# CS 234

Module 2

September 18, 2018

## Case study

Problem: Given a collection of values with possible repeats, a guesser can repeatedly choose values. Correctly guessed values are removed. The process will stop when all values have been found and removed.

### Recipe for user/plan

1. Determine types of data and operations.
2. For each type, choose/customize an ADT.
3. Develop a pseudocode algorithm using ADT operations.
4. Calculate the algorithm's cost with respect to costs of operations.
5. Using information from the provider, choose the best option.

## Why not just use a Python list?

**Keep a clear distinction between planning and coding stages**

- In the planning stage, the goal is to assess options without coding.
- A Python list is a data type specific to Python, not an abstract data type, which can be implemented in any language.

**Learn techniques for any language**

- You need strategies for languages that do not support Python lists.

**Find the ADT that fits best**

- There is a tradeoff between ADTs that support a few operations quickly and those that support more operations more slowly.
- Don't "pay" for operations you don't use.

## ADT Multiset

The ADT Multiset can store multiple copies of data items.

Preconditions: For all M is a multiset and item any data.

Postconditions: Mutation by Add (add one copy of item) and Delete (delete one copy of item if any are present).

| Name | Returns |
|---|---|
| Create() | a new empty multiset |
| IsEmpty(M) | true if empty, else false |
| LookUp(M, item) | true if present, else false |
| Add(M, item) | |
| Delete(M, item) | |

## Choosing and designing ADTs

While you are learning ADTs:

- Case study
- Determine types of data and operations
- Learn about a common ADT

After you have learned about ADTs:

- Choose among common ADTs for a good fit.
- If necessary, customize an ADT by adding or removing operations.
- Or, create an ADT.

## Common ADT operations

- Create an empty ADT
- Determine status of ADT
- Search for an item based on its value
- Search for an item based on its location
- Search for multiple items fitting some criteria
- Add an item
- Add an item in a specific location
- Modify an item with a specific value
- Modify an item in a specific location
- Delete an item selected by value
- Delete an item selected by location
- Rearrange items

**Caution**

Similar names used for consistency, but always check definition of operation.

## Types of ADTs to be considered in the course

Group A: **No order of any kind**
Examples: Multiset, Set

Group B: **Order imposed by operations**
Examples: Stack, Queue, List, Ranking, Grid

Group C: **Items related by structure**
Examples: Unordered Tree, Binary Tree, Ordered Tree, Graph

Group D: **Pairs of values**
Examples: Dictionary, Priority Queue

Note: Groups have been introduced to give you a sense of the relationships among ADTs. You are not responsible for learning groups names or contents.

## Types of data

Although all data is stored as sequences of 0's and 1's, a particular programming language may only allow certain types of operations on particular kinds of data.

**General data** can only be compared for equality.

**Orderable data** can be compared for equality or ordering ($<$, $>$, $\leq$, and $\geq$), e.g. numbers or strings.

**Digital data** supports computations other than comparions for equality or ordering, such as:

- Using the value of the data as an index into a structure
- Decomposing a string into characters
- Applying arithmetic operations to a number

## Pseudocode

**Pseudocode** is a way of expressing an algorithm that can simultaneously:

- be analyzed to give a good approximation of actual costs, and
- be translated into any programming language.

Features to notice:

- Use of functions (e.g. ADT operations) with name and arguments in parentheses
- Branching and looping like in Python

In this course:

- pseudocode you read will adhere to the guidelines in the resource available on the course website, and
- pseudocode you write can be in Python (but you might be doing more work than necessary).

## Pseudocode user/plan for case study

```
myData $\leftarrow$ Create()
while there are still data items
    Add(myData, item)
value $\leftarrow$ guesser's guess
empty $\leftarrow$ IsEmpty(myData)
while value is not "stop" and not empty
    if LookUp(myData, value)
        Delete(myData, value)
    empty $\leftarrow$ IsEmpty(myData)
    if not empty
        value $\leftarrow$ guesser's guess
```

**User/plan step 4**: Calculate the algorithm's cost with respect to costs of operations.

## Lessons about calculating costs

Which box of cookies is the best one to buy? (For 1116 people)

- 1 for 2.25
- 9 for 17.75
- 124 for 59.89

Questions:

- Exact division done?
- Why not cheapest box (2.25)?
- 124-packs expired in 1992.

## How not to do more work than needed

To figure out which house to build from a set of plans, which choice is best:

1. Build all the houses, figure out the costs of each, and then decide.
2. Figure out detailed costs for all plans (each type of floor covering, cabinetry), and then decide.
3. Obtain a rough estimate of the relative costs of the plans and then decide.

## Applying the lessons to CS 234

Lessons learned:

- Calculation without implementation.
- Measure cost per unit.
- Avoid details that aren't needed.
- Cost may not be all that matters

Techniques to use:

- Analyze running time of an algorithm based on pseudocode.
- Express the running time as a function of the input size.
- Categorize functions into rough categories.
- Be aware of limitations of the method.

## Comparing algorithms

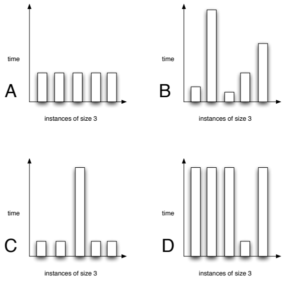
Too simple:

- “good” or “bad”
- How do we compare among the good ones?

Too complex:

- Running time on each possible instance of each possible size.
- How do we compare algorithms where one is better on some and one is better on others?

## Finding a representative for one size of instance



An **instance** is a possible input.

## Analyzing algorithms (SLIDE PLUS - two pages)

Idea: Given a representative for each input size $n$, create a function $f(n)$ to express the **running time** of the algorithm.

**Best case**: The value of $f(k)$ is the fastest running time of the algorithm on any input of size $k$.

# Analyzing algorithms (SLIDE PLUS continued)

**Average case**: The value of $f(k)$ is the sum over all inputs $I$ of size $k$ of the probability of $I$ multiplied by the running time of the algorithm on input $I$.
**Worst case**: The value of $f(k)$ is the slowest running time of the algorithm on any input of size $k$.
When comparing two algorithms, use the same type of running time for both.

## Understanding cases

- Each is a function with a “representative value” chosen for each input size.
- For any input size, the representative value for best case is less than or equal to the representative value for average case which in turn is less than or equal to the representative value for worst case.
- The functions can be equal, if the same representative values are chosen.
- For a specific algorithm and a chosen set of variables, the best/average/worst case should hold for any value of the variables.

## Putting functions into groups

|  |  |
|---|---|
| Constant | $1$, $.5$, $10$, $7.6$, $201$ |
| Logarithmic | $\log_2 n$, $\log_3 n$, $4\log_2 n - 6$ |
| Linear | $n$, $5n$, $n/3$, $4n+2$ |
| Quadratic | $n^2$, $5n^2 - 45n$, $n^2/2 + 6n - 34$ |
| Exponential | $2^n$, $2^n + n^6 + 12$ |

Key ideas:

- Determine group membership without determining details of a function.
- If two functions are in different groups, we have enough information to compare them.
- If functions are in the same group, we need more information to compare them.

## Asymptotic notation a.k.a. order notation

**Asymptotic notation** (or **order notation**) is a way of quickly comparing options without getting mired in insignificant details; **asymptotic** refers to the classification of functions by their behaviour as the size of the input increases towards infinity.

$f(n)$ is in $O(g(n))$ if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for every $n \geq n_0$.

$f(n)$ is in $\Omega(g(n))$ if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq cg(n)$ for every $n \geq n_0$.

$f(n)$ is in $\Theta(g(n))$ if there are real constants $c_1 > 0$ and $c_2 > 0$ and an integer constant $n_0 \geq 1$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for every $n \geq n_0$.

## Understanding asymptotic notation

Using the symbols, functions are groups into big groups based on “simple” functions $g(n)$.

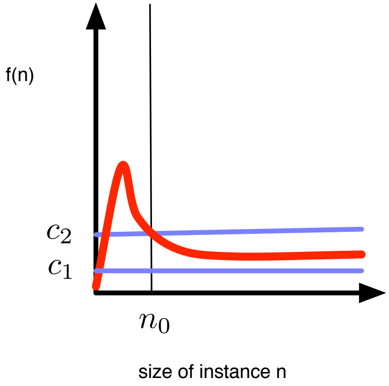
We don’t give the specific $c$ (multiplicative factor) or $n_0$ minimum size of value for comparison.

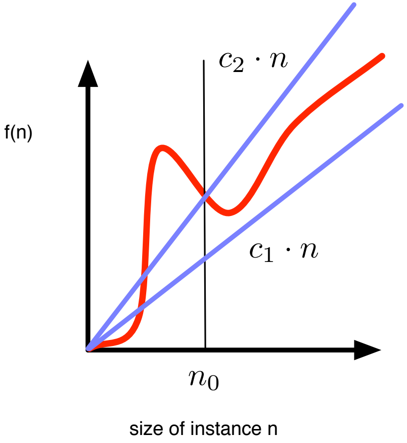
Informally:

- $f(n)$ is in $O(g(n))$ (“Big O”) means there is an upper bound on $f(n)$, based on $g(n)$.
- $f(n)$ is in $\Omega(g(n))$ (“Big Omega”) means there is a lower bound on $f(n)$, based on $g(n)$.
- $f(n)$ is in $\Theta(g(n))$ (“Theta”) means that $f(n)$ is in $O(g(n))$ and that $f(n)$ is also in $\Omega(g(n))$.

## Examples

*SHOW O, then $\Omega$, then $\Theta$. Left is $\Theta(1)$ and right is $\Theta(n)$.*





## Ways to think of asymptotic notation

View $O(g(n))$, $\Omega(g(n))$, and $\Theta(g(n))$ as sets of functions.

- $O$ gives an rough upper bound on a function.
- $\Omega$ give a rough lower bound on a function.
- $\Theta$ means that both $O$ and $\Omega$ hold, forming a "sandwich".

Uses:

- "The running time of the algorithm is in $O(g(n))$."
- "These lines of the algorithm can be executed in $O(g(n))$ time."

Note: Here $g(n)$ is something simple like 1, $\log n$, $n$, not 5, $3\log n + 5$, or $n/4 - \log\log n + 21$.

Note: Each of $O$, $\Omega$, and $\Theta$ can be used to describe any type of function (worst-case, average-case, best-case, or a function that is not a running time).

## Drawings of all the functions in a box

*SHOW lines for* $O(n)$, $\Omega(n)$, $O(\log n)$ *and* $\Omega(\log n)$.

*SHOW where* $\Theta(n)$ *and* $\Theta(\log n)$ *are.*

Mention various functions, ask where to put them.

- $n/4+5$
- $3n^2$
- $6\log n+12$
- $4n-2$
- $6$
- $2^n$
- $\log\log n$

## Tricky questions on asymptotic notation

Suppose we have algorithms $A$ and $B$ with worst-case running times $f(n)$ and $g(n)$.

- If we know that $f(n)$ is in $O(n)$ and $g(n)$ is in $O(2^n)$, do we know that $A$ is a better choice than $B$?
- If we know that $f(n)$ is in $\Omega(n)$ and $g(n)$ is in $\Omega(2^n)$, we know that $A$ is a better choice than $B$?
- If we know that $f(n)$ is in $\Theta(n)$ and $g(n)$ is in $\Theta(2^n)$, we know that $A$ is a better choice than $B$?
- If we know that $f(n)$ is in $\Theta(n)$ and $g(n)$ is in $\Theta(n)$, are they equally good choices?

## Analyzing pseudocode

Examples of $O(1)$-time operations:

- Assign or use a variable
- An arithmetic operation
- Most built-in operations on numbers and strings

Code with simple calculations:

- Sequential statements/blocks: sum (maximum cost for one variable)
- Branching: maximum cost of any branch
- Looping: sum of cost of each iteration

Code that requires lookup/calculation:

- Use of a user-defined function
- Operations on a list or other complex data type

## Example calculation for sequential blocks

Block X has worst-case cost in $\Theta(f(n))$
$c_{f_1} f(n) \leq$ cost of X $\leq c_{f_2} f(n)$ for $n \geq n_f$

Block Y has worst-case cost in $\Theta(g(n))$
$c_{g_1} g(n) \leq$ cost of Y $\leq c_{g_2} g(n)$ for $n \geq n_g$

Block Z has worst-case cost in $\Theta(h(n))$
$c_{h_1} h(n) \leq$ cost of Z $\leq c_{h_2} h(n)$ for $n \geq n_h$

To show: total cost is in $\Theta(M(n))$ where

- $M(n) = \max\{f(n), g(n), h(n)\}$
- $N = \max\{n_f, n_g, n_h\}$
- $C_1 = \min\{c_{f_1}, c_{g_1}, c_{h_1}\}$ (lower bound)
- $C_2 = c_{f_2} + c_{g_2} + c_{h_2}$ (upper bound)

## Pseudocode user/plan for case study

```
1   myData ← Create()
2   while there are still data items
3       Add(myData, item)
4   value ← guesser's guess
5   empty ← IsEmpty(myData)
6   while value is not "stop" and not empty
7       if LookUp(myData, value)
8           Delete(myData, value)
9       empty ← IsEmpty(myData)
10      if not empty
11          value ← guesser's guess
```

For $k$ the number of items stored:
Create costs $C(k)$
IsEmpty costs $E(k)$
LookUp costs $L(k)$
Add costs $A(k)$
Delete costs $D(k)$

## Analysis of parts of the example

Sequential statements/blocks:

- Line 1 is the sum of $\Theta(1)$ operations and $C(k)$, or $\Theta(C(k))$ in total.
- Lines 4 and 5 together are $\Theta(E(k))$.
- Lines 2-3 and 6-11 require understanding analysis of loops.
- Total will be maximum cost of any block.

## Analyzing loops

Situations in this course:

1. Cost of the loop body is the same for each iteration.
2. Cost of the loop body is in $\Theta(i)$ for iteration $i$.

Note: In general, analysis can be much more complex.

Situation 1: Cost is the product of the number of iterations and the cost of each iteration.

Situation 2: Cost is in $\Theta(n^2)$ for $n$ the number of iterations.

Intuition: Arithmetic series $\sum_{i=1}^{n} i = n(n+1)/2$.

## Nested loops

```
sum $\leftarrow$ 0
for i from 1 to n
    for j from 1 to i
        sum $\leftarrow$ sum $+ (i-j)^3$
return sum
```

## Simplifying expressions

Example given, but not proof in general: $n^3/2 - 2n^2 + 3 \in \Theta(n^3)$.

Part 1: $n^3/2 - 2n^2 + 3 \in O(n^3)$.
$n^3/2 - 2n^2 + 3 \leq n^3/2 + 3$ and since $n^3/2 \geq 3$ for $n \geq 2$, $n^3/2 + 3 \geq n^3$.
We have chosen $c = 1$ and $n_0 = 2$.

Part 2: $n^3/2 - 2n^2 + 3 \in \Omega(n^3)$.
$n^3/2 - 2n^2 + 3 \geq n^3/2 - 2n^2$
We want to choose a big enough $n_0$ so that $n^3/2 - 2n^2$ is bounded below by some fraction times $n^3$. Each $n^2$ should be no bigger than $n^3/4$. Let's aim to bound them by $1/6$.
For $n \geq 7$, $n^2 < n^3/6$, so $n^3/2 - 2n^2 \geq n^3/2 - 2n^3/6 = n^3/6$.
We have chosen $c = 1/6$ and $n = 7$.

## Comparing order notation on multiple variables

- $f(n,m) = n^2 + 5n + 3m + 1$ is in $\Theta(n^2 + m)$
- $g(n,m) = 5mn + m - n + 3$ is in $\Theta(mn)$
- $h(n,m) = n \log n + 5n + 3m^2 + m \log m$ is in $\Theta(n \log n + m^2)$

### Using multiple variables

- Using knowledge about relative values of variables is OK.
- Making assumptions about relative values is NOT OK.
- For best-case or worst-case running time, keep all variables as variables.

Examples:

- If $m \in \Theta(n^3)$, then $f(n,m)$ is in $\Theta(n^3)$.
- If $m \in \Theta(n)$, then $f(n,m)$ is in $\Theta(n^2)$.

## Case study example analyzed using two variables

Define

- $n$ is the number of inputs in the set
- $m$ is the number of guesses

Number of iterations of the loop in lines 6-11 is a function of the number of guesses.
Suppose $L(k)$ is in $\Theta(k)$, all others in $\Theta(1)$.
Then determine total as a function of n and m.

- Lines 1, 4, and 5 are $\Theta(1)$
- Lines 2-3 are $\Theta(n)$
- Lines 6-11, for worst case all guesses are bad ones so the ADT is full and cost of LookUp is at max (all else constant), with cost of $\Theta(n)$ for the loop body, and total cost $\Theta(mn)$ for this loop and the entire algorithm.

## How to compare algorithms

Compare the same types of functions (worst-case, average-case, or best-case).

Do not set values of variables.

If functions are very different (e.g. linear vs. constant):

- No need to determine exact functions.
- Save work by getting rough notion of functions.

If functions are very close (e.g. both linear):

- It is not enough to use rough notion of functions.
- Both linear does not mean neither is better.
- More precise counting is necessary.

Note: The term **complexity** is used to refer to running time.