Lecture 01

Abstract Data Type (ADT):= set of objects with set of operations on these objects

- two components:
  1. object/data
  2. operations
- objects: integers; operations: `add(x,y), multiply(x,y)`, etc
- stack
  - objects: lists (or sequences)
  - operations: `push(S,v), pop(S), isEmpty(S)`
- ADT's are important for specification; provide modularity and reuse since usage is independent of implementation

Data structure:= specific implementation of an ADT: a way to represent the objects and an algorithm for each operation

Stack implementations:

(a) linked list
  - $head \to [X] \to [X] \to [\,]$
  - keep pointer to head
  - `isEmpty`: test `head == None`
  - `push`: insert at front of list
  - `pop`: remove front of list (if not empty)

(b) array with counter (size of stack)

5	7	9	2	12	···

In general,

- ADT describes *what* (data and operations)
- data structure describes *how* (storing data, performing operations)

In this course we will encounter many ADTs and many data structures for these ADTs. Use careful analysis (of correctness and complexity) to compare possibilities.

Algorithm Analysis (Review)

Complexity:= the amount of $\underbrace{\text{resources}}_{\text{running time, memory (space)}}$ required by an algorithm, measured as a function of input size.

- input size measured as..
  - number → # bits
  - list → # elements
  - graph → # vertices
- Important: measure must be roughly proportional to true bit-size (# of bits required to fully encode input). In practice, allow ourselves to ignore log factors, e.g., we use size$([a_1, a_2, \dots, a_n]) = n$ when it is really size$(a_1) + \cdots +$ size$(a_n) \propto n$ only if each a_i has constant size.
- running time measured at high level asymptotic notation - Big O, Big Θ, Big Ω