Data Abstraction
Experimental Studies (4.1)

Janyl Jumadinova

October 12, 2020
Analysis of Algorithms

- Most algorithms transform input objects into output objects.
- The running time of an algorithm typically grows with the input size.
Average case time is often difficult to determine. We focus on the worst case running time. Easier to analyze. Crucial to applications such as games, finance and robotics.
Experimental Studies

1. Write a program implementing the algorithm.
2. Run the program with inputs of varying size and composition, noting the time needed.
3. Plot the results and analyze them.
Timing an Algorithm in Java

// record the starting time
long startTime = System.currentTimeMillis();

/* run the algorithm */

// record the ending time
long endTime = System.currentTimeMillis();
long elapsed = endTime - startTime;
The Random Access Machine (RAM) Model

- A CPU
- An potentially unbounded bank of memory cells, each of which can hold an arbitrary number or character.
- Memory cells are numbered and accessing any cell in memory takes unit time.
Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult.
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used.
Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation.
- Characterizes running time as a function of the input size, \( n \).
- Takes into account all possible inputs.
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment.
Pseudocode

- High-level description of an algorithm.
- More structured than English prose.
- Less detailed than a program.
- Preferred notation for describing algorithms.
- Hides program design issues.
Pseudocode Details

Control Flow

- if ... then ... [else ...]
- while ... do ...
- repeat ... until ...
- for ... do ...
- Indentation replaces braces
Pseudocode Details

Method Declaration

- Algorithm method(arg [, arg ...])
- Input
- Output
Pseudocode Details

**Method Declaration**
- Algorithm method(arg [, arg ...])
- Input
- Output

**Method Call**
- method(arg [, arg ...])
Pseudocode Details

Method Declaration
- Algorithm method(arg [, arg ...])
- Input
- Output

Method Call
- method(arg [, arg ...])

Return value
- return expression
Expressions

- $\leftarrow$ Assignment
- $=$ Equality testing
- $n^2$ Superscripts and other mathematical formatting allowed
Pseudocode Example

INSERTION-SORT(A)
1. for j = 2 to n
2. key ← A[j]
3. // Insert A[j] into the sorted sequence A[1..j-1]
4. j ← i - 1
5. while i > 0 and A[i] > key
7. i ← i - 1
8. A[j+1] ← key
Primitive Operations

- Basic computations performed by an algorithm.
- Identifiable in pseudocode.
- Largely independent from the programming language.
- Exact definition not important.
- Assumed to take a constant amount of time in the RAM model.

Examples:
- Evaluating an expression.
- Assigning a value to a variable.
- Indexing into an array.
- Calling a method.
- Returning from a method.
Primitive Operations

- Basic computations performed by an algorithm.
- Identifiable in pseudocode.
- Largely independent from the programming language.
- Exact definition not important.
- Assumed to take a constant amount of time in the RAM model.

Examples:

- Evaluating an expression.
- Assigning a value to a variable.
- Indexing into an array.
- Calling a method.
- Returning from a method.
Counting Primitive Operations

By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size.
Counting Primitive Operations

By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size.

```java
1    /** Returns the maximum value of a nonempty array of numbers. */
2    public static double arrayMax(double[] data) {
3        int n = data.length;
4        double currentMax = data[0];
5        for (int j=1; j < n; j++)
6            if (data[j] > currentMax)
7                currentMax = data[j];
8        return currentMax;
9    }
```
Counting Primitive Operations

By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size.

```java
1 /** Returns the maximum value of a nonempty array of numbers. */
2 public static double arrayMax(double[] data) {
3     int n = data.length;
4     double currentMax = data[0];       // assume first entry is biggest (for now)
5     for (int j=1; j < n; j++)         // consider all other entries
6         if (data[j] > currentMax)       // if data[j] is biggest thus far...
7             currentMax = data[j];       // record it as the current max
8     return currentMax;
9 }
```

Step 3: 2 ops, 4: 2 ops, 5: 2n ops, 6: 2n ops, 7: 0 to n ops, 8: 1 op

Next: how to estimate running time!