Week 6: Generics and sorting

Friday, October 9, 2020 8:25 AM

Generics

→ placeholder for any type of object
  * cannot be primitive
→ allows code reusability
→ declare in class header
  public class ArrayList<E> ...
→ must provide type when creating object
  * ArrayList<String> list = ...
  * get warning about raw type if don't state

public ArrayList<E>(capacity){
  :
}

E → String

→ at compile time, Java replaces all parameters with given type
  → default to object
  → no bugs from generics during runtime
    * all errors or warnings occur before running
  * no trace of parameter left as far as Java sees

* ? → wildcard
  → cover week 10—maps
  * do not use this week

Restrictions

→ can't instantiate a generic object
  i.e. E var = new E();
→ can't create arrays of generics
  i.e. E[] arr = new E[size];
  → instead create object array & cast
    E[] arr = (E[]) new Object[size];
    * warning: unchecked cast *
    → can safely ignore → @SuppressWarning("unchecked")
→ dynamic array that is generic
  ArrayBag<Integer> bag;
  → add method (String s)
    * error
* cannot check instanceof generic
  * no obj instanceof E

→ <? extends E> * lower bound
  ↳ whatever type

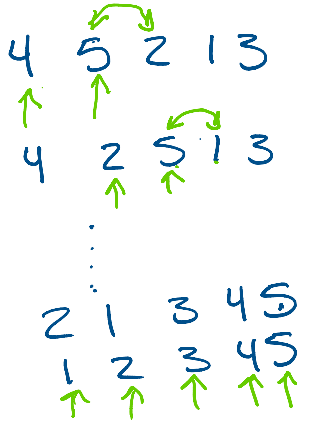
→ Student class that extends Person

LinkedCollection: E = person
  → add any collection that has type extending person
→ TA extends Student extends person

Sorting

→ Bubble Sort
  → while swaps have been made
    → loop through & compare each element to one after

→ while swaps have been made
  → loop through & compare each element to one after
    → if out of order: swap



runtime:

worst-case: $O(n^2)$

→ 5 4 3 2 1
4 5 3 2 1
4 3 5 2 1
4 3 2 5 1
4 3 2 1 5

* for each element you go through all other elements

Best-case:

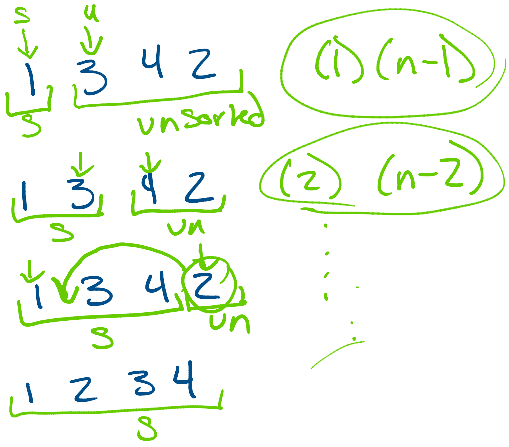
→ already sorted

$O(n)$

→ loop through list ← $O(n)$
  → each element gets compared to next ← $O(1)$

Insertion Sort

→ sorted & unsorted section
→ for each element in unsorted, find where it belongs in sorted
  → insert



Run time:

Worst-case:

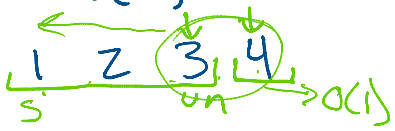
$O(n^2)$

→ for each element, looking at all other elements

Best-case:

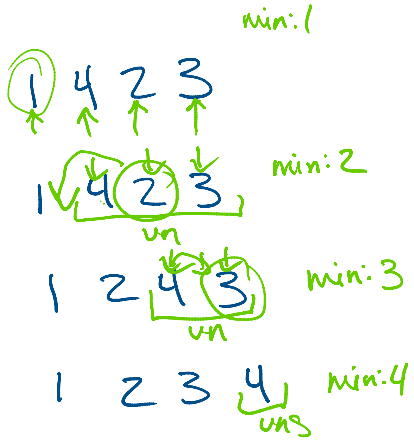
→ already sorted

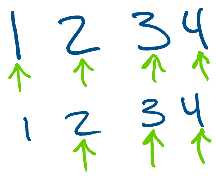
$O(n)$



Selection Sort

→ sorted & unsorted
→ repeatedly look for smallest element in unsorted, put at end of sorted section
  → LL: insert between sorted & unsorted
  → Arrays: swap smallest w/ first in unsorted





Worst case: n^2

Best case: $O(n^2)$