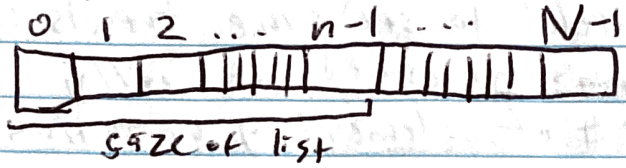
List ADT — stores ordered list of elements, w/ operations to read, write, insert, remove
↑ specifies state & operations

Array Implementation: (of List ADT)



n = list size
N = capacity of array

Linked List Implementation: (of List ADT)
Idea - a node stores one element & reference to next node.

How to get(i)?
- Start at head
- Go "next" i times.

How to add(i,y)?
- Make node storing y,
- Change i-1 "next" to new node
- Change new node "next" to i+1.

How to remove(i)?
- Change i-1 "next" to i+1.

Array - searching List fast, shifting list slow
Linked List - searching List slow, shifting List fast.

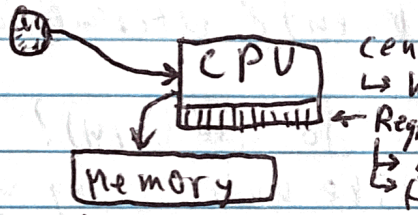
Insert at front - LL faster
Insert at back - Array faster
Insert randomly - Array faster.

List Restrictions (List-Related ADTs).
Deque - can only access first & last element.
Stack - can only access last element in the list
Queue - can only add to back, can only read/remove from front

Running Times for Add(x, y).
Differences for Diff Computers?
→ Scale: Absolute running times change
→ "trend" (relative to other implementations) approx. same

What assumptions are needed to see same trends?
- Elementary operations performed efficiently
→ execution time ~ # of elementary operations

Section 2 RAM model



central processing unit
↳ basic ("primitive") operations, ~1/tick
↳ store primitive values
↳ fixed # of these

↳ stores data, code, variables/values, etc.

Primitive Ops:
- copy values from memory → registers
- basic arithmetic on registers
  - increment value of register
  - add contents of register 1 to 2, store in 3
  - branching: if-then-else.
- copy values from register → memory.

Running time determined by
- Speed of clock
- # of clock cycles / primitive operation
- # of primitive operations performed. } what we control

Goal: Understand trends for running times that are "independent" of hardware particulars.