CS 341 — Lecture 1

Bartosz Antczak *Instructor: Bin Ma* *September 7, 2017*

1.1 Learning Goals of this Course

- Learn algorithms (already looked at sorting algorithms in previous CS courses)
- Algorithm analysis
- How to adapt an algorithm to solve new problems
- Algorithm design
- Implementation (not the key focus of the course, but still touched upon)

1.2 Definition

A **problem** defines:

- A format of *input*
- A property of *output*

An **algorithm** is a solution to a problem. Multiple algorithms may exist for a particular problem. We analyze an algorithm by determining its time and space complexity. We also look at its easiness of implementation. Algorithms in this course are easy to implement.

Example 1.2.1. *Find the missing number*

- **Input:** an array A of length $n-1$ containing all but one number from 0 to $n-1$
- **Output:** the missing number

Possible solutions:

Algorithm	Time complexity
$\left(\sum_{i=0}^{n-1} i\right) - \left(\sum_{j=0}^{n-2} A[j]\right)$	$O(n)$
Sort and scan	$O(n \log n)$
$\left(\bigoplus_{i=0}^{n-1} i\right) \oplus \left(\bigoplus_{j=0}^{n-2} A[j]\right)$	*not mentioned*
Hashing	$O(n)$, space $O(n)$

Example 1.2.2. *Duplicate elements*

- **Input:** an array A of distinct integers for which there exist two numbers that are the same
- **Output:** the duplicate number

Possible solutions:

Algorithm	Time complexity
Compare every pair	$O(n^2)$
Hashing	$O(n)$ average, space $O(n)$
Sort and check	$O(n \log n)$

The hashing algorithm would be implemented as:

```
for i=1 to n
    if (Hash.contains(A[i]) output A[i]; exit
    else Hash.add(A[i])
```

The two previous algorithms made an assumption of the model of computation. They assumed:

- Cost of basic operation
- Random access of memory
- Random number generator (which is very useful for computers)

Example 1.2.3. *Min and max*

- **Input:** an array A of size n containing integers
- **Output:** the largest and smallest integers in A

A very trivial algorithm exists:

```
max = A[0]
min = A[0]
for i=1 to n-1
    if (A[i] > max) max = A[i]
    if (A[i] < min) min = A[i]
```

This algorithm has a runtime of $O(n)$, which is our best possible runtime. Great! Is there a way we can optimize this algorithm even more? Yes!
Note that this algorithm performs $2n$ comparisons — two comparisons for every item in our array; however, we can devise an algorithm that performs less comparisons and still works:

```
min = A[0]
max = A[1]
for int k=2 to n/2
    if (A[2k-1] > A[2k])
        if (A[2k-1] > max) max = A[2k-1]
        if (A[2k] < min) min = A[2k]
    else
        if (A[2k-1] < min) min = A[2k-1]
        if (A[2k] > max) max = A[2k]
```

This algorithm has three comparisons for every two items in the list; ergo, it only performs $\frac{3}{2}n$ comparisons.