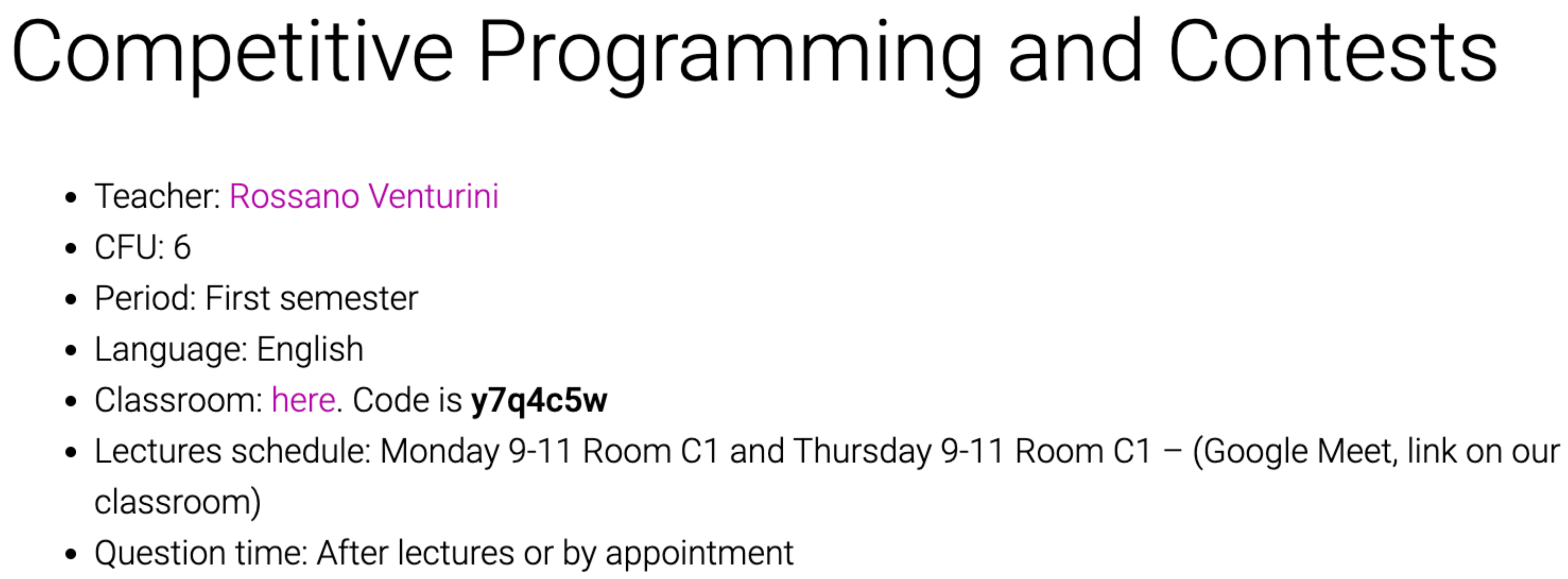
Competitive Programming and Contests

Rossano Venturini
rossano.venturini@unipi.it



Competitive Programming and Contests

- Teacher: Rossano Venturini
- CFU: 6
- Period: First semester
- Language: English
- Classroom: here. Code is **y7q4c5w**
- Lectures schedule: Monday 9-11 Room C1 and Thursday 9-11 Room C1 – (Google Meet, link on our classroom)
- Question time: After lectures or by appointment

https://github.com/rossanoventurini/CompetitiveProgramming

Google classroom: y7q4c5w

Please send me an email with your name and curriculum (or degree course)

Google™

Y!™

twitter

LinkedIn

facebook®

ebay®

YouTube

flickr™

amazon.com®

Microsoft®

Bloomberg

How we hire

We're looking for our next Noogler - someone who's good for the role, good for Google and good at lots of things.

Things move quickly around here. At Internet speed. That means we have to be nimble, both in how we work and how we hire. We look for people who are great at lots of things, love big challenges and welcome big changes. We can't have too many specialists in just one particular area. We're looking for people who are good for Google—and not just for right now, but for the long term.

This is the core of how we hire. Our process is pretty basic; the path to getting hired usually involves a first conversation with a recruiter, a phone interview and an onsite interview at one of our offices. But there are a few things we've baked in along the way that make getting hired at Google a little different.

How we interview

We're looking for smart, team-oriented people who can get things done. When you interview at Google, you'll likely interview with four or five Googlers. They're looking for four things:

Leadership

We'll want to know how you've flexed different muscles in different situations in order to mobilize a team. This might be by asserting a leadership role at work or with an organization, or by helping a team succeed when you weren't officially appointed as the leader.

Role-Related Knowledge

We're looking for people who have a variety of strengths and passions, not just isolated skill sets. We also want to make sure that you have the experience and the background that will set you up for success in your role. For engineering candidates in particular, we'll be looking to check out your coding skills and technical areas of expertise.

How You Think

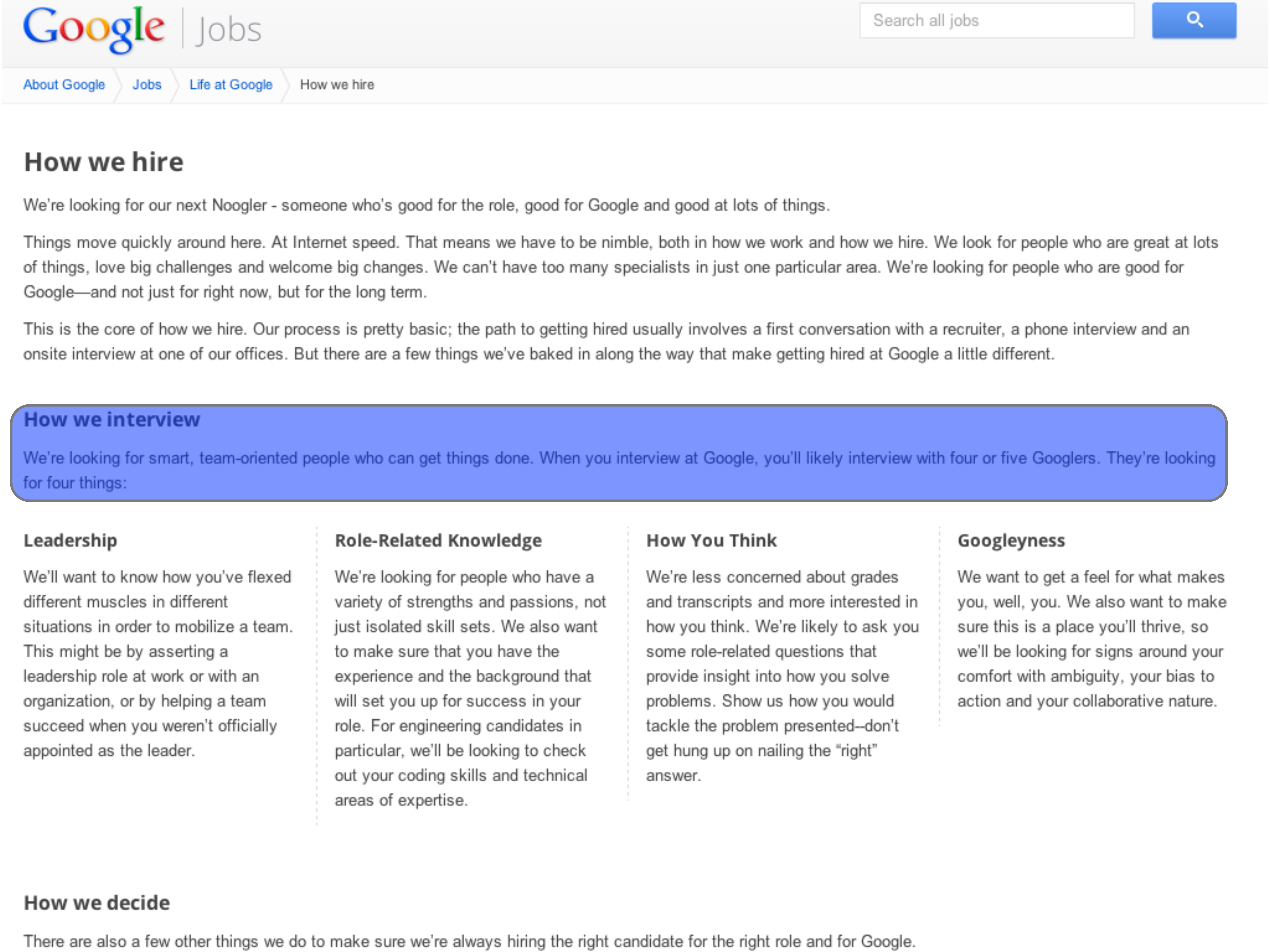
We're less concerned about grades and transcripts and more interested in how you think. We're likely to ask you some role-related questions that provide insight into how you solve problems. Show us how you would tackle the problem presented--don't get hung up on nailing the "right" answer.

Googleyness

We want to get a feel for what makes you, well, you. We also want to make sure this is a place you'll thrive, so we'll be looking for signs around your comfort with ambiguity, your bias to action and your collaborative nature.

How we decide

There are also a few other things we do to make sure we're always hiring the right candidate for the right role and for Google.



How we hire

We're looking for our next Noogler - someone who's good for the role, good for Google and good at lots of things.

Things move quickly around here. At Internet speed. That means we have to be nimble, both in how we work and how we hire. We look for people who are great at lots of things, love big challenges and welcome big changes. We can't have too many specialists in just one particular area. We're looking for people who are good for Google—and not just for right now, but for the long term.

This is the core of how we hire. Our process is pretty basic; the path to getting hired usually involves a first conversation with a recruiter, a phone interview and an onsite interview at one of our offices. But there are a few things we've baked in along the way that make getting hired at Google a little different.

How we interview

We're looking for smart, team-oriented people who can get things done. When you interview at Google, you'll likely interview with four or five Googlers. They're looking for four things:

Leadership

We'll want to know how you've flexed different muscles in different situations in order to mobilize a team. This might be by asserting a leadership role at work or with an organization, or by helping a team succeed when you weren't officially appointed as the leader.

Role-Related Knowledge

We're looking for people who have a variety of strengths and passions, not just isolated skill sets. We also want to make sure that you have the experience and the background that will set you up for success in your role. For engineering candidates in particular, we'll be looking to check out your coding skills and technical areas of expertise.

How You Think

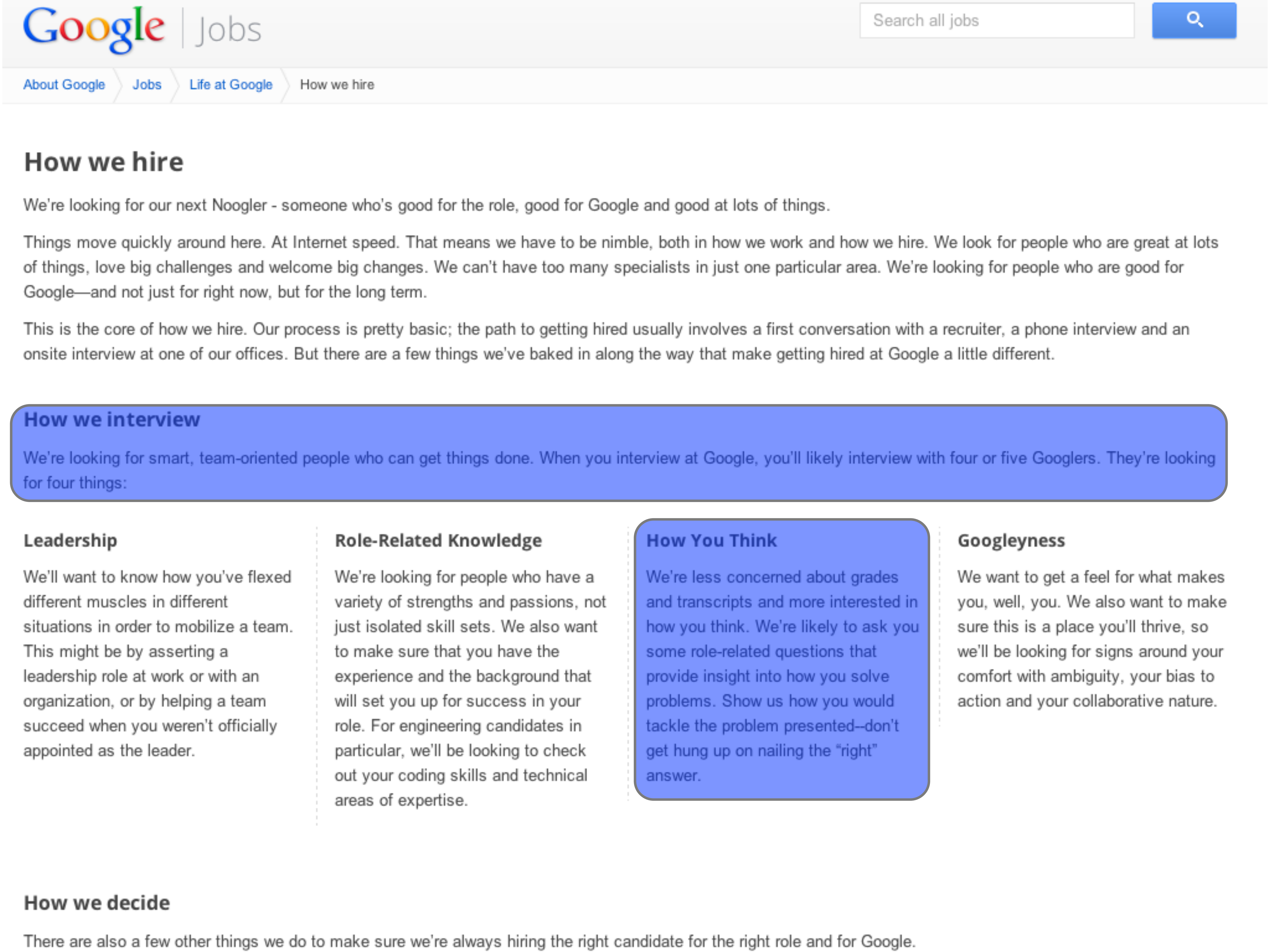
We're less concerned about grades and transcripts and more interested in how you think. We're likely to ask you some role-related questions that provide insight into how you solve problems. Show us how you would tackle the problem presented--don't get hung up on nailing the "right" answer.

Googleyness

We want to get a feel for what makes you, well, you. We also want to make sure this is a place you'll thrive, so we'll be looking for signs around your comfort with ambiguity, your bias to action and your collaborative nature.

How we decide

There are also a few other things we do to make sure we're always hiring the right candidate for the right role and for Google.



How we hire

We're looking for our next Noogler - someone who's good for the role, good for Google and good at lots of things.

Things move quickly around here. At Internet speed. That means we have to be nimble, both in how we work and how we hire. We look for people who are great at lots of things, love big challenges and welcome big changes. We can't have too many specialists in just one particular area. We're looking for people who are good for Google—and not just for right now, but for the long term.

This is the core of how we hire. Our process is pretty basic; the path to getting hired usually involves a first conversation with a recruiter, a phone interview and an onsite interview at one of our offices. But there are a few things we've baked in along the way that make getting hired at Google a little different.

How we interview

We're looking for smart, team-oriented people who can get things done. When you interview at Google, you'll likely interview with four or five Googlers. They're looking for four things:

Leadership

We'll want to know how you've flexed different muscles in different situations in order to mobilize a team. This might be by asserting a leadership role at work or with an organization, or by helping a team succeed when you weren't officially appointed as the leader.

Role-Related Knowledge

We're looking for people who have a variety of strengths and passions, not just isolated skill sets. We also want to make sure that you have the experience and the background that will set you up for success in your role. For engineering candidates in particular, we'll be looking to check out your coding skills and technical areas of expertise.

How You Think

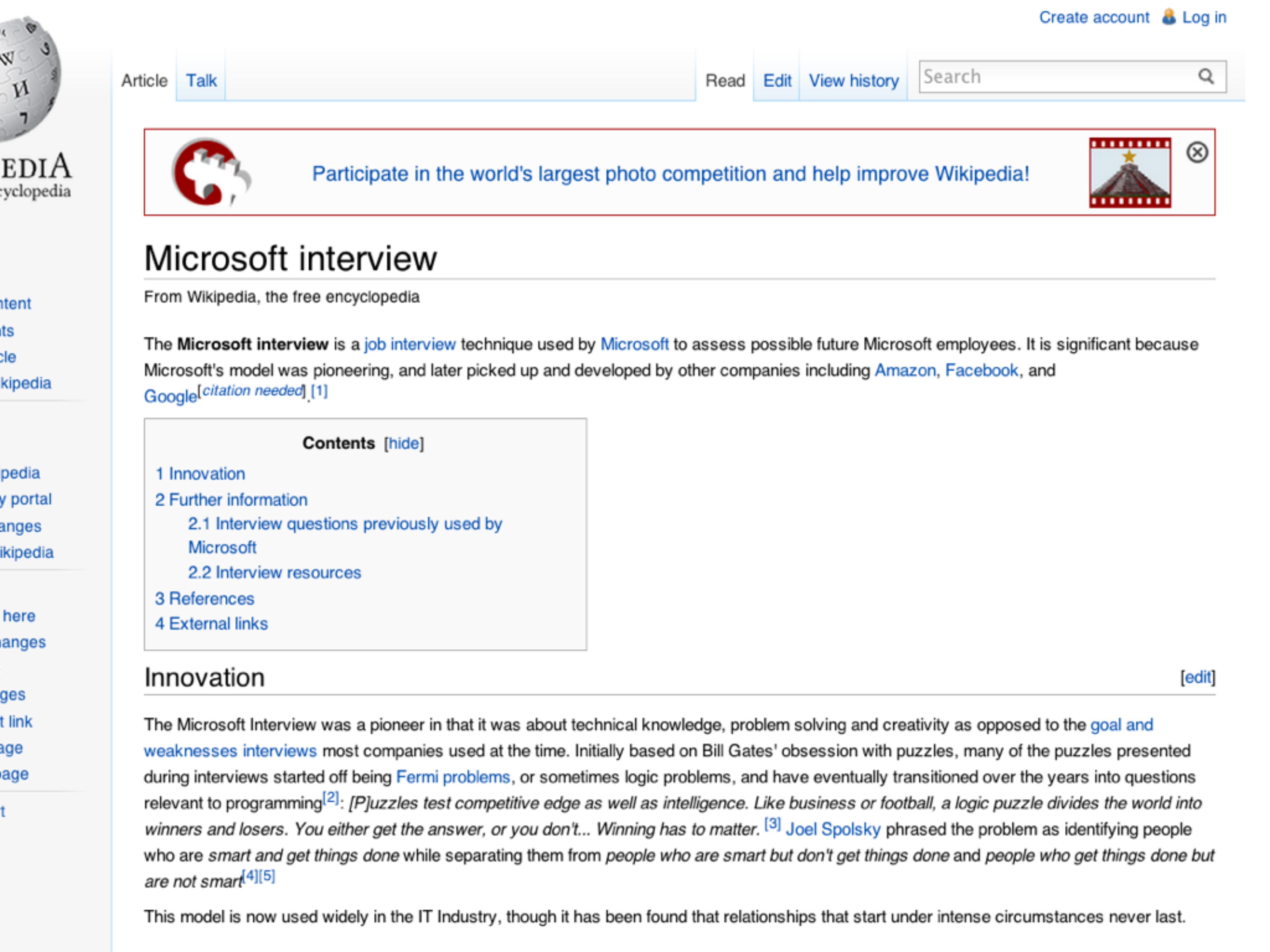
We're less concerned about grades and transcripts and more interested in how you think. We're likely to ask you some role-related questions that provide insight into how you solve problems. Show us how you would tackle the problem presented--don't get hung up on nailing the "right" answer.

Googleyness

We want to get a feel for what makes you, well, you. We also want to make sure this is a place you'll thrive, so we'll be looking for signs around your comfort with ambiguity, your bias to action and your collaborative nature.

How we decide

There are also a few other things we do to make sure we're always hiring the right candidate for the right role and for Google.



EDIA
cyclopedia

ntent
ts
cle
kipedia

pedia
y portal
anges
ikipedia

here
anges

ges
t link
age
age

t

Create account Log in

Article Talk Read Edit View history Search

Participate in the world's largest photo competition and help improve Wikipedia!

Microsoft interview

From Wikipedia, the free encyclopedia

The **Microsoft interview** is a job interview technique used by Microsoft to assess possible future Microsoft employees. It is significant because Microsoft's model was pioneering, and later picked up and developed by other companies including Amazon, Facebook, and Google[*citation needed*].[1]

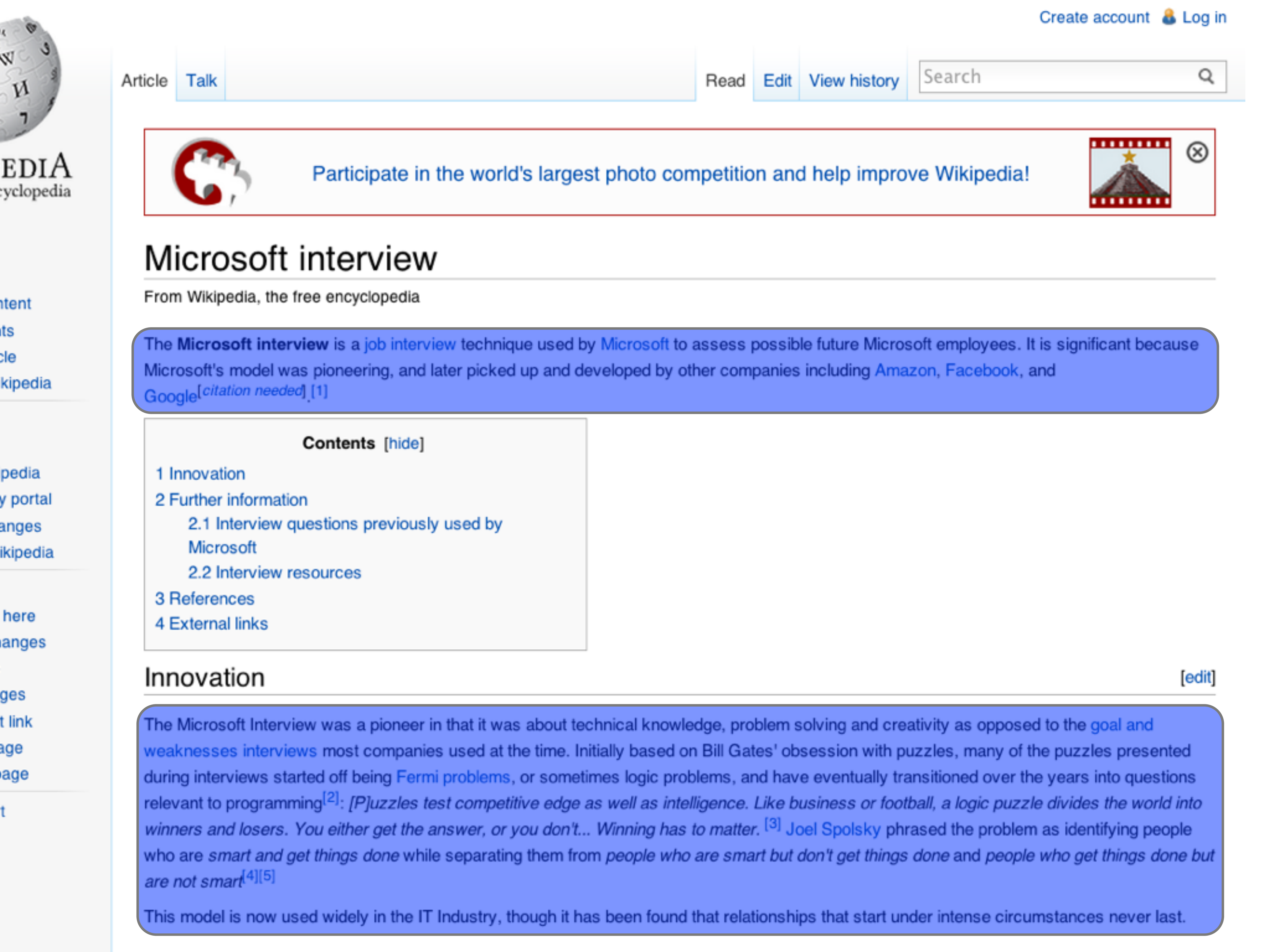
Contents [hide]

Innovation

[edit]

The Microsoft Interview was a pioneer in that it was about technical knowledge, problem solving and creativity as opposed to the goal and weaknesses interviews most companies used at the time. Initially based on Bill Gates' obsession with puzzles, many of the puzzles presented during interviews started off being Fermi problems, or sometimes logic problems, and have eventually transitioned over the years into questions relevant to programming[2]: *[P]uzzles test competitive edge as well as intelligence. Like business or football, a logic puzzle divides the world into winners and losers. You either get the answer, or you don't... Winning has to matter.* [3] Joel Spolsky phrased the problem as identifying people who are *smart and get things done* while separating them from *people who are smart but don't get things done* and *people who get things done but are not smart*[4][5]

This model is now used widely in the IT Industry, though it has been found that relationships that start under intense circumstances never last.



Participate in the world's largest photo competition and help improve Wikipedia!

Microsoft interview

From Wikipedia, the free encyclopedia

The **Microsoft interview** is a job interview technique used by Microsoft to assess possible future Microsoft employees. It is significant because Microsoft's model was pioneering, and later picked up and developed by other companies including Amazon, Facebook, and Google[*citation needed*].[1]

Contents [hide]

Innovation

[edit]

The Microsoft Interview was a pioneer in that it was about technical knowledge, problem solving and creativity as opposed to the goal and weaknesses interviews most companies used at the time. Initially based on Bill Gates' obsession with puzzles, many of the puzzles presented during interviews started off being Fermi problems, or sometimes logic problems, and have eventually transitioned over the years into questions relevant to programming[2]: *[P]uzzles test competitive edge as well as intelligence. Like business or football, a logic puzzle divides the world into winners and losers. You either get the answer, or you don't... Winning has to matter.* [3] Joel Spolsky phrased the problem as identifying people who are *smart and get things done* while separating them from *people who are smart but don't get things done* and *people who get things done but are not smart*[4][5]

This model is now used widely in the IT Industry, though it has been found that relationships that start under intense circumstances never last.

Google™

Please review the following topics:
Big-O notations also known as "the run time characteristic of an algorithm". You may want to refresh hash tables, heaps, binary trees, linked lists, depth-first search, recursion. For more information on Algorithms you can visit:
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index

Coding: You should know at least one programming language really well, and it should preferably be C++ or Java. C# is OK too, since it's pretty similar to Java. You will be expected to write some code in at least some of your interviews. You will be expected to know a fair amount of detail about your favorite programming language.

Sorting: Know how to sort. Don't do bubble-sort. You should know the details of at least one n*log(n) sorting algorithm, preferably two (say, quick sort and merge sort). Merge sort can be highly useful in situations where quick sort is impractical, so take a look at it.

Google™

Hashtables: Arguably the single most important data structure known to mankind. You absolutely should know how they work. Be able to implement one using only arrays in your favorite language, in about the space of one interview.

Trees: Know about trees; basic tree construction, traversal and manipulation algorithms. Familiarize yourself with binary trees, n-ary trees, and trie-trees. Be familiar with at least one type of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree, and know how it's implemented. Understand tree traversal

Algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

Graphs: Graphs are really important at Google. There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros & cons. You should know the basic graph traversal algorithms: breadth-first search and depth-first search. Know their computational complexity, their tradeoffs, and how to implement them in real code. If you get a chance, try to study up on fancier algorithms, such as Dijkstra and A*.

Other Data Structures: You should study up on as many other data structures and algorithms as possible. You should especially know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem, and be able to recognize them when an interviewer asks you them in disguise. Find out whatNP-complete means.

Mathematics: Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because counting problems, probability problems, and other Discrete Math 101 situations surrounds us. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of combinatorics and probability. You should be familiar with n-choose-k problems and their ilk – the more the better.

Operating Systems: Know about processes, threads and concurrency issues. Know about locks and mutexes and semaphores and monitors and how they work. Knowabout deadlock and livelock and how to avoid them. Know what resources a processes needs, and a thread needs, and how context switching works, and how it's initiated by the operating system and underlying hardware. Know a little about scheduling. The world is rapidly moving towards multi-core, so know the fundamentals of "modern" concurrency constructs. For information on System

Leaders in an array

Show Topic Tags Payu

Write a program to print all the LEADERS in the array. An element is leader if it is greater than all the elements to its right side. The rightmost element is always a leader.

Input:
The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows.
The first line of each test case contains a single integer N denoting the size of array.
The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of the array.

Output:
Print all the leaders.

Kadane's Algorithm

Show Topic Tags

Hike Flipkart FactSet D-E-Shaw Amazon Accolite
Morgan-Stanley Microsoft MetLife Housing.com
Snapdeal Samsung Payu Oracle Ola-Cabs
[24]7 Innovation Lab Zoho Walmart Visa Teradata

Given an array containing both negative and positive integers. Find the contiguous sub-array with maximum sum.

Input:
The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. The first line of each test case contains a single integer N denoting the size of array. The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of the array.

Output:
Print the maximum sum of the contiguous sub-array in a separate line for each test case.

Kadane's Algorithm

Show Topic Tags

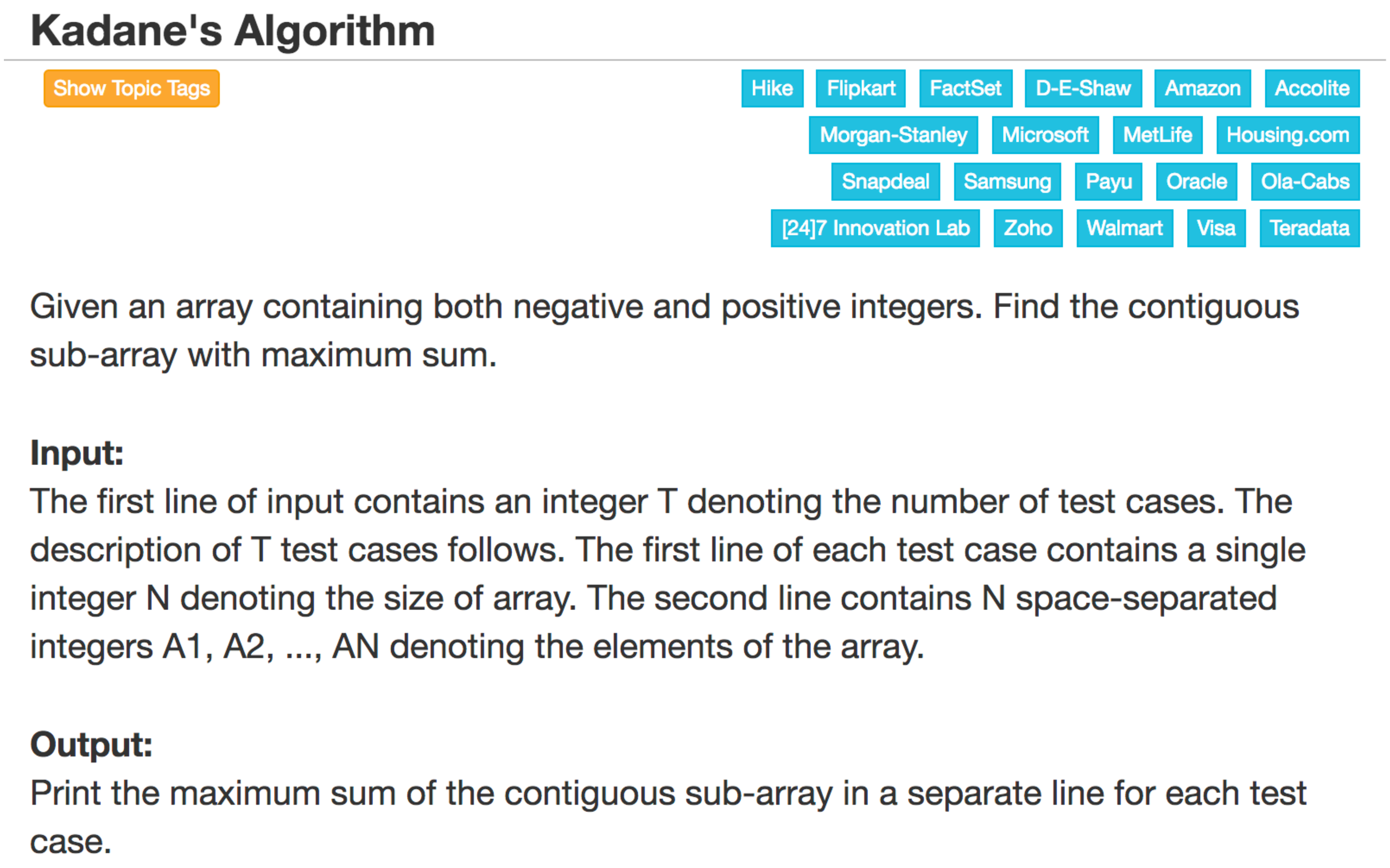
Hike Flipkart FactSet D-E-Shaw Amazon Accolite
Morgan-Stanley Microsoft MetLife Housing.com
Snapdeal Samsung Payu Oracle Ola-Cabs
[24]7 Innovation Lab Zoho Walmart Visa Teradata

Given an array containing both negative and positive integers. Find the contiguous sub-array with maximum sum.

Input:
The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. The first line of each test case contains a single integer N denoting the size of array. The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of the array.

Output:
Print the maximum sum of the contiguous sub-array in a separate line for each test case.

a	-1	5	8	-9	4	1

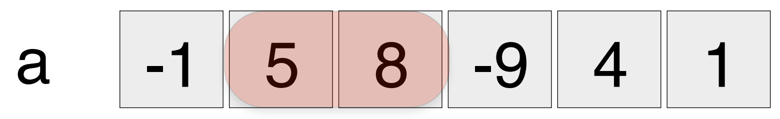


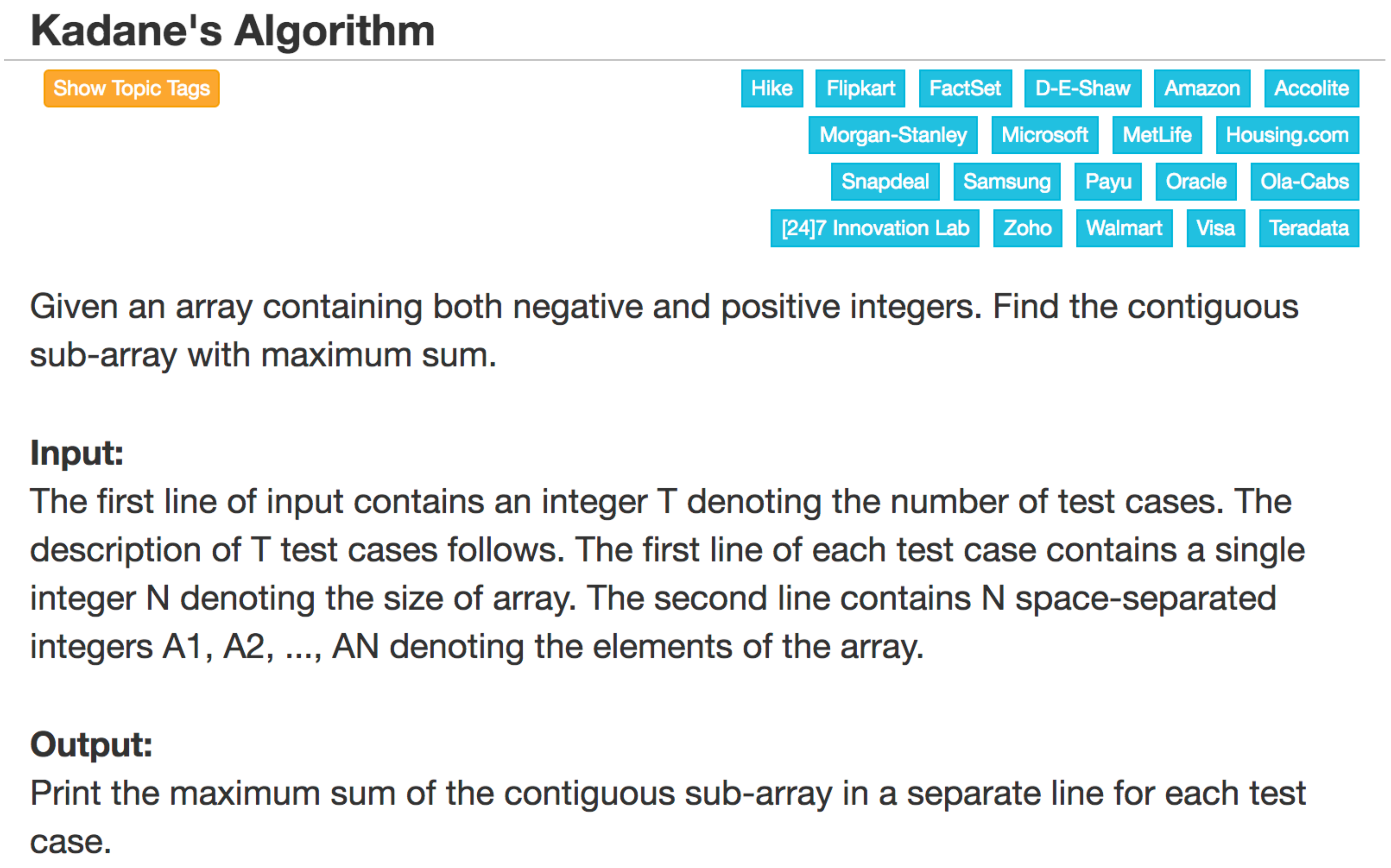
Kadane's Algorithm

Given an array containing both negative and positive integers. Find the contiguous sub-array with maximum sum.

Input:
The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. The first line of each test case contains a single integer N denoting the size of array. The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of the array.

Output:
Print the maximum sum of the contiguous sub-array in a separate line for each test case.



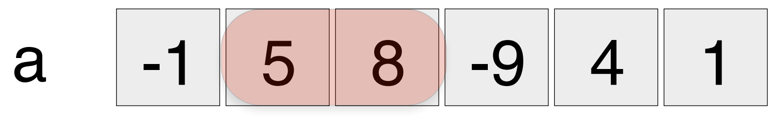


Kadane's Algorithm

Given an array containing both negative and positive integers. Find the contiguous sub-array with maximum sum.

Input:
The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. The first line of each test case contains a single integer N denoting the size of array. The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of the array.

Output:
Print the maximum sum of the contiguous sub-array in a separate line for each test case.



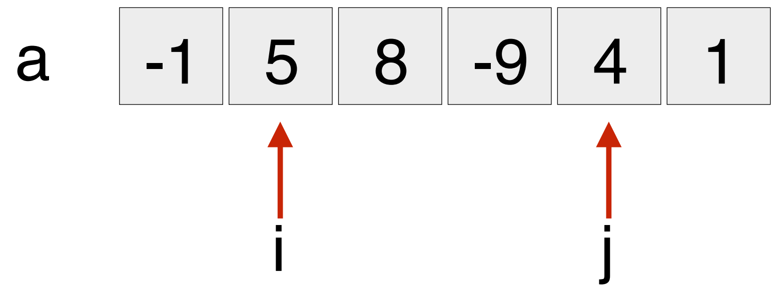
output: 13

Solution 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        sum=0;
        for(k=i; k<=j; k++)
        {
            sum+=a[k];
        }
        if(sum > max) max=sum;
    }
}
```

Solution 1

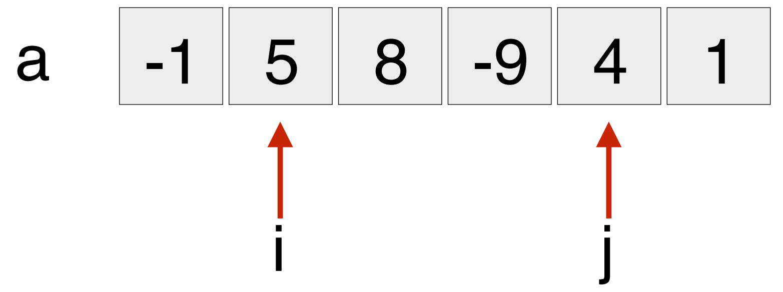
```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        sum=0;
        for(k=i; k<=j; k++)
        {
            sum+=a[k];
        }
        if(sum > max) max=sum;
    }
}
```



Solution 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        sum=0;
        for(k=i; k<=j; k++)
        {
            sum+=a[k];
        }
        if(sum > max) max=sum;
    }
}
```

O(1)

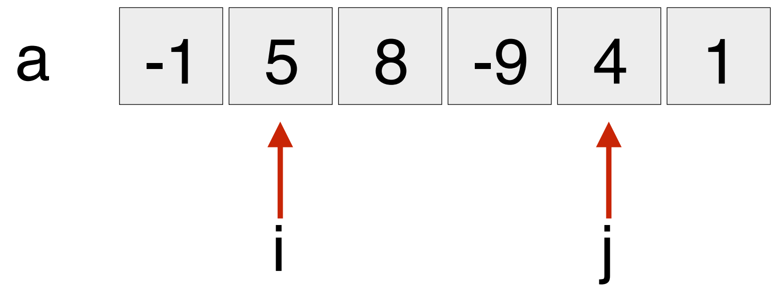


Solution 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        sum=0;
        for(k=i; k<=j; k++)
        {
            sum+=a[k];
        }
        if(sum > max) max=sum;
    }
}
```

O(1)

O(n)



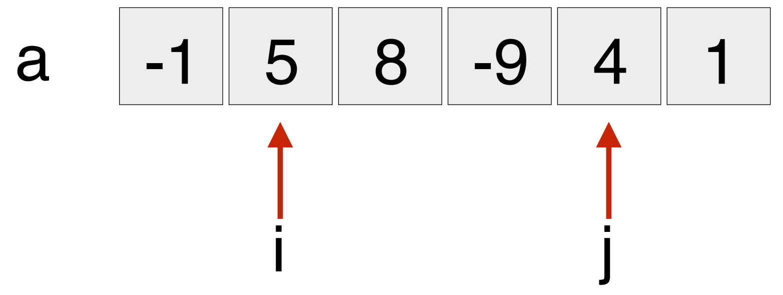
Solution 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        sum=0;
        for(k=i; k<=j; k++)
        {
            sum+=a[k];
        }
        if(sum > max) max=sum;
    }
}
```

O(1)

O(n)

$O(n^2)$



Solution 1

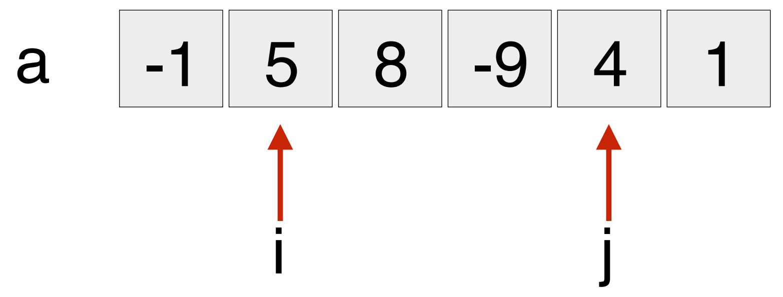
```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        sum=0;
        for(k=i; k<=j; k++)
        {
            sum+=a[k];
        }
        if(sum > max) max=sum;
    }
}
```

$O(1)$

$O(n)$

$O(n^2)$

$O(n^3)$



Solution 1

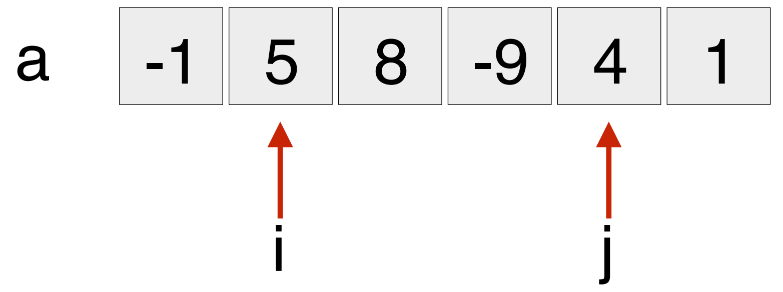
```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        sum=0;
        for(k=i; k<=j; k++)
        {
            sum+=a[k];
        }
        if(sum > max) max=sum;
    }
}
```

O(1)

O(n)

O(n^2)

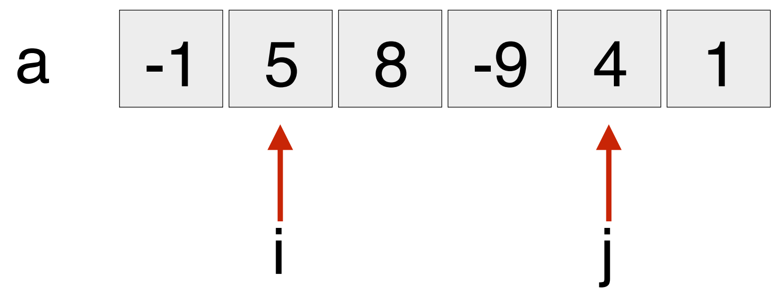
O(n^3)



Complexity: O(n^3) :-(

Solution 2

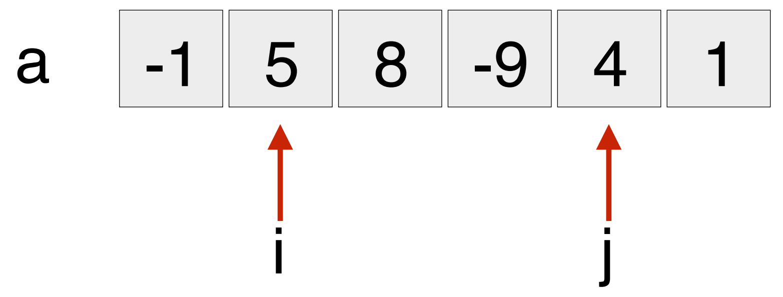
```
max = 0;

for(i=0; i<n; i++)
{
    sum=0;
    for(j=i; j<n; j++)
    {
        sum+=a[j];
        if(sum > max)
            max=sum;
    }
}
```



Solution 2

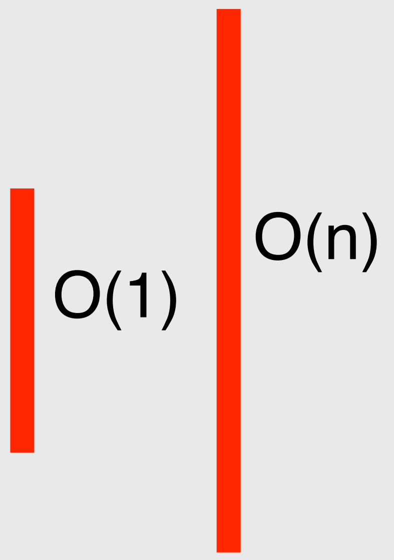
```
max = 0;

for(i=0; i<n; i++)
{
    sum=0;
    for(j=i; j<n; j++)
    {
        sum+=a[j];
        if(sum > max)
            max=sum;
    }
}
```

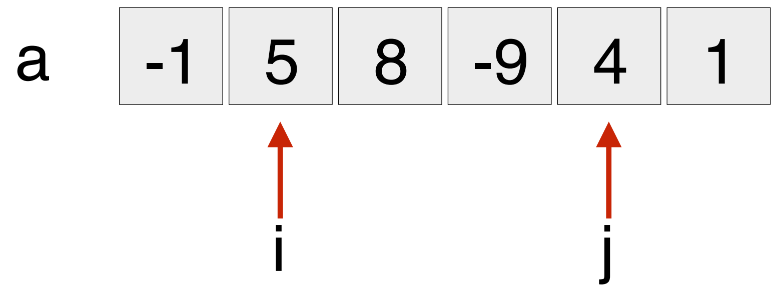
O(1)



Solution 2

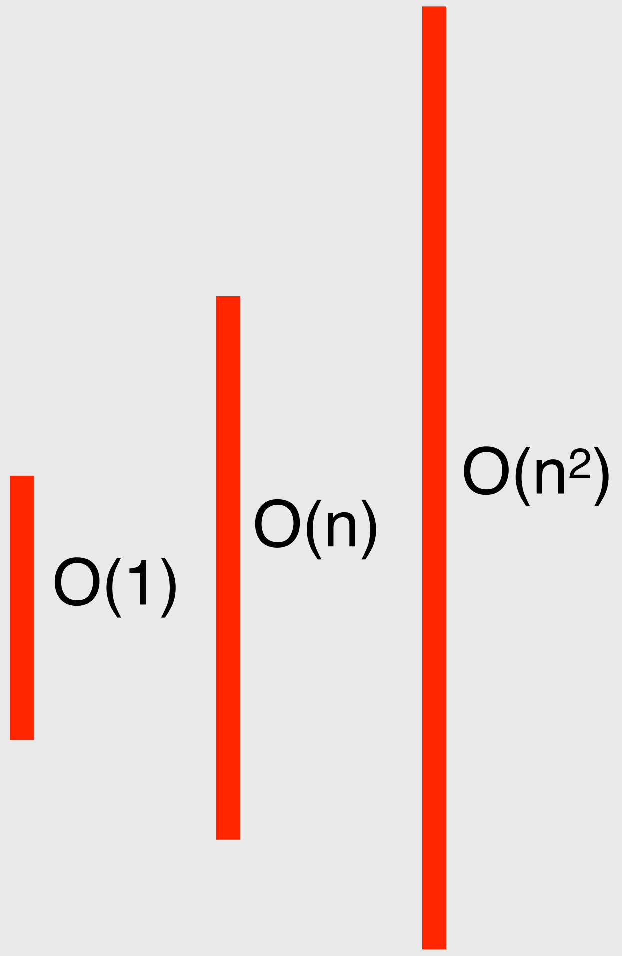
```
max = 0;

for(i=0; i<n; i++)
{
     sum=0;
     for(j=i; j<n; j++)
     {
          sum+=a[j];
          if(sum > max)
                    max=sum;
     }
}
```

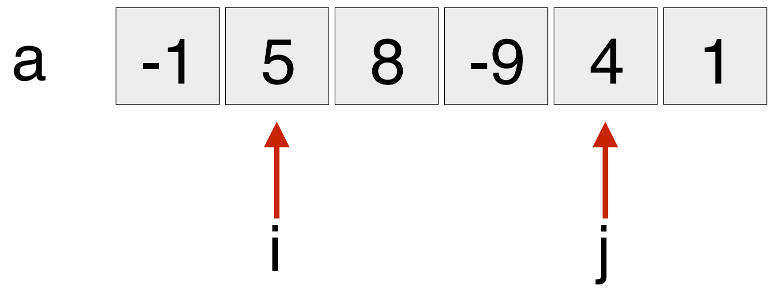




Solution 2

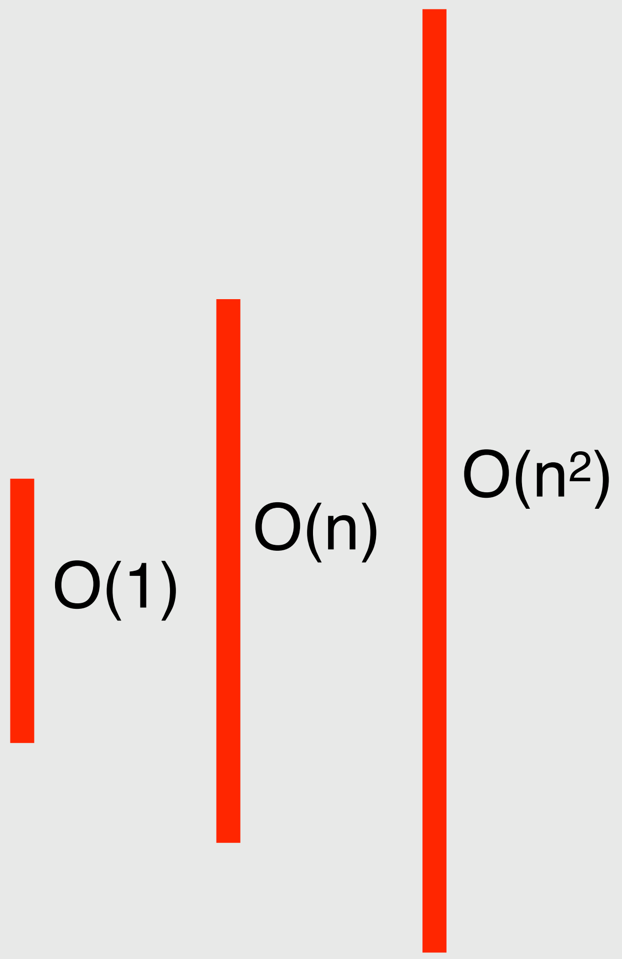
```
max = 0;

for(i=0; i<n; i++)
{
    sum=0;
    for(j=i; j<n; j++)
    {
        sum+=a[j];
        if(sum > max)
                max=sum;
    }
}
```

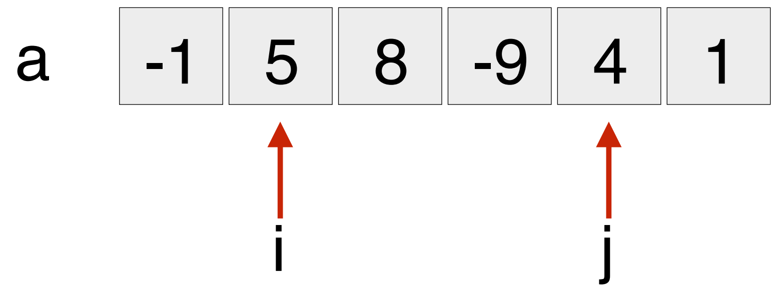




Solution 2

```
max = 0;

for(i=0; i<n; i++)
{
    sum=0;
    for(j=i; j<n; j++)
    {
        sum+=a[j];
        if(sum > max)
            max=sum;
    }
}
```





Complexity: $O(n^2)$:-|

Any better solution?

We can exploit two properties of the subarray with maximum sum

Any better solution?

We can exploit two properties of the subarray with maximum sum

1) Sum of values in any prefix of the optimal sub-array is positive. By contradiction, remove a negative prefix to get sub-array with a larger sum.

Any better solution?

We can exploit two properties of the subarray with maximum sum

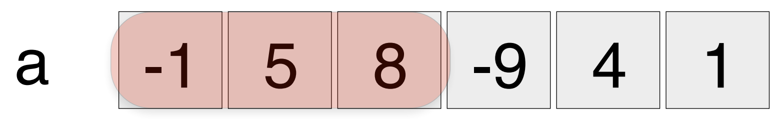
1) Sum of values in any prefix of the optimal sub-array is positive. By contradiction, remove a negative prefix to get sub-array with a larger sum.

a	-1	5	8	-9	4	1

Any better solution?

We can exploit two properties of the subarray with maximum sum

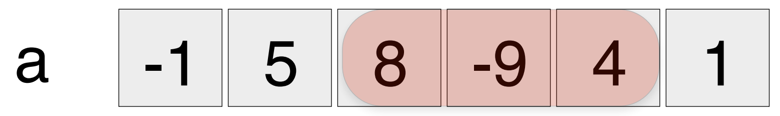
1) Sum of values in any prefix of the optimal sub-array is positive. By contradiction, remove a negative prefix to get sub-array with a larger sum.



Any better solution?

We can exploit two properties of the subarray with maximum sum

1) Sum of values in any prefix of the optimal sub-array is positive. By contradiction, remove a negative prefix to get sub-array with a larger sum.

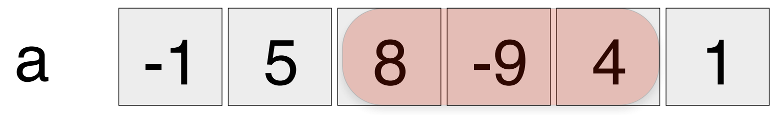


Any better solution?

We can exploit two properties of the subarray with maximum sum

1) Sum of values in any prefix of the optimal sub-array is positive. By contradiction, remove a negative prefix to get sub-array with a larger sum.

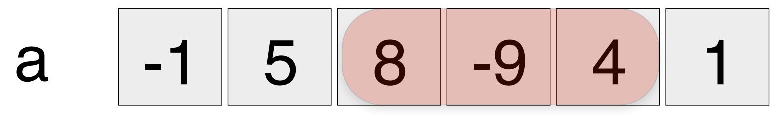
2) The value that precede the first value of the optimal sub-array is negative. By contradiction, include this positive value to get sub-array with a larger sum.



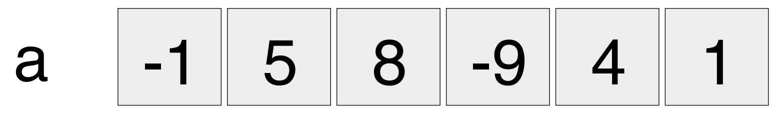
Any better solution?

We can exploit two properties of the subarray with maximum sum

1) Sum of values in any prefix of the optimal sub-array is positive. By contradiction, remove a negative prefix to get sub-array with a larger sum.



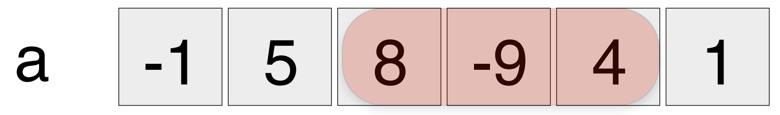
2) The value that precede the first value of the optimal sub-array is negative. By contradiction, include this positive value to get sub-array with a larger sum.



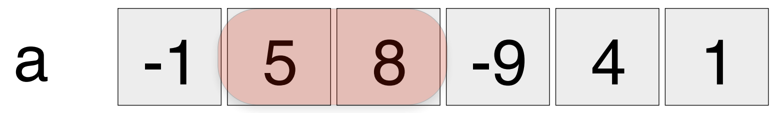
Any better solution?

We can exploit two properties of the subarray with maximum sum

1) Sum of values in any prefix of the optimal sub-array is positive. By contradiction, remove a negative prefix to get sub-array with a larger sum.



2) The value that precede the first value of the optimal sub-array is negative. By contradiction, include this positive value to get sub-array with a larger sum.



Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```

O(1)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```

O(n)

O(1)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```

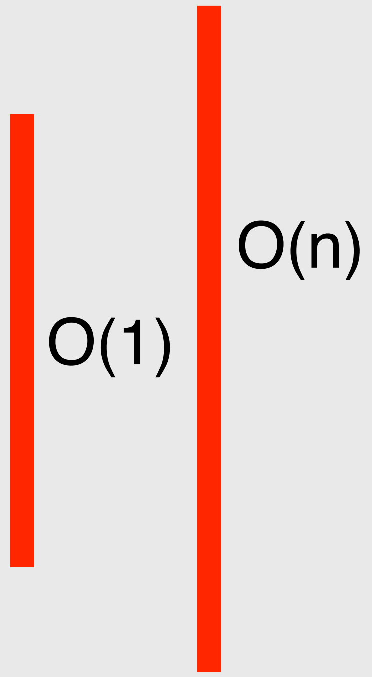
O(1)

O(n)

Complexity: O(n) :-)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```



Sum

-1

-1	5	8	-9	4	1

Complexity: O(n) :-)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```

O(1)

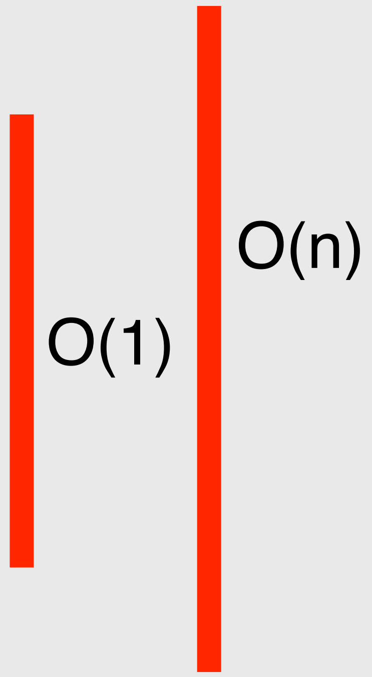
O(n)

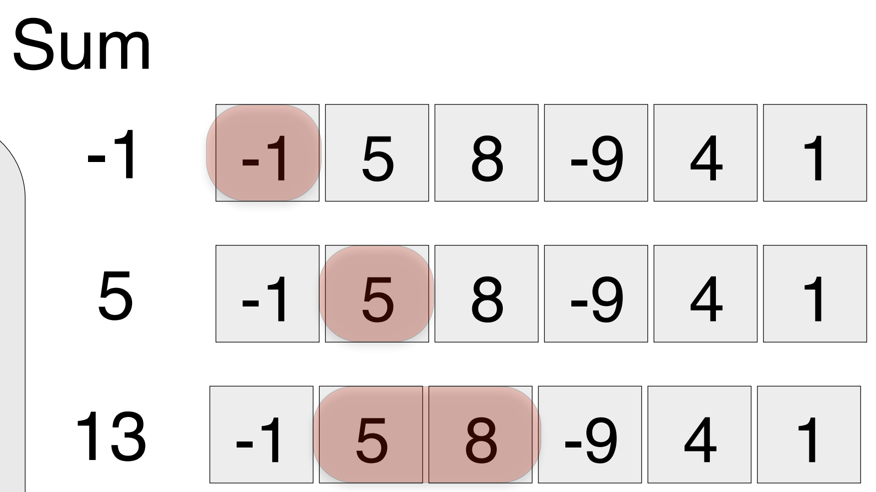
Sum						
-1	-1	5	8	-9	4	1
5	-1	5	8	-9	4	1

Complexity: O(n) :-)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```





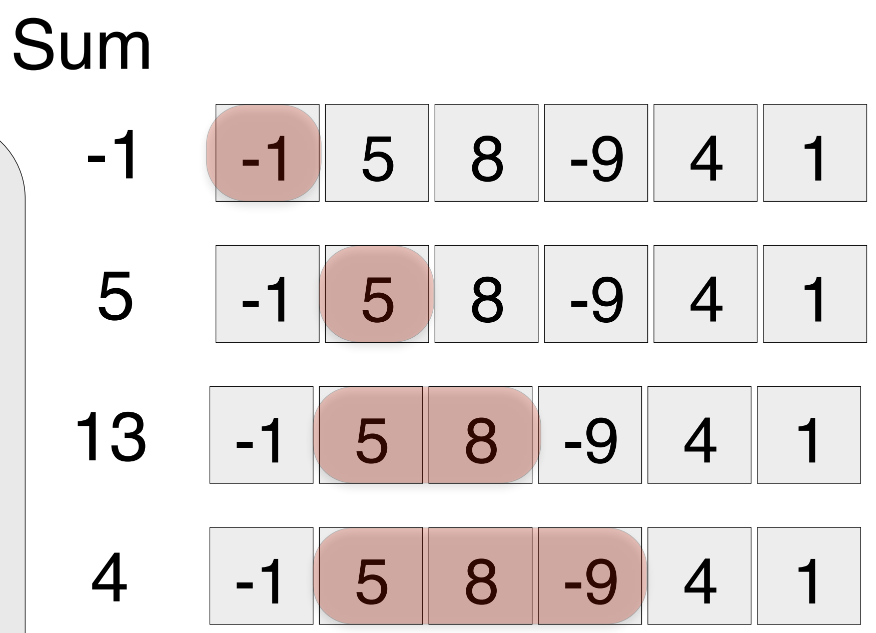
Complexity: O(n) :-)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```

O(1)

O(n)



Complexity: O(n) :-)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```

O(1)

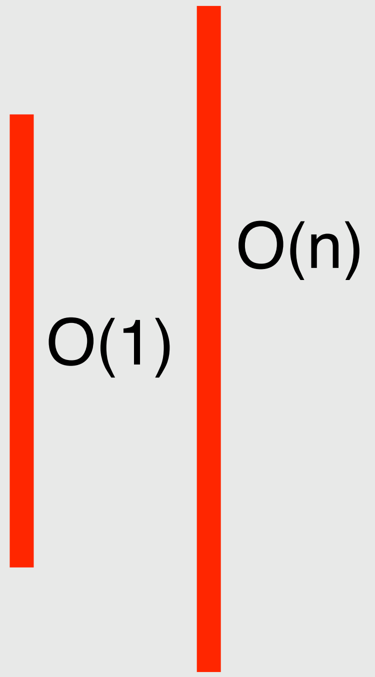
O(n)

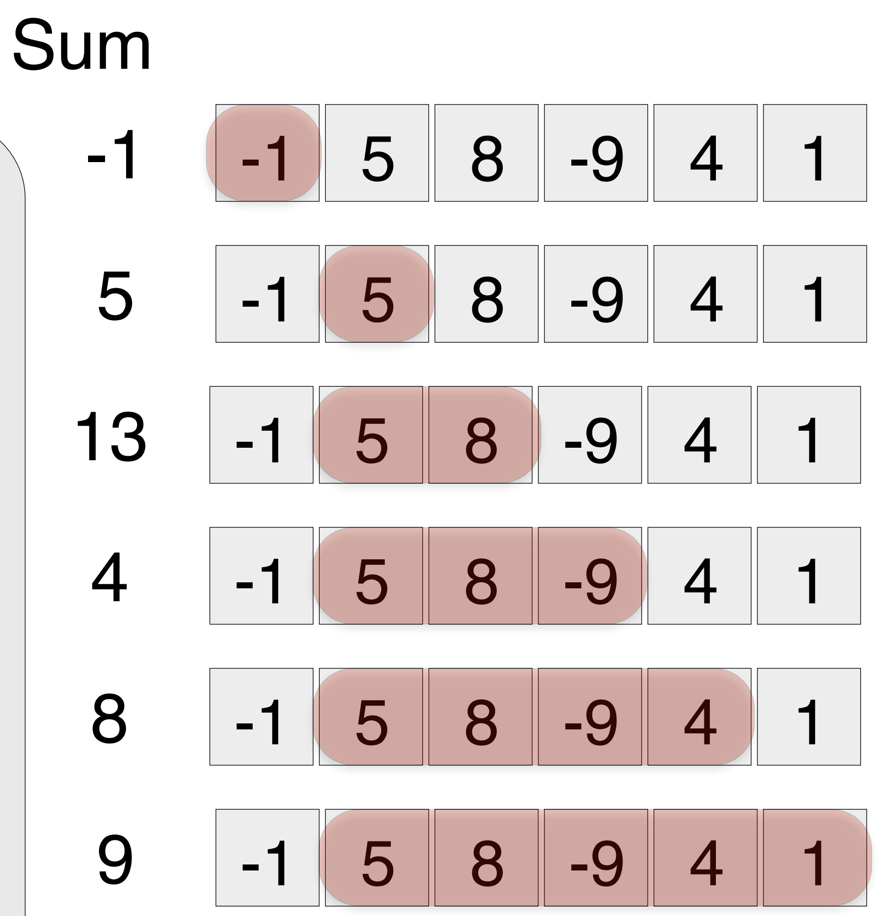
Sum						
-1	**-1**	5	8	-9	4	1
5	-1	**5**	8	-9	4	1
13	-1	**5**	**8**	-9	4	1
4	-1	**5**	**8**	**-9**	4	1
8	-1	**5**	**8**	**-9**	**4**	1

Complexity: O(n) :-)

Solution 3

```
max = 0; sum = 0;

for(i=0; i<n; i++)
{
   if(sum > 0) sum+=a[i];
   else sum=a[i];

   if(sum > max) max=sum;
}
```





Complexity: O(n) :-)

Missing number in array

Show Topic Tags

Payu Ola-Cabs Microsoft Intuit Cisco Amazon

Given an array of size n-1 and given that there are numbers from 1 to n with one missing, the missing number is to be found.

Input:

The first line of input contains an integer T denoting the number of test cases.
The first line of each test case is N,size of array.
The second line of each test case contains N-1 input C[i],numbers in array.

Output:

Print the missing number in array.

Goals and opportunities

The goal of the course is to improve programming and problem solving skills of the students by facing them with difficult problems and by presenting the techniques that help their reasoning in the implementation of correct and efficient solutions. The importance of these skills has been recognized by the most important software companies worldwide, which evaluate candidates in their job interviews mostly by the ability in addressing such difficult problems (e.g., see here).

A natural goal is to involve the students in the intellectual pleasure of programming and problem solving, also preparing them for the most important international online contests, such as Topcoder, Codeforces, HackerRank, CodeChef, Facebook Hacker Cup, Google Code Jam and so on, for internships in most important companies and their interviews. A desirable side-effect of the course could be to organize and prepare teams of students for online contests.

The course will provide the opportunity of

- facing with challenging algorithmic problems;
- improving problem solving and programming skills;
- getting in touch with some big companies for internships, scholarships, or thesis proposals.

Background

If you wish to refresh your mind on basic Algorithms and Data Structures, I suggest you to look at the well-known book Introduction to Algorithms, 3rd Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

I strongly suggest you to watch the following video lectures as soon as possible.

- Insertion Sort and Merge Sort
- Heaps and Heap Sort
- Counting Sort, Radix Sort, Lower Bounds for Sorting
- Binary Search Trees
- AVL trees
- Hashing with Chaining

References

- Introduction to Algorithms, 3rd Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2009 (Amazon) [CCLR]
- Algorithms, 4th Edition, Robert Sedgewick, Kevin Wayne, Addison-Wesley Professional, 2011 (Amazon) [RS]
- Algorithms, Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani, McGraw-Hill, 2006. (Amazon) [DPZ]
- Programming Challenges: The Programming Contest Training Manual, Steven S. Skiena, Miguel A. Revilla, Springer, 2003 (Amazon) [SR]
- Competitive Programming 3: The New Lower Bound of Programming Contests, Steven Halim, Felix Halim, (here) [HH]
- Competitive Programmer's Handbook, Antti Laaksonen (here) [L]
- The C++ Programming Language, 4th Edition, Bjarne Stroustrup, Addison-Wesley Professional, 2013 (Amazon)
- The C++ Standard Library: A Tutorial and Reference (2nd Edition), Nicolai M. Josuttis, Addison-Wesley Professional, 2012 (Amazon)

Useful links

- Erik D Demaine, Srini Devadas, Nancy Ann Lynch, "MIT Design and Analysis of Algorithms", MIT Algorithm course which includes video lectures
- Tim Roughgarden, "Algorithm specialization", Coursera
- Visualgo: visualising data structures and algorithms through animation
- Geeks for Geeks: Company interviews preparation
- Interviews common coding questions
- How to: Work at Google — Example Coding/Engineering Interview Video
- Google's Code Jam
- Topcoder
- Codeforces
- Geeks for Geeks
- CodeChef - Data Structures and Algorithms links
- Geeks for Geeks: Top 10 Algorithms and Data Structures for Competitive Programming
- List of resources for competitive programming

Lectures

Date	Lecture	References	Problems
17/09/2018	Introduction	Slides	Leaders in array (solution), Kadane's algorithm (solution), Missing number in array (solution), Trapping rain water (solution), and Sliding window maximum (solution)
18/09/2017	Solutions of Trapping rain water and Sliding window maximum	Rossano's notes*	Next larger element (solution), Towers (solution), and Finding Team Member (solution)

Last year lectures

Date	Lecture	References	Problems
27/09/2017	Standard Template Library (STL)	Slides. Tutorial and STL algorithms	Megacity (solution), Find pair (solution), and Two heaps (solution)
29/09/2017	Standard Template Library (STL). Coding	Slides	
05/10/2017	Searching and Sorting: Binary Search, Merge Sort, QuickSort, Counting Sort, and Radix Sort.	Rossano's notes*. [CCLR] Chapters 2.3, 7, and 8. Binary search. Exponential search. Interpolation search (optional)	Inversion count (solution) and Largest Even Number (solution)
11/10/2017	Trees: representation, traversals, and Binary	Rossano's notes*. [CCLR] Chapters 10.4 and 12. Tree traversals. Euler	Firing employees (solution), Check for BST (solution), Preorder traversal and BST

How to solve a problem

Here’s a step-by-step guide on how to effectively solve a problem:

- **Read the Problem Description:** Begin by carefully reading the problem statement. Ensure that you have a clear understanding of what the problem is asking you to solve.
- **Understand with Examples:** Examine any provided examples. Use these examples to validate your understanding of the problem’s requirements and constraints.
- **Design a Trivial Solution:** Start by designing a straightforward and naive solution to the problem. This is often referred to as a brute-force approach. It may not be the most efficient solution, but it helps you get started and ensures that you have a working solution.
- **Brainstorm for Efficiency:** Once you have a working solution, think about how you can optimize it. Consider different algorithms, data structures, and strategies that might lead to a more efficient solution. Use running examples to help you visualize and refine your approach.
- **Seek Help and Hints:** If you’re stuck and unable to find a more efficient solution, don’t hesitate to seek help. Discuss the problem with other students, ask questions in the group, or consult resources like your course materials, relevant websites, or textbooks. Gathering hints and insights from others is a valuable part of the learning process.
- **Document Your Solution:** Write a brief description of your chosen solution in English. Explain the logic behind it and provide an analysis of its time and space complexity. This documentation helps you clarify your thought process and is valuable for future reference.
- **Implement Your Solution:** Write the code for your solution in Rust. Ensure that your code follows the algorithm you’ve designed and documented.
- **Test and Debug:** Submit your implementation to the problem’s platform or testing environment. Work on debugging and refining your code until it passes all the tests and produces correct results.
- **Compare Solutions:** Always compare your solution and implementation with existing ones if available. This can help you learn different approaches and best practices for solving similar problems.

Exam

Exam

- **Homeworks (No longer mandatory)**
    - I strongly recommend you to solve all the problems in the Web page
    - Use a github repository (public or private) to collect your solutions
    - **Remember**: we are here to learn and improve our skills. Just a motivation for not cheating.

Exam

- **Homeworks (No longer mandatory)**
  - I strongly recommend you to solve all the problems in the Web page
  - Use a github repository (public or private) to collect your solutions
  - **Remember**: we are here to learn and improve our skills. Just a motivation for not cheating.
- **1) Written/Oral exam (1)**
  - Well, you know, the professor asks, the student tries to answer…
    - Questions about the material of the course (easy part)
    - Solve some **new** problems. Impossible to do if you didn’t do the homeworks.

Exam

- **Homeworks (No longer mandatory)**
    - I strongly recommend you to solve all the problems in the Web page
    - Use a github repository (public or private) to collect your solutions
    - **Remember**: we are here to learn and improve our skills. Just a motivation for not cheating.
- **1) Written/Oral exam (1)**
    - Well, you know, the professor asks, the student tries to answer…
        - Questions about the material of the course (easy part)
        - Solve some **new** problems. Impossible to do if you didn’t do the homeworks.
- **2) Report/Oral exam**
    - ~20 pages report about a topic (data structures, technique, …)
        - Detailed description with a lot of examples and figures
        - Solutions to problems using those ideas

Exam

- **Homeworks (No longer mandatory)**
    - I strongly recommend you to solve all the problems in the Web page
    - Use a github repository (public or private) to collect your solutions
    - **Remember**: we are here to learn and improve our skills. Just a motivation for not cheating.
- **1) Written/Oral exam (1)**
    - Well, you know, the professor asks, the student tries to answer…
        - Questions about the material of the course (easy part)
        - Solve some **new** problems. Impossible to do if you didn’t do the homeworks.
- **2) Report/Oral exam**
    - ~20 pages report about a topic (data structures, technique, …)
        - Detailed description with a lot of examples and figures
        - Solutions to problems using those ideas
- You can choose either option 1 or 2

Exam

- **Homeworks (No longer mandatory)**
    - I strongly recommend you to solve all the problems in the Web page
    - Use a github repository (public or private) to collect your solutions
    - **Remember**: we are here to learn and improve our skills. Just a motivation for not cheating.
- **1) Written/Oral exam (1)**
    - Well, you know, the professor asks, the student tries to answer…
        - Questions about the material of the course (easy part)
        - Solve some **new** problems. Impossible to do if you didn't do the homeworks.
- **2) Report/Oral exam**
    - ~20 pages report about a topic (data structures, technique, …)
        - Detailed description with a lot of examples and figures
        - Solutions to problems using those ideas
- You can choose either option 1 or 2
- **Bonus points (0 — +∞ points)**
    - Active participation to our class: 1 – 5 points for outstanding contributors
    - Well-prepared attempt to partecipate to online contests: +1 point for each high level contest
    - Successful interview with a big company: +3 points

Exam

- ~~**Homeworks (No longer mandatory)**~~
    - ~~I strongly recommend you to solve all the problems in the Web page~~
    - ~~Use a github repository (public or private) to collect your solutions~~
    - ~~**Remember**: we are here to learn and improve our skills. Just a motivation for not cheating.~~
- ~~**1) Written/Oral exam (1)**~~
    - ~~Well, you know, the professor asks, the student tries to answer…~~
        - ~~Questions about the material of the course (easy part)~~
        - ~~Solve some **new** problems. Impossible to do if you didn't do the homeworks.~~
- ~~**2) Report/Oral exam**~~
    - ~~~20 pages report about a topic (data structures, technique, …)~~
        - ~~Detailed description with a lot of examples and figures~~
        - ~~Solutions to problems using those ideas~~
- ~~You can choose either option 1 or 2~~
- ~~**Bonus points (0 — +∞ points)**~~
    - ~~Active participation to our class: 1 – 5 points for outstanding contributors~~
    - ~~Well-prepared attempt to partecipate to online contests: +1 point for each high level contest~~
    - ~~Successful interview with a big company: +3 points~~

Doc.mod:	ROSSANO	VENTURINI
Cod.mod:	59547_50143	COMPETITIVE PROGRAMMING AND CONTESTS

2017/18

BS02	Giudizio complessivo sull'insegnamento.	3,8		🟧

2018/19

BS02	Giudizio complessivo sull'insegnamento.	3,5		🟧

2019/20

BS2	Giudizio complessivo sull'insegnamento.	3,3	*10*	🟧

2020/21

BS2	Giudizio complessivo sull'insegnamento.	3,3	*10*	🟧

2021/22

BS2	Giudizio complessivo sull'insegnamento.	3,6	*9*	🟧

2022/23

BS2	Giudizio complessivo sull'insegnamento.	3,6	*14*	🟧

Exam

Exam

- **Homeworks**
  - Solve problems on our Web page. Not Mandatory.
  - Use a github repository (public or private) to collect your solutions

Exam

- **Homeworks**
    - Solve problems on our Web page. Not Mandatory.
    - Use a github repository (public or private) to collect your solutions
- **Hands-on**
    - **One or two problems to solve**
    - **You have to send me your solutions in two weeks!**
    - We will have 3 hands-ons

Exam

- **Homeworks**
  - Solve problems on our Web page. Not Mandatory.
  - Use a github repository (public or private) to collect your solutions
- **Hands-on**
  - **One or two problems to solve**
  - **You have to send me your solutions in two weeks!**
  - We will have 3 hands-ons
- Two options:
  - submit hands-on during the class with deadlines
  - submit hands-on + homework

Exam

- **Homeworks**
  - Solve problems on our Web page. Not Mandatory.
  - Use a github repository (public or private) to collect your solutions
- **Hands-on**
  - **One or two problems to solve**
  - **You have to send me your solutions in two weeks!**
  - We will have 3 hands-ons
- Two options:
  - submit hands-on during the class with deadlines
  - submit hands-on + homework
- **Oral exam**
  - Well, you know, the professor asks questions about the material of the course, the student tries to answer…

Exam

- **Homeworks**
    - Solve problems on our Web page. Not Mandatory.
    - Use a github repository (public or private) to collect your solutions
- **Hands-on**
    - **One or two problems to solve**
    - **You have to send me your solutions in two weeks!**
    - We will have 3 hands-ons
- Two options:
    - submit hands-on during the class with deadlines
    - submit hands-on + homework
- **Oral exam**
    - Well, you know, the professor asks questions about the material of the course, the student tries to answer…

Exam

- **Homeworks**
  - Solve problems on our Web page. Not Mandatory.
  - Use a github repository (public or private) to collect your solutions
- **Hands-on**
  - **One or two problems to solve**
  - **You have to send me your solutions in two weeks!**
  - We will have 3 hands-ons
- Two options:
  - submit hands-on during the class with deadlines
  - submit hands-on + homework
- **Oral exam**
  - Well, you know, the professor asks questions about the material of the course, the student tries to answer…

- **Bonus points (0 — +∞ points)**
  - Active participation to our class: 1 – 5 points for outstanding contributors
  - Well-prepared attempt to participate to online contests: +1 point for each high level contest
  - Successful interview with a big company: +3 points

Exam

- **Homeworks**
  - Solve problems on our Web page. Not Mandatory.
  - Use a github repository (public or private) to collect your solutions
- **Hands-on**
  - **One or two problems to solve**
  - **You have to send me your solutions in two weeks!**
  - We will have 3 hands-ons
- Two options:
  - submit hands-on during the class with deadlines
  - submit hands-on + homework
- **Oral exam**

((Hands-on with deadlines) || (Homeworks + hands-on without deadlines)) && (Oral exam)

- **Bonus points (0 — +∞ points)**
  - Active participation to our class: 1 – 5 points for outstanding contributors
  - Well-prepared attempt to participate to online contests: +1 point for each high level contest
  - Successful interview with a big company: +3 points

Rust

Rust

- C++ was the programming language of the course

Rust

- C++ was the programming language of the course
- Rust

Rust

- C++ was the programming language of the course
- Rust
    - **Performance**
        - As fast as C/C++

Rust

- C++ was the programming language of the course
- Rust
  - **Performance**
    - As fast as C/C++
  - **Safety**
    - Designed to be memory/thread safe. “If it compiles, it works"

Rust

- C++ was the programming language of the course
- Rust
  - **Performance**
    - As fast as C/C++
  - **Safety**
    - Designed to be memory/thread safe. “If it compiles, it works"
  - **Tools**
    - A lot of modern tools (package manager, documentation, formatting, testing, benchmarking, …) make programming in Rust easy and funny

Rust

- C++ was the programming language of the course
- Rust
  - **Performance**
    - As fast as C/C++
  - **Safety**
    - Designed to be memory/thread safe. “If it compiles, it works"
  - **Tools**
    - A lot of modern tools (package manager, documentation, formatting, testing, benchmarking, …) make programming in Rust easy and funny
    - The compiler is your coach!

Rust

- C++ was the programming language of the course
- Rust
  - **Performance**
    - As fast as C/C++
  - **Safety**
    - Designed to be memory/thread safe. “If it compiles, it works"
  - **Tools**
    - A lot of modern tools (package manager, documentation, formatting, testing, benchmarking, …) make programming in Rust easy and funny
    - The compiler is your coach!
- Maybe not the best programming language for competitions (see e.g. this discussion)

Rust

- C++ was the programming language of the course
- Rust
  - **Performance**
    - As fast as C/C++

((Hands-on with deadlines) || (Homeworks + hands-on without deadlines)) && (Oral exam)

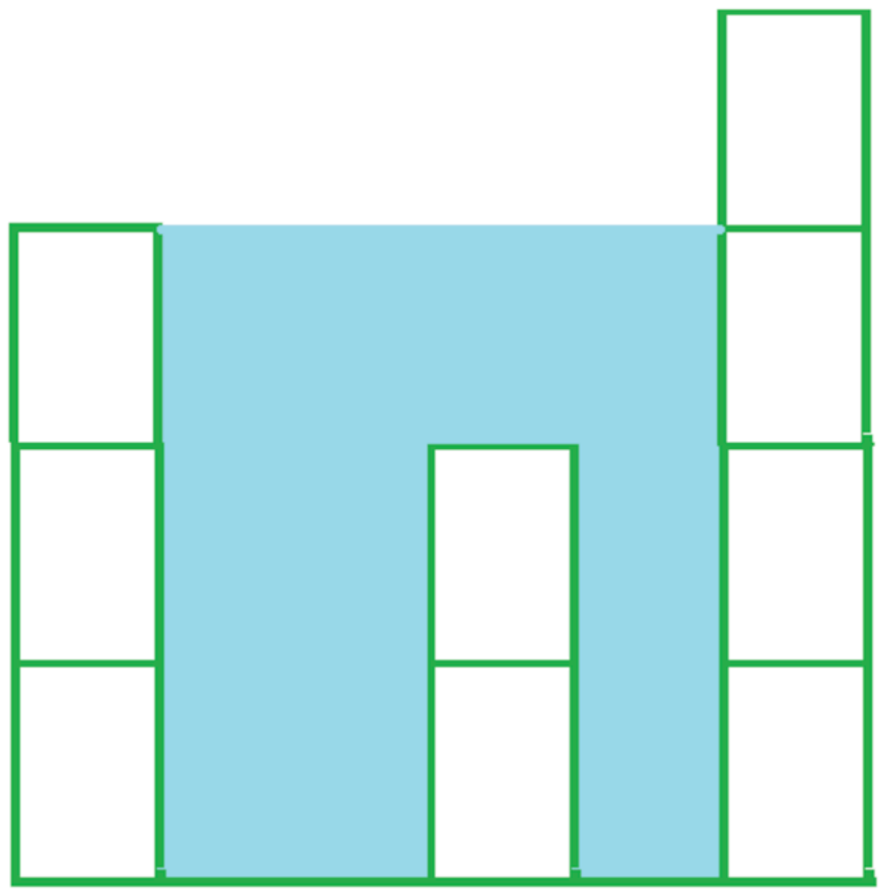
&& (Learn Rust)

  - **Tools**
    - A lot of modern tools (package manager, documentation, formatting, testing, benchmarking, …) make programming in Rust easy and funny
    - The compiler is your coach!
- Maybe not the best programming language for competitions (see e.g. this discussion)

Trapping Rain Water

Show Topic Tags   Payu   Microsoft   D-E-Shaw   Amazon   Accolite

Given n non-negative integers in array representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.



Bars for input {3, 0, 0, 2, 0, 4}
Total trapped water = 3 + 3 + 1 + 3 = 10

Maximum of all subarrays of size k

Show Topic Tags Flipkart Directi Amazon

Given an array and an integer k, find the maximum for each and every contiguous subarray of size k.

Input:
The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows.
The first line of each test case contains a single integer 'N' denoting the size of array and the size of subarray 'k'.
The second line contains 'N' space-separated integers A1, A2, ..., AN denoting the elements of the array.

Output:
Print the maximum for every subarray of size k.

Constraints:
$1 \leq T \leq 200$
$1 \leq N \leq 100$
$1 \leq k \leq N$
$0 \leq A[i] < 1000$