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.
ADT Multiset

Data: Any type.

Operations:
- CreateMultiset()
- IsEmptyMultiset(M)
- LookUpMultiset(M, item)
- AddToMultiset(M, item)
- DeleteFromMultiset(M, item)
Choosing and designing ADTs

Recipe for choosing an ADT
- Determine data used.
- Determine operations needed.
- Does a common ADT suffice?
- Does a variant on a common ADT suffice?
- If not, create an ADT.

Recipe for designing a new ADT
- Specify data used. Should it be generalized?
- Specify operations. Can some be implemented using others?
Common ADT operations

- Create an empty ADT
- Determine status of ADT (e.g. empty or not, number of items stored)
- 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
Types of ADTs

- **Form of the data** (e.g. single item, pair of items)
- **Type of the data** (e.g. any, orderable, digital - to be discussed soon)
- **Information relating data** (e.g. two vertices form an edge)
- **Positioning of the data** (e.g. placing items in specified positions)
Types of ADTs to be considered in the course

Categorized by form/type/information/positioning:

**No order of any kind**
Group A: single items/any/none/none
(Multiset, Set)

**Order imposed by operations**
Group B: single items/any/none/by order of insertion or specification of position
(Stack, Queue, List, Ranking, Superlist, Grid)

**Items related by structure**
Group C: vertices, edges, other data/any/edges are pairs of vertices/none
(Unordered Tree, Binary Tree, Ordered Tree, Graph)

**Pairs of values**
Group D: pairs/(x, any)/none/none
(Dictionary, Priority Queue)
Types 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 comparisons 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
Developing an algorithm

**User/plan step 3**: Develop pseudocode algorithm using ADT operations. **Pseudocode** is a way of expressing an algorithm that simultaneously:

- Can be analyzed to give a good approximation of actual costs
- Can 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

See handout for more details.
Pseudocode user/plan for case study

1. myData ← CreateMultiset()
2. while there are still data items
3.     AddToMultiset(myData, item)
4. value ← guesser’s guess
5. empty ← IsEmptyMultiset(myData)
6. while value is not ”stop” and not empty
7.     if LookUpMultiset(myData, value)
8.         DeleteFromMultiset(myData, value)
9.     empty ← IsEmptyMultiset(myData)
10. if not empty
11.     value ← guesser’s guess

**User/plan step 4:** Calculate cost of algorithm with respect to costs of operations.
Comparing running times of two algorithms
Finding a representative for one size of instance

A

B

C

D

instances of size 3

instances of size 3

instances of size 3

instances of 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 \( f(k) \) is the fastest running time of the algorithm on any input of size \( k \).

**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 \( k \).
## Comparison of algorithms

<table>
<thead>
<tr>
<th>Problem</th>
<th>Approach</th>
<th>Solution</th>
</tr>
</thead>
</table>
| • Too many criteria  
• Too much work  
• Too many numbers  
• Too many functions | • Focus on what we can control  
• Calculate without implementation  
• Measure per unit  
• Obtain guaranteed behaviour | • Use running time  
• Use pseudocode  
• Use $f(n)$ for size $n$  
• Use worst case |
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:

- Use pseudocode to place worst-case running time in a group.
- Details of function not needed to determine group membership.
- If functions are in different groups, we have enough information.
- If functions are in the same group, we need more information.
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.

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

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 $\Omega(g(n))$. 
Ways to think of asymptotic notation

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

- $O$ gives an upper bound on a function.
- $\Omega$ gives a 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 $O(f(n))$.”
- “These lines of the algorithm can be executed in $O(f(n))$ time.”

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

Note: Any notation can be used for worst case, average case, or best case.
Definitions of 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 $n \geq n_0$.

$f(n)$ 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 $n \geq n_0$. 
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 A has worst-case cost in $\Theta(f(n))$
$c_{f_1} f(n) \leq \text{cost of A} \leq c_{f_2} f(n)$ for $n \geq n_f$

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

Block C has worst-case cost in $\Theta(h(n))$
$c_{h_1} h(n) \leq \text{cost of C} \leq c_{h_2} h(n)$ for $n \geq n_h$
Pseudocode user/plan for case study

1. myData $\leftarrow$ CreateMultiset()
2. while there are still data items
   3. AddToMultiset(myData, item)
4. value $\leftarrow$ guesser’s guess
5. empty $\leftarrow$ IsEmptyMultiset(myData)
6. while value is not ”stop” and not empty
   7. if LookUpMultiset(myData, value)
      8. DeleteFromMultiset(myData, value)
   9. empty $\leftarrow$ IsEmptyMultiset(myData)
10. if not empty
11. value $\leftarrow$ guesser’s guess

For $k$ the number of items stored:
- CreateMultiset costs $C(k)$
- LookUpMultiset costs $L(k)$
- AddToMultiset costs $A(k)$
- DeleteFromMultiset costs $D(k)$
- IsEmptyMultiset costs $E(k)$
Techniques for more precise bounds

Finding a lower bound:
- Drop some terms.
- Replace some terms with smaller numbers.

Finding an upper bound:
- Add some terms
- Replace some terms with larger numbers
Nested loops

1. `sum ← 0`
2. `for i from 1 to n`
3. `    for j from 1 to i`
4. `    sum ← sum + (i − j)^3`
5. `return sum`
Conventions for expressing a function in order notation

- Replace each multiplicative constant by 1 or $-1$.
- Remove all but the largest term.
- Use base 2 for any constant-base logarithm.

$$2^n/4 - n^2/5 + 20n \log n + 6$$

$$2^n - n^2/5 + 20n \log n + 6$$

$$2^n - n^2 + 20n \log n + 6$$

$$2^n - n^2 + n \log n + 6$$

$$2^n - n^2 + n \log n + 1$$

$\Theta(2^n)$ (or $O(2^n)$ or $\Omega(2^n)$)
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.

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 \( \Theta(n^2) \).
Pseudocode user/plan for case study

1. myData ← CreateMultiset()
2. while there are still data items
3.     AddToMultiset(myData, item)
4. value ← guesser’s guess
5. empty ← isEmptyMultiset(M)
6. while value is not ”stop” and not empty
7.     if LookUpMultiset(M, value)
8.         DeleteFromMultiset(M, value)
9.     empty ← isEmptyMultiset(M)
10. if not empty
11.     value ← guesser’s guess

For $k$ the number of items stored:
- CreateMultiset costs $C(k)$
- LookUpMultiset costs $L(k)$
- AddToMultiset costs $A(k)$
- DeleteFromMultiset costs $D(k)$
- isEmptyMultiset costs $E(k)$
How to compare algorithms

Note: there may not be a single best algorithm for solving a problem. Compare using same measure; use the worst case of A versus the worst case of B NOT the worst case of A versus the best case of B.

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.
Bounds on algorithms versus problems

In this course we discuss upper and lower bounds on the running times of algorithms.

**Note**

This slide is not relevant to this course, but may be useful in using these terms in future courses.

**CAUTION:** A bound on worst-case running time of an algorithm is not equal to a bound on the worst-case complexity of a problem.

A lower bound of $\Omega(f(n))$ on the worst-case complexity of a problem means that every algorithm for the problem uses $\Omega(f(n))$ time in the worst case.

An upper bound of $O(f(n))$ on the worst-case complexity of a problem means that there exists an algorithm with worst-case running time in $O(f(n))$. 