So here's a table of the possibilities:

	$2^0$	$2^1$	$2^2$	$2^3$	$2^4$
$3^0$	1	2	4	8	16
$3^1$	3	6	12	24	48
$3^2$	9	18	36	72	144

From the table, it's easy to see that there are  $5 \times 3 = 15$  divisors of 144.

In general, if you have the prime factorization of the number  $n$ , then to calculate how many divisors it has, you take all the exponents in the factorization, add 1 to each, and then multiply these "exponents + 1"s together.

## Algorithm

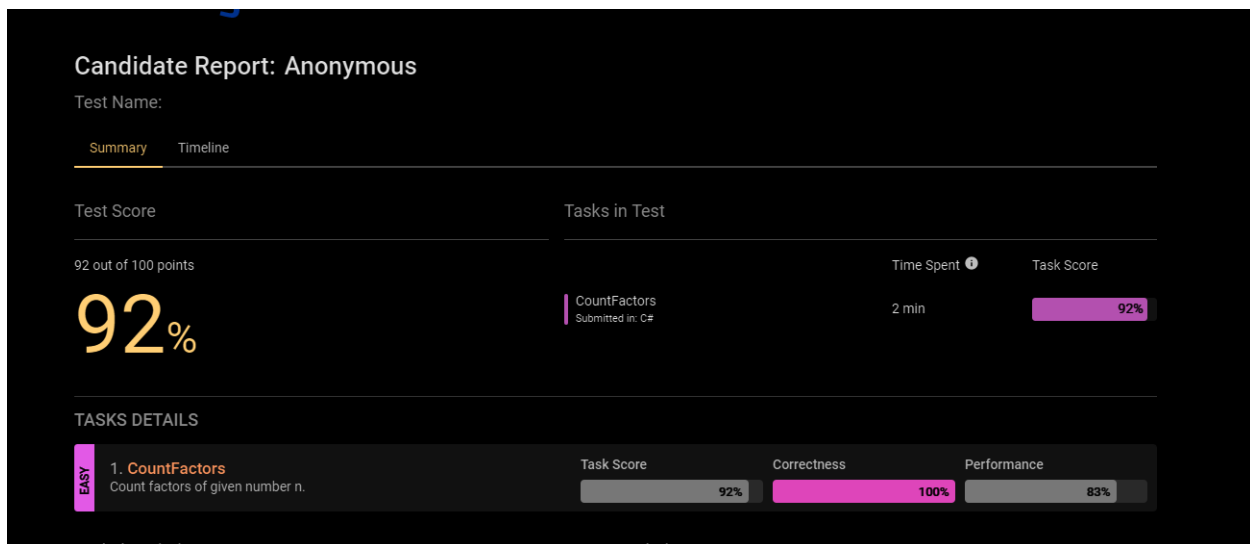
```
5 references | 0 changes | 0 authors, 0 changes
public static int Solution(int number)
{
    int result = 0;
    int i = 1;
    while (i*i<number)
    {
        if (number % i == 0)
            result += 2;
        i++;
    }

    if (i * i == number)
        result++;

    return result;
}
```

To find all prime numbers till specific number we do this shit. I mean it's strange but still

## Count factors



It was just rewriting the algorithm given in the book. And it's sort of sad that there should be other way of doing that.

## MinPeremeterRectangle

```
6 references | 0 changes | 0 authors, 0 changes
public static int Solution(int N)
{
    int i = (int) Math.Sqrt(N);
    while (i > 0)
    {
        if (N % i == 0)
        {
            int b = N / i;
            return (b + i) * 2;
        }
        i--;
    }

    return -1;
}
```

By the given formula I know that at least one of two multipliers should be less then  $\sqrt{N}$

And therefore I can start my search there

Candidate Report: Anonymous

Test Name:

Summary Timeline

Test Score

100 out of 100 points

100%

Tasks in Test

Task Name	Submitted in	Time Spent	Task Score
MinPerimeterRectangle	C#	1 min	100%

TASKS DETAILS

Task Name	Task Score	Correctness	Performance
1. MinPerimeterRectangle Find the minimal perimeter of any rectangle whose area equals N.	100%	100%	100%

Task description

An integer  $N$  is given, representing the area of some rectangle.  
The area of a rectangle whose sides are of length  $A$  and  $B$  is  $A * B$ , and the perimeter is  $2 * (A + B)$ .  
The goal is to find the minimal perimeter of any rectangle whose area equals  $N$ . The sides of this rectangle should be only integers.  
For example, given integer  $N = 30$ , rectangles of area 30 are:

- (1, 30), with a perimeter of 62,
- (2, 15), with a perimeter of 34,
- (3, 10), with a perimeter of 26,
- (5, 6), with a perimeter of 22.

Write a function:

Solution

Programming language used: C#

Total time used: 1 minutes

Effective time used: 1 minutes

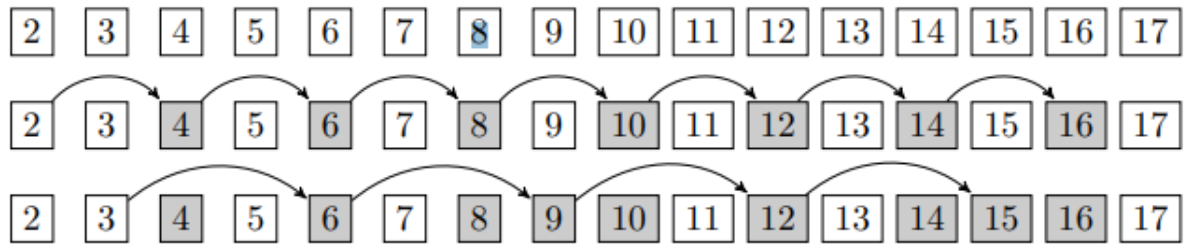
Notes: not defined yet

Task timeline

By that solution I have the best performance garnty in many cases it will be almost linier.

Sieve of Eratosthenes

## Lesson 11: Sieve of Eratosthenes



### Lesson 11.1: Fractolization