Algorithms and Data Structures (ECS529)

Nikos Tzevelekos

Lecture 1

Introduction

Organisation

Module Organiser: Nikos Tzevelekos

Lecturers: Matthew Huntbach (labs lead) and Nikos Tzevelekos (lectures)

Team of **Demonstrators**

Lectures and Exercises:

- Thursdays 13:00 – 14:00, weeks 1 – 12, Arts II
- Fridays 12:00 – 14:00, weeks 1 – 12, Great Hall

Labs *(check your assigned slot!)*:

- Thursdays 11:00 – 13:00, weeks 1 – 12, ITL ground floor
- Fridays 9:00 – 11:00, weeks 1 - 12, ITL middle floor

Communication:

ask questions during the lectures/labs

post questions – either by email or at the **Q&A forum** on QM+

More organisation

Assessment:

- 10 lab sheets (10%), 1 in-term test (10%)
- 1 **individual** mini project (10%)
- final exam (70%)

Module material (available on QM+ page each week):

- Lecture slides available each week
- Lecture recordings
- Exercises and lab sheets, with solutions!

Assessment

- Lab exercises are marked each week. For n in [1,2,3,4,5,6,7,9,10,11]:
  - lab sheet n is made available on QM+ on Friday of week n-1
  - it is marked in the lab of week n+1 at the latest
- In-term test will be on week 7
- Mini project:
  - to be submitted on week 11
  - made available on QM+ on week 8
- Exam questions are not going to be unexpected – they will be similar to lab and lecture exercises

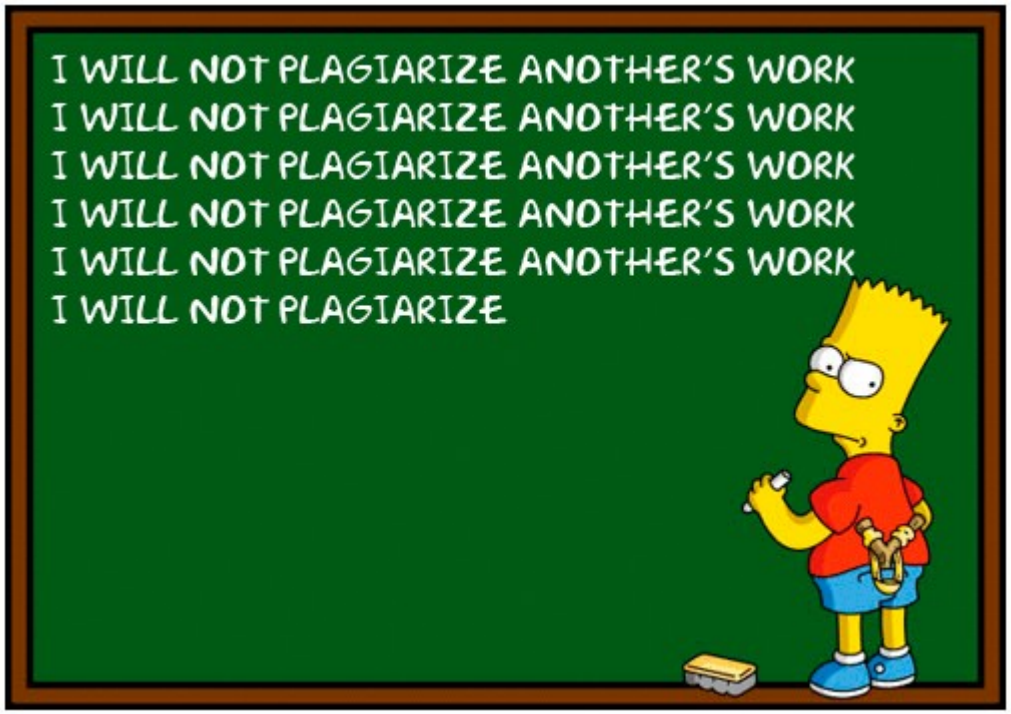
Where to get more help

All that is required for the module is in the lecture slides, exercises and lab sheets.

If you would like a textbook to supplement them with:

- *Introduction to Algorithms*. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
- *Problem Solving with Algorithms and Data Structures Using Python*. Bradley W. Miller, David L. Ranum. Freely available as an interactive textbook here.
- *Algorithms*. Robert Sedgewick, Kevin Wayne.

Where **not** to get help from:



and neither:

Math Call to 911

www.youtube.com/watch?v=FT4NOe4vtoM

Two pieces of advice

#1:

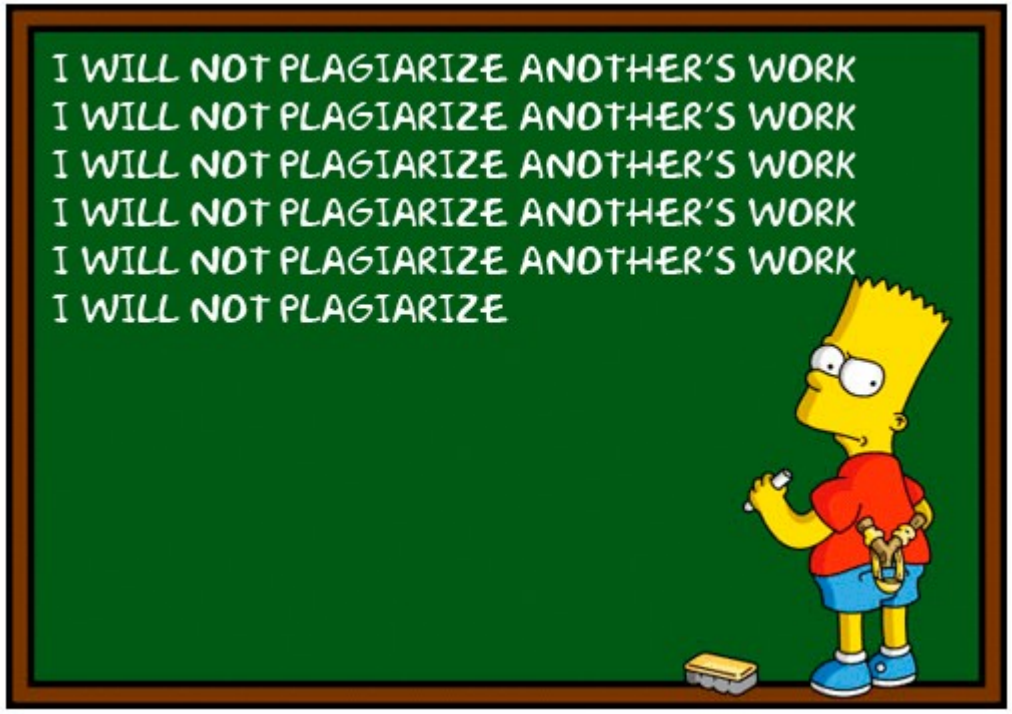
This is a **hard** module (e.g. clearly more difficult than ECS421)

So, try to **engage** as much as possible:

- come to lectures
- try the exercises in class
- work on lab sheets (in labs and at home)
- ask questions (in person, on QM+, by email)
- stay away from plagiarism

#2:

Repeat the second part of #1 as much as necessary



ADS Keywords: algorithm

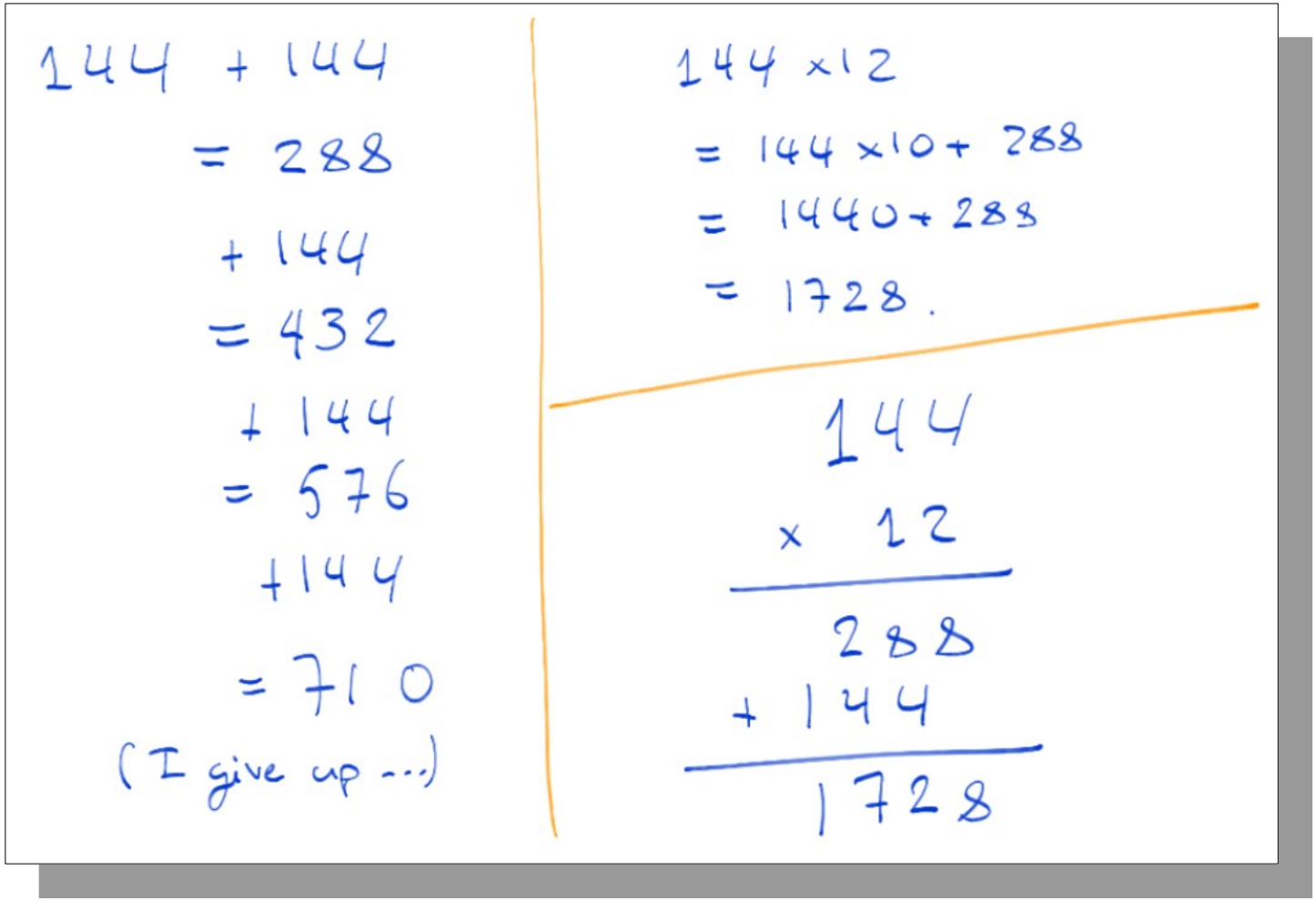
An algorithm is a description of a formal process, i.e. sequence of steps or rules, to solve a specific problem:

- we use algorithms in every-day life (e.g. routing, timetabling, cooking)
- number calculations use algorithms (e.g. *long multiplication*)
- web protocols use algorithms (e.g. authentication, routing)
- databases use algorithms (e.g. searching, inserting)
- image processing, artificial intelligence, cryptography, cryptocurrencies, cloud computing and more generally **every** App/program/website that does anything useful uses some (optimised!) algorithm for it

– OK, but why study them?

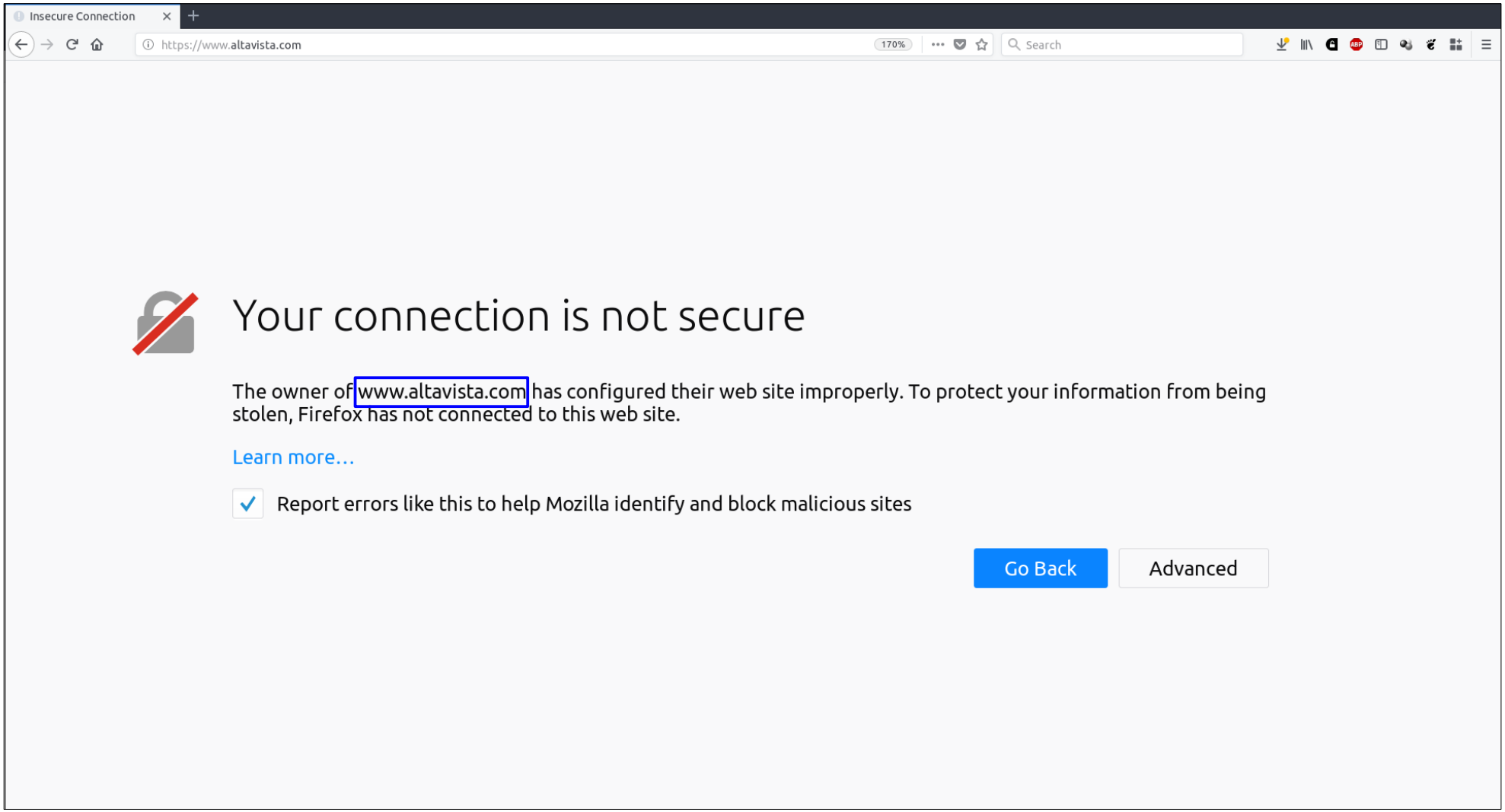
A good algorithm makes life easier

compute 144 x 12 without using a smartphone



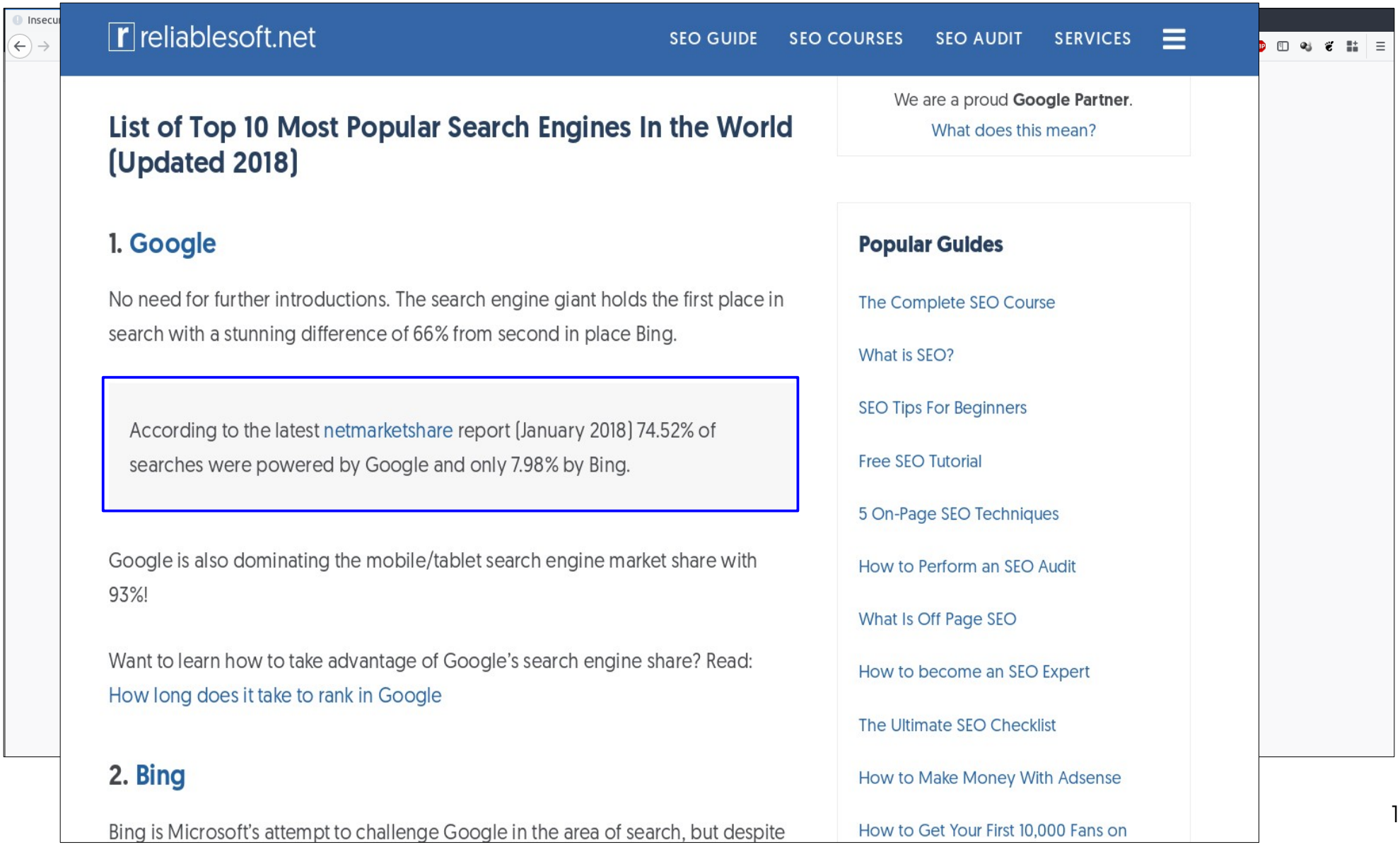
A good algorithm makes all the difference

imagine life without google search (i.e. no “let’s google this”!)



A good algorithm makes all the difference

imagine life without google search (i.e. no “let’s google this”!)



A good algorithm makes all the difference

PageRank (**PR**) is an algorithm used by Google Search to rank websites in their search engine results.

PageRank

From Wikipedia, the free encyclopedia

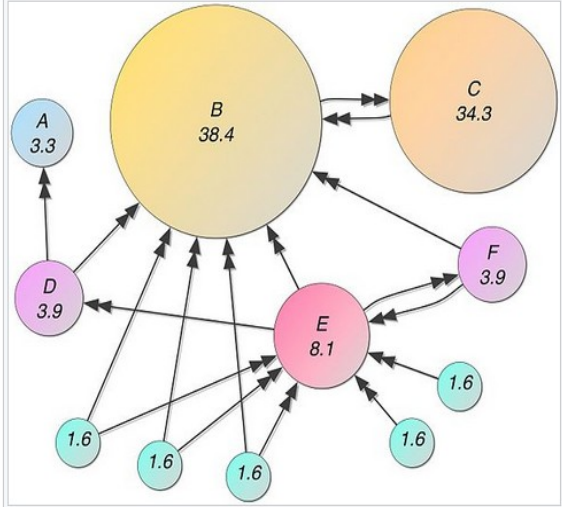
"Google search algorithm" redirects here. For other search algorithms used by Google, see Google Penguin, Google Panda, and Google Hummingbird.

PageRank (**PR**) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page,[1] one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

> PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.[2]

It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known.[3][4]

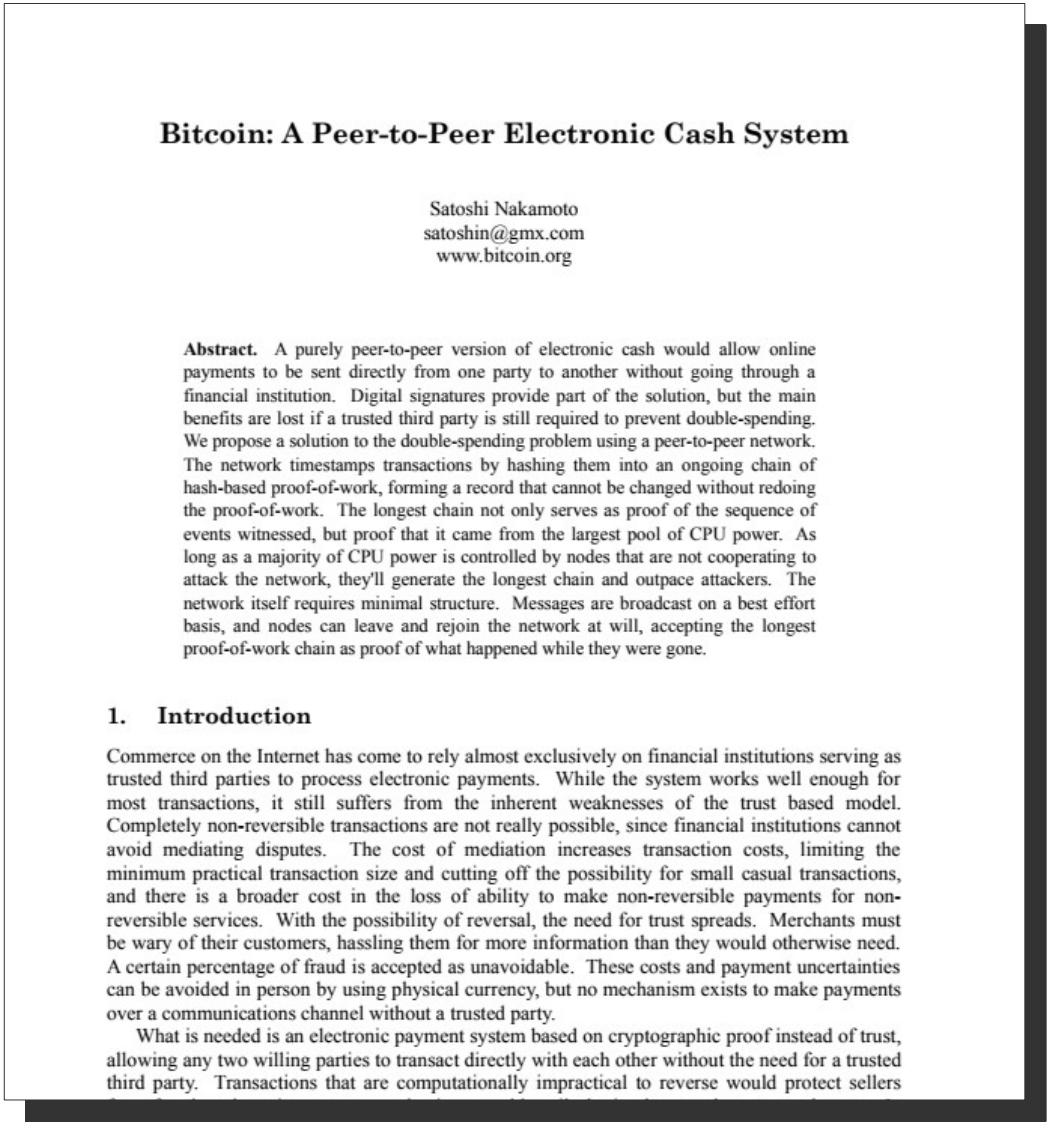
Contents [hide]



Mathematical **PageRanks** for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. (The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.

A good algorithm starts a revolution

have you heard of blockchains (e.g. bitcoin)



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers

What is a Data Structure

Algorithms manipulate data in different forms:

- number calculations manipulate numbers and digits
- recipes manipulate ingredients, kitchen tools, etc.
- data bases manipulate table entries (e.g. users, products, etc.)
- computer programs manipulate objects (e.g. in Java)

Data structures are structures carrying data, designed in such a way that makes data manipulation *and algorithms* easier.

Example data structures:

- arrays, stacks, lists, queues, sets, tuples, …
- trees, ordered trees, graphs, …

Example algorithms: Searching

Every algorithms course starts with searching. *– Anonymous*

SEARCH is the following problem:

- given an array of integers A and an integer k
- if k is in A then return its position, otherwise return -1

Notes:

- return its position means: return some such that $A[i] = k$
- it is OK if k occurs many times in A: just return one of its positions
- the array A can be arbitrarily large
- in general, SEARCH is about arrays of elements of any type (not necessarily integers)

First solution: linear search

Algorithm:

> We go through all the elements of the array A and compare each of them with k. At the end, if we found an i such that $A[i] = k$ then we return that i, otherwise we return -1.

This is a correct algorithm description, but written in plain language:

- it correctly describes a solution to SEARCH
- it has some imprecision: e.g. what does "*go through all the elements*" mean? Or "*if we found an i such that*"?
- it is not very helpful for writing a program solving SEARCH

A more technical-algorithmic language is needed in order to describe algorithms precisely.

We will use a programming language for that purpose: 

Linear search code

Here is our solution to search, in Python (top) and in Java (bottom)

```
def search(A, k):
   found = -1
   for i in range(len(A)):
      if A[i] == k:
         found = i
   return found
```

```
public static int search(int[] A, int k)
{
   int found = -1;
   for (int i=0; i<A.length; i++) {
      if (A[i] == k) found = i;
   }
   return found;
}
```

Python

We will use Python as our algorithmic language:

- all algorithms in the lectures will be written in Python
- most lab exercises will require you to write Python code

We will not ban Java though:

- most algorithms will also be given in Java
- in the exams, you can write algorithms in either Python or Java
  - you are not going to lose marks for choosing one over the other
  - but note this does not apply to pythonic lab exercises – almost all lab exercises require you to write Python code
- you can solve the mini-project using either Python or Java

Optimised solution: linear search with early return

We can go through all the elements of the array and check them, but return early if we find our k. This will save us time in most cases!

```
def search2(A, k):
    for i in range(len(A)):
        if A[i] == k:
            return i
    return -1
```

```
public static int search2(int[] A, int k)
{
    for (int i=0; i<A.length; i++) {
        if (A[i] == k) return i;
    }
    return -1;
}
```

Example algorithms: Searching in sorted array

Every algorithms course starts with searching. *– Anonymous*

SORTED-SEARCH is the following problem:

- given a **sorted** array of integers A and an integer k
- if k is in A then return its position, otherwise return -1

Notes:

- sorted means sorted in increasing order: e.g. [1,4,6,6,12,55,90]
- return its position means: return some i such that $A[i] = k$
- it is OK if k occurs many times in A: just return one of its positions
- the array A can be arbitrarily large

Linear search in sorted array

We can go through all the elements of the array and check them, but return early if we find our k or we have already arrived at a larger element:

```
def sortedSearch(A, k):
   for i in range(len(A)):
      if A[i] == k:
         return i
      if A[i] > k:
         return -1
   return -1
```

```
public static int sortedSearch(int[] A, int k)
{
   for (int i=0; i<A.length; i++) {
      if (A[i] == k) return i;
      if (A[i] > k) return -1;
   }
   return -1;
}
```

Second solution: binary search

Instead of starting from the left and going one-by-one, we start from the middle and keep dividing in half the elements we need to search:

- we look at the middle element of the array $A[mid]$:
  - if it equal to k then we found our element so we return mid
  - if it is less than k then we continue our search in the upper half of A
  - otherwise, we continue in the lower half of A
- we return -1 if we end up searching an array of length 0

Note:

- we keep dividing A in smaller and smaller pieces, so at some point we will either find k or end up with an empty array to search.
- Called binary search because we divide the array in half in each step. Linear search, instead, checks all the elements one by one.

Second solution: binary search

```
def binSearch(A, k):
    lo = 0
    hi = len(A)-1
    while (lo <= hi):
        mid = (lo+hi)//2
        if A[mid] == k:
            return mid
        else:
            if A[mid] < k:
                lo = mid+1
            else:
                hi = mid-1
    return -1
```

Second solution:

hi and low specify the part of A where we search at, which is A[lo], A[lo+1], ..., A[hi]

```
def binSearch(A, k):
    lo = 0
    hi = len(A)-1
    while (lo <= hi):
        mid = (lo+hi)//2
        if A[mid] == k:
            return mid
        else:
            if A[mid] < k:
                lo = mid+1
            else:
                hi = mid-1
    return -1
```

this is floor-division for integers, i.e. 4//2 = 2 and 5//2=2.

So (lo+hi)//2 gives us the middle point of the part that we search at.

If the part that we look at contains e.g. 6 elements, we cannot get an exact middle point, but the next best point to the left!

If we found our element then that is great, return.

If not, decide which half to search at:

- if A[mid]<k, search at A[mid+1], ..., A[hi]
- if A[mid]>k, search at A[lo], ..., A[mid-1]

and go back to beginning of the loop

Note that the above algorithm returns either when it finds k, or when it reaches the point of lo>hi (i.e. when the examined part of A has length 0).

So, if k is not in A, we need to loop until we get lo>hi.

Second solution: binary search

```
public static boolean binSearch(int[] A, int k)
{
  int lo = 0;
  int hi = A.length-1;
  while (lo <= hi) {
    int mid = (lo+hi)/2;
    if (A[mid] == k) return mid;
    else
      if (A[mid] < k) lo = mid+1;
      else hi = mid-1;
  }
  return -1;
}
```

Summary

We started with an introduction to what ADS is about:

- why are (good) algorithms important
- why are data structures important

We also looked at search algorithms:

- simple linear search vs optimised one
- sorted linear search vs binary search

Exercises

1. Write a function

   ```
   def isIn(A, k)
   ```

   that takes an array `A` and an integer `k` and returns `True` if `k` is in `A`, and `False` otherwise.

2. Recall the optimised linear search algorithm:

   ```
   def search2(A, k):
       for i in range(len(A)):
           if A[i] == k:
           return i
       return -1
   ```

   Let `A` be an array of size 1024 and suppose we run `search2(A,42)`. How many times will the for loop be repeated in the worst case?

   Recall the optimised linear search algorithm:

   ```
   def sortedSearch(A, k):
       for i in range(len(A)):
           if A[i] == k:
               return i
           if A[i] > k:
               return -1
       return -1
   ```

   Suppose now we run `search2(A,42)` on `A` that is of size 1024. How many times will the for loop be repeated in the worst case?

3. Recall the binary search algorithm:

   ```
   def binSearch(A, k):
       lo = 0
       hi = len(A)-1
       while (lo <= hi):
           mid = (lo+hi)//2
           if A[mid] == k:
               return mid
           else:
               if A[mid] < k:
                   lo = mid+1
               else:
                   hi = mid-1
       return -1
   ```

   Suppose we run `binSearch(A,42)` on `A` of size 1024. How many times will the while loop be repeated in the worst case?

4. Write a function `append` that takes an array `A` and an integer `k` and returns a new array which contains the same elements as `A` but with `k` appended at the end.

5. Write an optimised function `searchLast` that takes an array `A` and an integer `k` and returns the index of the last occurrence of `k` in `A` (or -1).

Exercises – Solutions

1.
```
def isIn(A, k):
    for i in range(len(A)):
        if A[i] == k:
            return True
    return False
```

2. Optimised linear search:
   The worst case is when k is not in A, or when it is only in the last position of A, in which case we will need to loop through all the elements of the array. That is, we will repeat the for loop 1024 times.

   Sorted linear search:
   The worst case is when k is greater than all elements of A (why? because sorted search will finish early if it finds an element greater than k), or when it is only in the last position of A. In that case we will need to loop through all the elements of the array. That is, we will repeat the for loop 1024 times.

3. Binary search:
   The worst case is when we keep dividing the part of the array that we examine all the way until we reach a point where lo>hi, i.e. when the part that we examine has length 0.

   Now, at each loop we start with an array part of length hi-lo+1 and we end up with one of length:

   - (hi-lo)/2, if (hi-lo+1) is odd
   - (hi-lo+1)/2 – 1 or (hi-lo+1)/2, if (hi-lo+1) is even

   So, starting with lo = 0 and hi=1023, in the worst case we have:

   hi-lo+1 = 1024 -> 512 -> 256 -> 128 -> 64 -> 32 -> 16 -> 8 -> 4 -> 2 -> 1 -> 0

   i.e. the while loop is repeated 11 times.

4.
```
def append(A, k):
    B = [0 for i in range(len(A)+1)]
    for i in range(len(A)):
        B[i] = A[i]
    B[len(A)] = k
    return B
```

5.
```
def searchLast(A, k):
    for i in range(len(A)-1,-1,-1):
        if A[i] == k:
            return i
    return -1
```