CS 234: Data Types and Structures

Naomi Nishimura

Module 2

Date of this version: September 17, 2019

WARNING: Drafts of slides are made available prior to lecture for your convenience. After lecture, slides will be updated to reflect material taught. Check the date on this page to make sure you have the correct, updated version.

WARNING: Slides do not include all class material; if you have missed a lecture, make sure to find out from a classmate what material was presented verbally or on the board.

Case study

You and a friend are going on a bird-watching expedition. In a not-very-fair division of labour, you are going to do all the bird-watching and your friend is going to keep track of the birds you have seen.

To be able to collaborate, you wish to decide on an abstract data type.

Recipe for user/plan

1. Determine types of data and operations.
2. For each type, choose/modify/create 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.

User/plan recipe step 2

Step 2: For each type, choose/modify/create an ADT.

Choosing among ADTs:

- More general ADTs (useful in more situations due to more operations) tend to have higher costs for each operation.
- Choose an ADT that provides all the operations needed but not too much more.

Key idea: Use the simplest ADT that can get the job done.

Modifying an ADT:

- **Augment** a known ADT by adding one or more new operations.
- **Restrict** a known ADT by removing one or more operations.

Common types of augmentation and restriction

Augmentation:

- Allow data items to be formed of **compound data**.
- Allow operations to access or set just one part (or **field**) of a data item.

Restriction:

- In the case of **static data**, no operation can change which data items are stored. In such a situation, there is typically an operation that creates an ADT from provided data items (instead of creating an empty ADT and adding to it).

Why not just use a Python list?

- In the planning stage, we wish to assess options without coding.
- We want a solution that can be implemented in any language.
- A Python list is a data type specific to Python, not an ADT.
- You need strategies for languages that do not support Python lists.
- Python lists support many operations that we do not need (and for which we shouldn't "pay").

ADT Multiset

The ADT Multiset can store multiple copies of data items.

Preconditions: For *ADD*, *M* is a multiset and *Data* is any data item.

Postconditions: Mutation by *ADD* (adds one copy of *Data*).

Name	Returns
CREATE()	a new empty multiset
ADD(M, Data)	

Case study, part 2

You just noticed that what you had identified as a hawk was really a decal put on a window to discourage other birds from flying into it.

Options to consider:

- Write an algorithm using existing ADT operations.
- Augment the ADT by adding a new operation.

ADT Multiset

The ADT Multiset can store multiple copies of data items.

Preconditions: For all *M* is a multiset and *Data* is any data item; for *DELETE Data* must be in *M*.

Postconditions: Mutation by *ADD* (adds one copy of *Data*) and *DELETE* (deletes one copy of *Data*).

Name	Returns
CREATE()	a new empty multiset
IS_IN(M, Data)	*True* if present, else *False*
ADD(M, Data)	
DELETE(M, Data)	

Case study, part 3

You have gone bird-watching, and have filled the ADT with information about what you saw.

It's not very interesting data, as you stopped by a fast-food restaurant for lunch and saw lots of the same type of cooked bird.

To make your data a little less appetizing, you'd like to fix up the data by replacing each instance of "Chicken nugget" by "Chicken".

Options to consider:

- Write an algorithm using existing ADT operations.
- Augment the ADT by adding a new operation.

Common ADT operations

- Create an empty ADT
- Determine status of the ADT
- Search for an item based on its value
- Search for an item based on its position
- Search for multiple items fitting some criteria
- Add an item
- Add an item in a specific position
- Modify an item with a specific value
- Modify an item in a specific position
- Delete an item selected by value
- Delete an item selected by position
- 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 positions**
Examples: Multiset, Set

Group B: **Positions imposed by operations**
Examples: Stack, Queue, Indexed Sequence, Ranking, Grid

Group C: **Data items related by structure**
Examples: Binary Tree, Ordered Tree, Unordered Tree, Undirected Graph, Directed Graph

Group D: **Data items are 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 group names or which ADT belongs in each group.

User/plan recipe step 3

Step 3: Develop a pseudocode algorithm using ADT operations.

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:

- Function applications are written as the name of the function followed by the names of arguments in parentheses.
- Branching and looping are like in Python.
- A preamble states the purpose and types of data.

In this course

- pseudocode you read will adhere to the guidelines in the mini-textbook, and
- pseudocode you write can be in Python (but you might be doing more work than necessary).

Pseudocode user/plan for case study

FIX_NUGGETS(Birds)
INPUT: *A multiset Birds*
OUTPUT: *Updated Birds*
EFFECTS: *In Birds each instance of "Chicken nugget" is replaced by "Chicken"*

```
1  while IS_IN(Birds, "Chicken nugget")
2      DELETE(Birds, "Chicken nugget")
3      ADD(Birds, "Chicken")
4  return Birds
```

User/plan recipe step 4

Step 4: Calculate the algorithm's cost with respect to costs of operations.

Goals:

- Determine costs before coding various options
- Obtain results that predict relative costs of coding various options

Goals and techniques

Goals:

- Calculation without building (coding)
- Measure cost per unit
- Avoid details that aren't needed
- Cost may not be all that matters

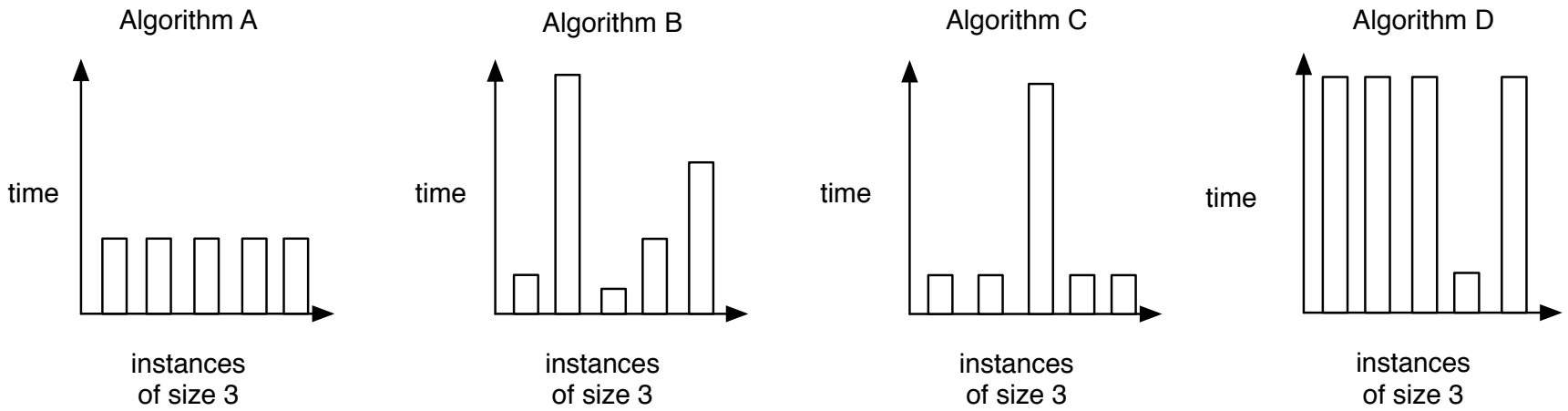
Techniques to use:

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

Finding a representative for one size of instance

An **instance** is a possible input.

We can compare four algorithms (A, B, C, and D) by comparing their running times on all the instances of a particular size (here, size 3).



Analyzing algorithms

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 $f(n)$ is the fastest running time of the algorithm on any input of size n.

Average case: The value of $f(n)$ is the sum over all inputs I of size n of the probability of I multiplied by the running time of the algorithm on input I.

Worst case: The value of $f(n)$ is the slowest running time of the algorithm on any input of size n.

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.

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:

- We don't care about multiplicative factors or additive terms.
- We do care about what happens as n goes to infinity.

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 $\Theta(g(n))$ 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 $n \geq n_0$.

- Each $g(n)$ is a "simple" function.
- A "simple" function has no multiplicative factors or additive terms.
- For example, $g(n) = 1$ for a constant function and $g(n) = n$ for a linear function.

More types of order notation

$f(n)$ 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 $n \geq n_0$.

Note: There may be many choices of constants c and n_0; for the definitions to hold, it suffices that some exist.

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.
- Ω give a rough lower bound on a function.
- Θ means that both O and Ω hold, forming a "sandwich".

Uses:

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

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

Exercises on asymptotic notation

Suppose we have two functions $f(n)$ and $g(n)$ and we want to know which is "better" (that is, asymptotically smaller).

- If $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$, do we know that $g(n)$ is better than $f(n)$?
- If $f(n) \in O(n^2)$ and $g(n) \in O(n)$, do we know that $g(n)$ is better than $f(n)$?
- If $f(n) \in \Omega(n^2)$ and $g(n) \in \Omega(n)$, do we know that $g(n)$ is better than $f(n)$?
- If $f(n) \in \Omega(n^2)$ and $g(n) \in O(n)$, do we know that $g(n)$ is better than $f(n)$?
- If $f(n) \in \Theta(n)$ and $g(n) \in \Theta(n)$, do we know that $f(n) = g(n)$?

Analyzing pseudocode

Examples of $O(1)$-time operations (see mini-textbook for more info):

- Assign or use a variable
- Use a simple arithmetic or Boolean operation
- Return a value from a function call

Calculating costs of pseudocode:

- Sequential statements/blocks: Sum of the costs of the blocks
- Branching: Maximum cost of any branch
- Looping: Sum of the costs of each iteration

Handling expressions

You can use the following ideas when handling expressions:

- You can multiply $\Theta(f(n))$ and $\Theta(g(n))$ to obtain $\Theta(f(n)g(n))$ and add $\Theta(f(n))$ and $\Theta(g(n))$ to obtain $\Theta(f(n)+g(n))$.
- For a function $f(n)$, you can conclude that $f(n)$ is in $\Theta(g(n))$ for $c \cdot g(n)$ the dominant term in $f(n)$, where c is a constant.
- When forming an upper bound on an expression, you can add a term to the expression, replace a term by a bigger term, or multiply the expression by a number greater than 1.
- When forming a lower bound on an expression, you can subtract a term from the expression, replace a term by a smaller term, or divide the expression by a number greater than 1.

Using the second idea in calculating the cost of sequential blocks, all which are a function of n, you can take the maximum of the costs (details on the next slide).

Example calculation for sequential blocks

Block X has worst-case cost in $\Theta(f(n))$
$x_1 f(n) \leq$ cost of X $\leq x_2 f(n)$ for $n \geq n_f$

Block Y has worst-case cost in $\Theta(g(n))$
$y_1 g(n) \leq$ cost of Y $\leq y_2 g(n)$ for $n \geq n_g$

Block Z has worst-case cost in $\Theta(h(n))$
$z_1 h(n) \leq$ cost of Z $\leq z_2 h(n)$ for $n \geq n_h$

To show: Total cost is in $\Theta(M(n))$ where $M(n) = \max\{f(n), g(n), h(n)\}$.

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

```
NESTED(N)
INPUT:      A positive integer N
OUTPUT:     A value computed using N
1  Sum ← 0
2  for Primary from 1 to N
3      for Secondary from 1 to Primary
4          Sum ← Sum + (Primary − Secondary)^3
5  return Sum
```

Comparing order notation on multiple variables

- $f(n,m) = 3n^2 + n/2 + m/4 + 1$ is in $\Theta(n^2 + m)$
- $g(n,m) = m^2n/5 + mn^2 - 5m + 3n$ is in $\Theta(m^2n + mn^2)$

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 $\Theta(n^2)$.

User/plan recipe step 4

Step 4: Calculate the algorithm's cost with respect to costs of operations.

Define

- n to be the total number of items in the multiset
- m to be the total number of times "Chicken nugget" appears

Use k for the number of items in the multiset at the time of the operation; for this example we will simplify the analysis by using n throughout.

Is_In costs $I(k)$

Add costs $A(k)$
Delete costs $D(k)$

Pseudocode user/plan for case study

FIX_NUGGETS(Birds)
INPUT: *A multiset Birds*
OUTPUT: *Updated Birds*
EFFECTS: *In Birds each instance of "Chicken nugget" is replaced by "Chicken"*

```
1  while IS_IN(Birds, "Chicken nugget")
2      DELETE(Birds, "Chicken nugget")
3      ADD(Birds, "Chicken")
4  return Birds
```

Analysis of pseudocode

- Sequential blocks: Lines 1-3 and line 4.
- Cost of 4 is in $\Theta(1)$ (use a variable + return a value from a function call).
- Cost of lines 1-3 can be determined by the number of iterations of the loop and the cost of the body of the loop.
- The number of iterations is the number of times that "Chicken nugget" appears, which is m.
- In the body of the loop, the total cost of lines 1 through 3 is the sum of $\Theta(I(n))$ (line 1), $\Theta(D(n))$ (line 2), and $\Theta(A(n))$ (line 3). The total is the maximum of the three, or $\Theta(M(n))$ were $M(n) = \max\{I(n), D(n), A(n)\}$.
- The total cost of lines 1-3 is in $\Theta(mM(n))$.
- The total cost of the function is dominated by lines 1-3.

Calculating the cost for specific costs of operations

Suppose we have the following costs:

- $I(n) \in \Theta(n)$
- $A(n) \in \Theta(1)$
- $D(n) \in \Theta(n)$

Then the total cost is $\Theta(mM(n))$ where $M(n) = \max\{I(n), D(n), A(n)\}$, or $\Theta(mn)$.

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 determining rough categories.

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

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

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

Other resources

In addition to time, implementations may be compared due to:

- amount of space used, or
- amount of randomness used (for randomized algorithms).

In this course we will mostly focus on time.

Module summary

Topics covered:

- Case study: Bird watching
- Augmentation and restriction
- ADT Multiset
- Common ADT operations
- Groups of ADTs
- Pseudocode
- Running time (best case, average case, worst case)
- Asymptotic notation
- Analyzing pseudocode
- Order notation on multiple variables
- Comparing algorithms