CS 240

Data Structures and Algorithms I

Alex Vondrak

ajvondrak@csupomona.edu

September 26, 2011

Today's Lecture

1. Clarifications
   - What was the meaning of the weighted average percentage?
   - Are late penalties "flat"?
   - Is Java required?
   - What were the corresponding chapters of CLRS?
2. The Role of Data Structures
   - Case study: arrays
3. The Role of Algorithms
   - Case study: searching

Questions From Last Time

Weighted Average Percentages

The weighted average percentage is just a regular average, where assignments are weighted as in the syllabus

$$\frac{\sum \left(\text{weight}_i \times \left(\frac{\text{score}_i}{\text{max}_i} - \text{penalty}_i\right)\right)}{\underbrace{\sum \text{weight}_i}_{= 100\% \text{ at end of quarter}}}$$

where

$$\text{penalty}_i = \begin{cases} 0 & \text{if assignment } i \text{ is on time} \\ 10(n+1)\% & \text{if } 0 \leq n \leq 9 \\ 100\% & \text{if } n \geq 10 \end{cases}$$

and $n = \#$ school days properly between the submission & due dates

Note that late penalties are "additive" (i.e., "flat"), like

$$\frac{\text{score}_i}{\text{max}_i} - \text{penalty}_i$$

not "multiplicative", like

$$\frac{\text{score}_i}{\text{max}_i} - \text{penalty}_i \times \frac{\text{score}_i}{\text{max}_i}$$

$$= \quad \frac{\text{score}_i}{\text{max}_i} \times (1 - \text{penalty}_i)$$

Java will be required for the programming projects:

- Cal Poly's official instructional language
- Course topics depend on Java (e.g., generics)
- Project grades should be apples-to-apples
- Languages like C++ are similar enough that you don't gain much by using them

Questions From Last Time

CLRS Reference Chapters

From the 2nd edition:

Arrays	N/A—presumed background
Analysis	Chapter 2.2, Chapter 3
Searching	N/A—in the exercises
Generics	N/A—not Java-based
Stacks	Chapter 10.1
Queues	Chapter 10.1
Linked lists	Chapter 10.2
Recursion	Chapter 2.3
Hashing	Chapter 11

These appear to hold for the 3rd edition as well

Data Structure + Algorithm = Program

What is the point of this course?

- Become a more proficient programmer
- "Grab-bag" of common data structures & algorithms
- The thought process for designing your own

Data Structures

data *uncountable* or *plural noun*

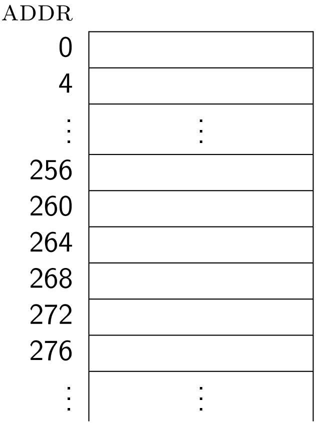
1. Plural form of "datum"; pieces of information
2. (*collectively*) information
3. A collection of object-units that are distinct from one another

structure *noun*, *pl* structures

1. a cohesive whole built up of distinct parts
2. the overall form or organization of something
3. a set of rules defining behavior
4. (*computing*) several pieces of data treated as a unit

Arrays

```
int[] array = new int[3];
for(int i = 0; i < array.length; i++)
  array[i] = 100;
```



Arrays

```
int[] array = new int[3];
for(int i = 0; i < array.length; i++)
  array[i] = 100;
```

ADDR		
0	256	← array
4		
⋮	⋮	
256	**int**[] object	
260	3	← array.length
264	0	← array[0]
268	0	← array[1]
272	0	← array[2]
276		
⋮	⋮	

Arrays

```
int[] array = new int[3];
for(int i = 0; i < array.length; i++)
  array[i] = 100;
```

ADDR		
0	256	← array
4	0	← i
⋮	⋮	
256	**int**[] object	
260	3	← array.length
264	100	← array[0]
268	0	← array[1]
272	0	← array[2]
276		
⋮	⋮	

Arrays

```
int[] array = new int[3];
for(int i = 0; i < array.length; i++)
  array[i] = 100;
```

ADDR		
0	256	← array
4	1	← i
⋮	⋮	
256	**int**[] object	
260	3	← array.length
264	100	← array[0]
268	100	← array[1]
272	0	← array[2]
276		
⋮	⋮	

Arrays

```
int[] array = new int[3];
for(int i = 0; i < array.length; i++)
  array[i] = 100;
```

ADDR		
0	256	← array
4	2	← i
⋮	⋮	
256	**int**[] object	
260	3	← array.length
264	100	← array[0]
268	100	← array[1]
272	100	← array[2]
276		
⋮	⋮	

Arrays

```java
int[] array = new int[3];
for(int i = 0; i < array.length; i++)
  array[i] = 100;
```

ADDR		
0	256	← array
4	3	← i
⋮	⋮	
256	**int**[] object	
260	3	← array.length
264	100	← array[0]
268	100	← array[1]
272	100	← array[2]
276		
⋮	⋮	

Arrays

What Are They Good For?

Pros

- They're a simple way to represent collections
- They map directly to computers' memory structures
- Contiguous chunks of memory make indexing as easy as

$$\texttt{base} + \|\texttt{word}\| \times \texttt{offset}$$

- Processors are highly optimized for array operations

Cons

- Fixed size $\implies$ less flexible
- Certain operations are complex (e.g., insertion in the middle)
- Too simplistic for many purposes

Algorithms

algorithm *noun*, *pl* algorithms; **related:** algorithmic, *adj*

1. A precise step-by-step plan for a computational procedure that begins with an input value and yields an output value in a finite number of steps

In Other Words

An algorithm is the way in which we solve a particular *computational problem*. E.g., the *searching* problem:

Input: Any array of **`int`**s, plus a single **`int`** to search for.

Output: The value **`true`** if the **`int`** is an element of the array, or the value **`false`** if it is not.

Any given input that satisfies the problem statement is called an *instance* of the problem

Analysis of Algorithms

In general, we always want our algorithms to be

- Efficient
  - Time
  - Space
- Correct
  - For every problem instance, the algorithm halts with the expected output

In reality, we often make trade-offs

- Time versus space
- Approximation algorithms for NP-complete problems (CS 331)
- Randomized algorithms that run a small chance of being incorrect