Big O & ArrayList

15-121 Fall 2020

Margaret Reid-Miller

Today

- Office Hours: Thursday afternoon (time to be announce on Piazza)

- Big O
- Amortized runtime
- ArrayLists

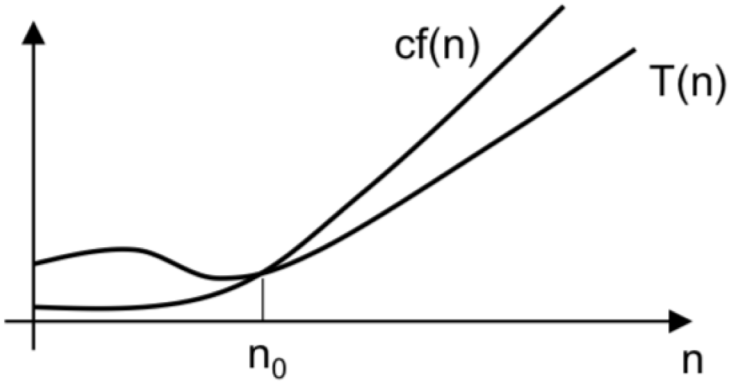
How do we determine how efficient (fast) and algorithm is?

Key Idea:

The running time of an algorithm depends on the size of the problem it's solving.

Big O: Formal Definition

- Let T(n) – the number of operations performed in an algorithm as a function of n.
- $T(n) \in O(f(n))$ if and only if there exists two constants, $n_0 > 0$ and $c > 0$ and a function f(n) such that for all $n > n_0$, $cf(n) \geq T(n)$.



How long does it take to

- say "California" and then
- fly to California

The time to fly to California (roughly 6 hours) dominates the time to say "California", so we ignore the time to say "California".

Big-O notation

Let n be the size of the problem (input):

Time(n) = $4n + 9 = O(n)$

Time(n) = $2n^2 - 4n + 1 = O(n^2)$

Time(n) = $\log_3(n) = O(\log n)$

Time(n) = $3^n = O(3^n)$, not $O(2^n)$!

There is no c that satisfies $c \cdot 2^n \geq 3^n$ for arbitrarily large n.

More on Big O

- Big O gives us an upper-bound approximation on the complexity of a computation.

- That is, think of Big O as "<="
  - n + 1000 is O(n), but it's also $O(n^2)$ and $O(n^3)$. We try to keep the bound as tight as possible, though, so we say n + 1000 is O(n).

Big-O when algorithm is A then B

- Suppose an algorithm is do

  A followed by B.

- Then the overall complexity of the algorithm is

  max (O(A), O(B)).

Examples:

$O(\log n) + O(n) = O(n)$

$O(n \log n) + O(n) = O(n \log n)$

$O(n \log n) + O(n^2) = O(n^2)$

$O(2^n) + O(n^2) = O(2^n)$

Big-O when algorithm is A encloses B

- *E.g., N*ested loops: A is the outer loop B is the inner loop, or A calls B repeatedly
- Then the overall complexity of the algorithm is

  O(A) * O(B),

  where O(A) excludes the complexity of B.

Examples:

O(n) * O(log n) = O(n log n)

O(n log n) * O(n) = $O(n^2 \log n)$

O(n) * O(1) = O(n)

How do we grow the `contacts` array when it is full?

We create a new array that is larger than the contacts array and copy all the elements to the new array.

Sometimes, the cost to add a single Person is O(1) because there is room in `contacts`.

But sometime the cost to add a single person is O(n), n = `numContacts`, because we need to expand the array and copy n elements.

What is the worst case runtime for calling `add(name, number)` n times, when we start with an array of length 1?

Number of copies to grow an array to length n starting with an array of length 1.

Grow by 1 each time:

The array is full when

1, 2, 3, 4, 5, 6, … elements in the array

After adding n elements we copied

$1 + 2 + 3 + 4 + \ldots(n-1) = n(n-1)/2 = O(n^2)$ elements

Grow by 100 each time:

The array is full when

100, 200, 300, 400, 500, … elements in the array

After we have added n = 100*k elements we copied

100 + 200 + 300 + … + 100(k-1) elements

$= 100k\ (k-1)/2$

$= n\ (n/100 - 1)/2 = O(n^2)$

Growing by a constant does $O(n^2)$ copies

By doubling the array length, adding n elements does O(n) copies.

The array is full when we have
1, 2, 4, 8, 16, 32, … elements

After we have added 32 elements we copy
1 + 2 + 4 + 8 + … + 16 = 31 elements to a larger array

After we have added 64 elements we copy
1 + 2 + 4 + 8 + … + 16 + 32 = 63 elements to a larger array

After adding n elements,
we have copied a total of O(n) elements to a larger array!

What is the worst-case run time for adding n Persons to a ContactList?

Therefore, the worst-case runtime for n calls to `add()` is O(n).

We, therefore, say that the ***amortized*** worst-case runtime for a single call to `add()` is O(1).

Definition: ***Amortized worst-case runtime*** is the expected runtime per operation of a worst-case **sequence of n operations**.

> (This is not the same as ***Average runtime***, which is runtime of **a single operation** averaged over **all distinct inputs**.)

ArrayLists

Abstract Data Types vs Data Structures

- **Abstract Data Type**: An ADT is a formal description of the behavior (semantics) of a type.
  1. The representation/organization of the data is hidden.
  2. Specifies the operations that can be applied to the underlying data independent of any particular implementation. (Defines the interface of the ADT.)
- **Data Structure**: A data structure is a concrete representation/organization of data and algorithms in a specific implementation of a ADT.

The ArrayList Class

- For Java folks, an ArrayList is like an array, but:
  - It's a class, so we construct it and call methods on it.
  - It's resizable. It grows as we add elements and shrinks as we remove them.

- For Python folks, an ArrayList is like a Python list, but:
  - We do not use subscript notation.

- ArrayLists (like arrays) are homogeneous.

```
ArrayList <String> names = new ArrayList<String>();
```

`java.util.ArrayList<E>`

ArrayList methods

`boolean` **`add`**`(E obj)`

Appends `obj` to end of this list; returns `true`

`void` **`add`**`(int index, E obj)` `(0 <= index <= size)`

Inserts `obj` at position `index`

`void` **`clear`**`()`

Removes all the elements from this list.

`boolean` **`contains`**`(Object obj)`

Returns `true` if this list contains `obj`.

`E` **`get`**`(int index)`

Returns the element at position `index (0 <= index < size)`

`int` **`indexOf`**`(Object obj)`

Returns the index of the first occurrence of `obj` in this list, or returns -1 if this list does not contain `obj`

`java.util.ArrayList<E>`

ArrayList methods

`boolean` **`isEmpty`**`()`

Returns true if the list is empty and false otherwise

`E` **`remove`**`(int index)` `(0 <= index < size)`

Removes element at position `index`;
Returns the element formerly at position `index`.

`boolean` **`remove`**`(Object obj)`

Removes the first occurrence of `obj`, if present;
Returns true if this list contained `obj`, `false` otherwise.

`E` **`set`**`(int index, E obj)` `(0 <= index < size)`

Replaces element at position `index` with `obj`;
Returns the element formerly at position `index`.

`int` **`size`**`()`

Returns the number of elements in the list.

ArrayList complexity

	Worst	Best
`int size()`	O(1)	
`add(E obj)`	O(1)*	
`add(int index, E obj)`	O(n)	O(1)*
`get(int index)`	O(1)	
`set(int index, E obj)`	O(1)	
`contains(Object obj)`	O(n)	O(1)
`remove(int index)`	O(n)	O(1)
`remove(Object obj)`	O(n)	
`indexOf(Object obj)`	O(n)	O(1)
`clear()`	O(1)	
`isEmpty()`	O(1)	

*amortized

ArrayList demo

```
import java.util.ArrayList;
ArrayList<String> names = new ArrayList<String>();
names.add("Margaret");          // Margaret
names.add("Dave");              // Margaret Dave
names.get(1);                   // returns Dave
names.set(0, "Mark");           // Mark Dave
names.add(1, "Tom");            // Mark Tom Dave
names.remove(1);                // Mark Dave
```

Wrapper Classes

- ArrayLists can only store references to objects, not primitive values.
- All primitive types have a corresponding object type (wrapper class).
- Example: `Integer`, `Double`, `Boolean`,...

```
Integer x = new Integer(62);
Integer y = new Integer("12");
int n = y.intValue();
```

ArrayLists with Integer

```
ArrayList<Integer> intList =
                        new ArrayList<Integer>();

intList.add(new Integer(3));
int n = intList.get(0).intValue();
```

YUCK!

Java does conversion between primitive and wrapper types

```
ArrayList<Integer> intList =
                   new ArrayList<Integer>();

intList.add(3);          // converts int to Integer
int n = intList.get(0); // converts Integer to int
```

- Like mixing primitive types, based on the types of literals, parameters and variables, Java converts between primitive and wrapper types.

Auto-boxing

```
Integer a = i++;     // auto-boxing
Integer b = j + 2;
int k = a + b;       // auto-unboxing
System.out.println(
  a.toString() + b.toString()); // concat
System.out.println(a + b);     // unboxed add
```

Warning:

```
if (list.get(0) == list.get(1))
```

Does not auto-unbox! It compares the references to Integer objects.

Example: count

```
// Returns the number of names in the given list
// with the given number of letters
public static int count( ArrayList<String> names,
                         int numLetters) {
   int count = 0;
   for (int i = 0; i < names.size() ; i++) {
      if (names.get(i).length() == numLetters) {
         count++;
      }
   }
   return count;
}
```

Example: `getNamesOfLength`

```
// Returns a list of names in the given list
// that have the given number of letters
public static ArrayList<String> getNamesOfLength(
      ArrayList<String> names, int numLetters) {
   ArrayList<String> result;
   result = new ArrayList<String>();
   for (int i = 0; i < names.size() ; i++) {
      if (names.get(i).length() == numLetters) {
         result.add(names.get(i)) ;
      }
   }
   return result;
}
```

Example: `removeNamesOfLength`

```
// Removes all names in the given list
// that have the given number of letters
public static void removeNamesOfLength(
        ArrayList<String> names, int numLetters) {
   for (int i = 0; i < names.size(); i++) {
      if (names.get(i).length() == numLetters) {
         names.remove(i);
      }
   }
}
```

Oops! When doesn't this code work correctly?

It won't remove 2 consecutive names.

Example: `removeNamesOfLength`

```
// Removes all names in the given list
// that have the given number of letters
public static void removeNamesOfLength(
        ArrayList<String> names, int numLetters) {

    for (int i = 0; i < names.size(); i++) {
        if (names.get(i).length() == numLetters) {
            names.remove(i);
            i--;
        }
    }
}
```

Solution 1:

Decrement i after each removal. Ugly

Example: `removeNamesOfLength`

```
// Removes all names in the given list
// that have the given number of letters
public static void removeNamesOfLength(
        ArrayList<String> names, int numLetters) {

    int i = 0;
    while (i < names.size()) {
        if (names.get(i).length() == numLetters)
            names.remove(i);
        else
            i++;
    }
}
```

Solution 2:

Increment only when don't remove.

Example: `removeNamesOfLength`

```
// Removes all names in the given list
// that have the given number of letters
public static void removeNamesOfLength(
        ArrayList<String> names, int numLetters) {

    for (int i = names.size()-1; i >= 0; i--) {
        if (names.get(i).length() == numLetters) {
            names.remove(i);
        }
    }
}
```

Solution 3:

Loop backward. Move only the elements you are keeping. *Sweet.*

Exercises

Rewrite `contactList` class using an `ArrayList` instead of an array. What fields do need? What fields can you drop?