CSE 4502/5717 Big Data Analytics – Spring 2018
Notes taken by: Lina Kloub
Lecture 1:

Types of problems:

- Decidable:
  - Tractable: These are problems for which we can devise algorithms that take reasonable amounts of time. Examples: Matrix multiplication, sorting, etc.
  - Intractable: These are problems for which we can devise algorithms but the best known algorithms take very long times to terminate. Examples: Clique, TSP.
- Undecidable – these are problems for which we cannot devise algorithms. Example: The halting problem.

Performance measures:

1. **Time Complexity:**
   - Total number of basic operations (such as +, -, *, /, comparison, etc.) performed by the algorithm.
   - Is a function of the input size. Input size is the number of the memory cells needed to describe the problem instance.

   Examples:

   1. Problem: Sorting n numbers.
      Input size: n
   2. Problem: Multiplying two n x n matrices.
      Input size: $2n^2$

2. **Space Complexity:**
   - Total number of memory cells needed to solve the problem.

Note: both time and space complexities are integer functions of the input size.

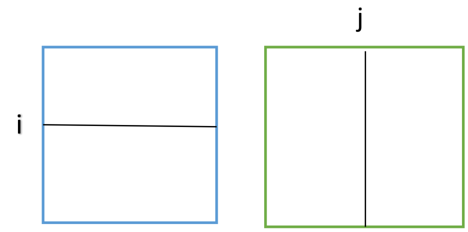
Algorithm specification:

Any understandable description is acceptable. To make the description concise the following constructs can be used:

1. Assignment statement.
2. While loops.
3. If-then-else statements, etc.

<u>Example:</u>

Input: two matrices A and B of size n x n each.
Output: multiplication of the two matrices C = A*B



$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

```
for i = 1 to n do
    for j = 1 to n do
        C[i,j] = 0.0;
        for k = 1 to n do
            C[i,j] = C[i,j] + A[i,k] *  B[k,j];
```

Asymptotic functions:

Enable us to simplify functions.

1. We say $f(n) = O(g(n)) \quad if \quad f(n) \leq c\, g(n)\ \forall n\ \geq n_0$, for some constants c and n_0.
   ex: Matrices multiplication: total number of operations:
   *[n + (n-1)] * n²*
   $= 2n^3 - n^2$
   $= O(n^3)$
2. W say $f(n) = \Omega(g(n)) \quad iff \quad g(n) = O(f(n))$
3. We say $f(n) = \theta(g(n))\ iff\ f(n) = O(g(n))\ and\ g(n) = O(f(n))$.

We could identify three different time complexities (or run times): Best case, Worst case, and average case.

<u>Example:</u> Search problem

Input: A sequence of numbers $X = k_1, k_2, k_3, k_n$; and another number x
Output: "Yes" $if\ x \epsilon X$
"No" otherwise
-$let\ D\ be\ the\ set\ of\ all\ possible\ inputs$
-$let\ T_I\ be\ the\ run\ time\ on\ input\ I\ \epsilon\ D$
-$Average\ run\ time\ complixity = \sum_{I\ \epsilon\ D} \frac{T_I}{|D|}$
-$More\ generaly, let\ P_I be\ the\ probability\ of\ input\ I\ \epsilon\ D$
-$Average\ run\ time = \sum_{I \epsilon D} P_I T_I$
-$Average\ run\ time\ for\ the\ search\ algorithm = \frac{(1+2+\cdots+n)+n}{(n+1)} = \frac{n}{2} + \frac{n}{(n+1)}$

Randomized algorithms:

A randomized algorithm is one where in certain decisions are made based on outcomes of coin flips.

1. **Monte Carlo algorithms:**
   algorithms whose run times can be pre-specified and which have a chance of producing an incorrect result with a *low probability.*
2. **Las Vegas algorithms:**
   always terminate with the correct answer. The run time of a Las Vegas algorithm is a random variable.
3. *Low probability*: a probability of $\leq \ n^{-\alpha}$
   where n is the input size, and α is a probability parameter, typically assumed to be a constant.

High probability: means a probability of $\geq (1 - n^{-\alpha})$.

Example:

Input: An array of n numbers $A[1{:}n]$, A has $\frac{n}{2}$ copies of one element and the other elements are distinct.
Output: the repeated element
Algorithms:
1. Iterate through the elements. Run time = $(n-1) + (n-2) + \cdots + \left(\frac{n}{2}\right) = \ \theta(n^2)$
2. Sorting can be used to solve this problem. Sorting takes $\theta(n \log n)$ time.
3. We can solve this problem in linear time. The idea is to group the input into groups of size 3 each and look for duplicates within the individual groups.

We can also see that any deterministic algorithm to solve this problem will need Ω(n) time in the worst case. We can devise an efficient Las Vegas algorithm to solve this problem.

A Las Vegas algorithm:

Repeat:

```
flip an n-sided coin to get i;
  flip an n-sided coin to get j;      Basic step
  if i≠j and A[i] = A[j] then
    output A[j] and quit;
```

Forever

Analysis:

Probability of success in one basic step $= \frac{\left(\frac{n}{2}\right)\left(\frac{n}{2}-1\right)}{n^2}$.
This probability is $\geq \frac{1}{5} \quad \forall n \geq 10$.
Probability of failure in one basic step is no more than $\frac{4}{5}$.
Probability of failure in k successive basic steps is $\leq \left(\frac{4}{5}\right)^k$;
we want this to be $\leq n^{-\alpha}$.

This happens when $k \log\left(\frac{4}{5}\right) \leq -\alpha \log n$;
i.e., when $k \geq \frac{\alpha \log n}{\log \frac{5}{4}}$.

Definition:
$We\ say\ that\ the\ run\ time\ of\ Las\ Vegas\ algorithm\ is\ \tilde{O}\big(f(n)\big)\ if\ the\ run\ time\ is\ \leq c\alpha f(n), with\ a\ probability\ of\ \geq (1 - n^{-\alpha}), \forall n \geq n_0, where\ c\ and\ n_0\ are\ some\ constants.$

It follows that the run time of the above Las Vegas algorithm is $\tilde{O}(\log n)$.