CS241, Jan 23, 9:30

ADT = abstract data type

- ↳ 2 parts: short description of what the data type represents (abstractly)
- ↳ List of operations that the data type is capable of.
  - These operations only tell us WHAT the ADT does, not HOW it does it.

Interface vs **Implementation**

WHAT something does — HOW it does it.

Ex

- Steering wheel → cables?
- gas pedal → engine?
- brake pedal → activate brakes

} diff for gas vs elec cars

ADT = interface

So to be useful, an ADT must be combined w/ an implementation.

Distinction between an ADT vs a "data structure"

- ADT ↓ interface
- "data structure" ↘ - implementation
  - implementation + interface together

LIST ADT

→ Description: A list consists of a collection of positions, each of which contains a single element of the list. Each position has a unique index, which is an integer in the range 0 .... n-1, where n = # of elements in the list.

0	1	2	3	4	5	6	…	n-1
							…	

Important: This description says nothing about how the list is stored in memory!

→ Operations

- → Return the element at a specific index. (GET)
- → Append an item to end of list.
- → Put an item @ beginning of list.
- → Put an item anywhere in the list. (SET)
- → Reverse the list.
- → merge two lists.
- → get the size of list.

Possible implementations of List ADT

- What is a good choice for a data structure to implement this ADT?

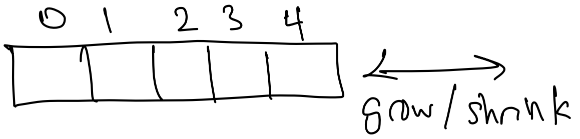
- We are going to use a Java array.

In most programming langs, size of arrays cannot be modified once created.

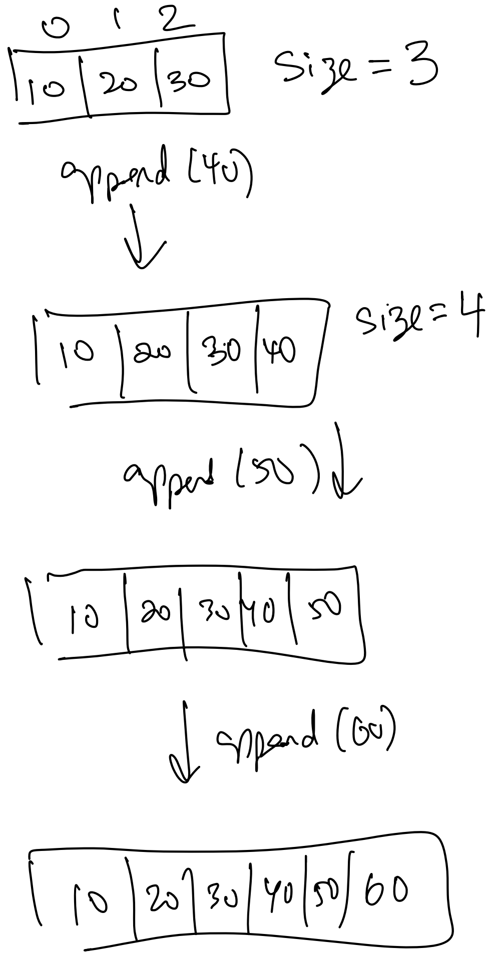
Why? int – 4 bytes same for arrays int array[10]; = 10×4 bytes
= 40 bytes

Java arrays give use "random access" → we can access any element of the array very quickly.

RList interface



→ int size

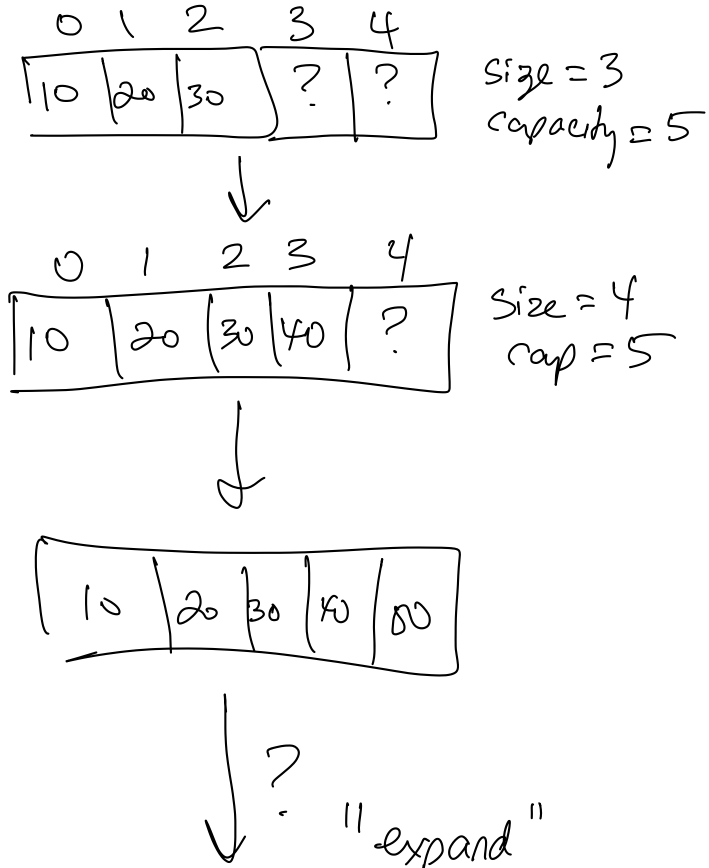


RList implementation

→ int size

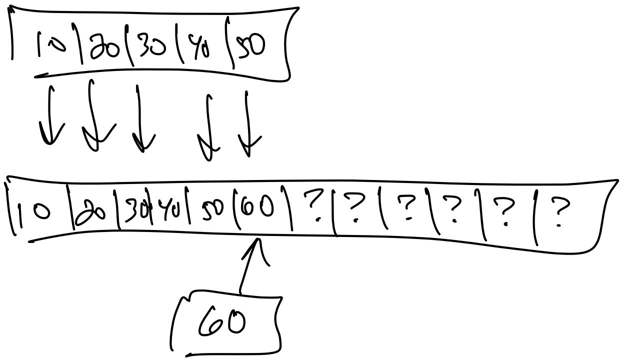
→ "capacity" of the array

Reserve extra capacity when we create the array.



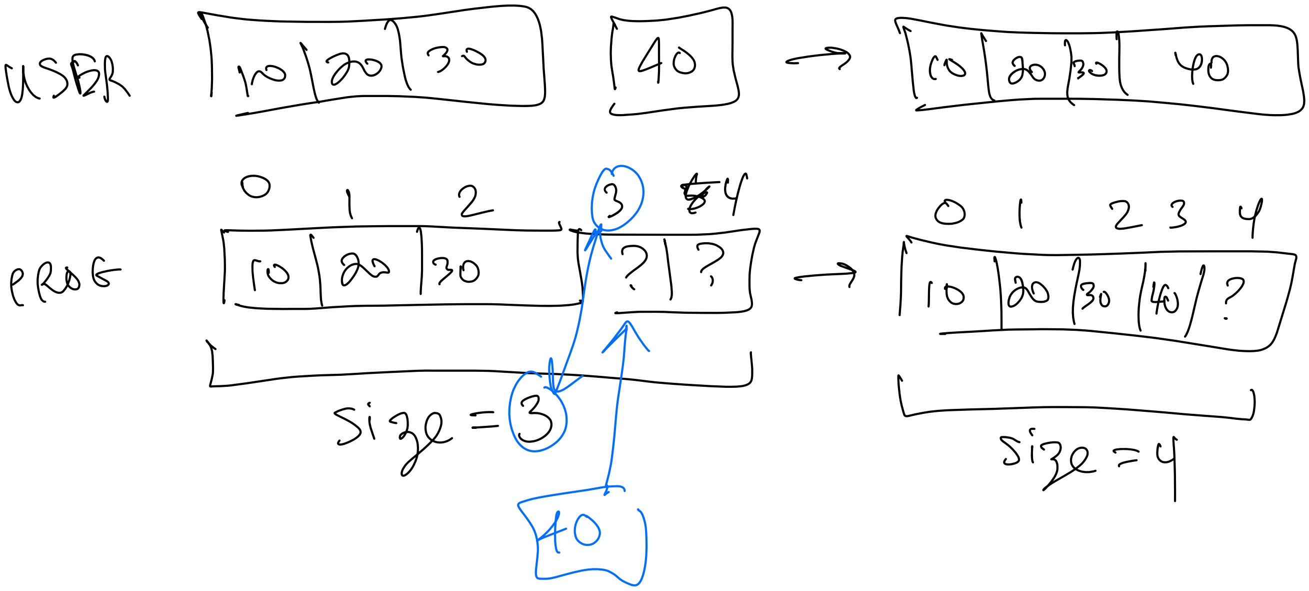
"expand"

- make a new, bigger array
- copy the old info to the new array



TIME / SPACE TRADE-OFF

APPEND


USER
10 | 20 | 30
40
→
10 | 20 | 30 | 40
PROG
0 1 2 3 ~~3~~4
10 | 20 | 30 | ? | ?
size = 3
40
→
0 1 2 3 4
10 | 20 | 30 | 40 | ?
size = 4