Why are Algorithms Important?

If we can discover an algorithm to perform a task, we can instruct a computing agent to execute it and solve the problem for us.

Representing Algorithms

- What language to use?
  - Expressive.
  - Clear, precise, and unambiguous.
- For example, we could use:
  - Natural language (e.g. English).
  - Formal programming languages (e.g. C++).
  - Something else?
Example: Adding 2 numbers

• Assume we know how to add 2 single digit numbers, but want to write an algorithm to add any 2 numbers:

\[
\begin{array}{c}
1 \\
1 8 2 \\
+ \\
2 6 3 \\
\hline \\
4 4 5
\end{array}
\]

Example using Natural Language

Initially, set the value of the variable carry to 0. When these initializations have been completed, begin looping until the value of the variable i becomes greater than m-1. First add together the values of the two digits \(a_i\) and \(b_i\) and the current value of the carry digit to get the result called \(c_i\). Now check the value of \(c_i\) to see whether it is greater than or equal to 10. If \(c_i\) is greater than or equal to 10, then ...

Natural Languages

• English or some other natural language.
• Are not particularly good:
  – too verbose
  – unstructured
  – too rich in interpretation (ambiguous)
  – imprecise
Example using Programming Language

```
{   int I, m, Carry;   int a[100], b[100], c[100];   cin >> m;   for ( int j = 0 ; k <= m-1 ; j++ ) {       cin >> a[j];       cin >> b[j];   }   Carry = 0;   i = 0;   while ( i < m ) { ... }
```

Programming Languages

• Are not particularly good either:
  – Too many implementation details to worry about
  – Too rigid syntax
• Easy to lose sight of the real task
  – We don't see the forest because of all the trees!

Pseudo-code

• We need a compromise between the two:
  ➔ Pseudo-code
• Computer scientists use pseudo-code to express algorithms:
  – English like constructs (or other natural language), but
  – modeled to look like statements in typical programming languages.
Pseudo-code for the Addition Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the value of $a_{m-1}, \ldots, a_0$</td>
</tr>
<tr>
<td>2</td>
<td>Set the value of $b_{m-1}, \ldots, b_0$</td>
</tr>
<tr>
<td>3</td>
<td>Set the value of carry to 0</td>
</tr>
<tr>
<td>4</td>
<td>Set the value of $i$ to 0</td>
</tr>
<tr>
<td>5</td>
<td>Repeat steps 6-8 until $i$ greater than $m-1$</td>
</tr>
<tr>
<td>6</td>
<td>Set the value of $c_i$ to $a_i + b_i + \text{carry}$</td>
</tr>
<tr>
<td>7</td>
<td>If $c_i \geq 10$, then set $c_i$ to $c_i - 10$ and carry to 1; otherwise set the value of carry to 0</td>
</tr>
<tr>
<td>8</td>
<td>Set value of $i$ to $i + 1$ (look at next digit)</td>
</tr>
<tr>
<td>9</td>
<td>Set $c_m$ to carry</td>
</tr>
<tr>
<td>10</td>
<td>Print out the final answer $c_m, c_{m-1}, \ldots, c_0$</td>
</tr>
<tr>
<td>11</td>
<td>Stop</td>
</tr>
</tbody>
</table>

What kind of operations do we need?

- Getting input and producing output
  - Get the two numbers
  - Display the outcome
- Referring to values within our algorithm
  - Add together the rightmost digits of the two numbers
  - Add together $a_0$ and $b_0$
- Doing something if some condition is true
  - If the outcome is greater or equal to 10 then ...
- Doing something repeatedly
  - Do this for all the digits in the numbers ...

Pseudo-code Primitives

Three basic kind of operations:

- Sequential
  - Computation (Set ...)
  - Input/Output (Get ... / Print ...)
- Conditional
  - If ...
  - Else
  - If ...
- Iterative / looping
  - Repeat ...
  - While ...
### Computation

General format:

```
Set the value of <variable> to <expression>
```

Performs a computation and stores the result.

**Example:**

- Set the value of \( C \) to \((A + B)\)
- Set the value of \( location \) to 0
- Set the value of \( GPA \) to \((\text{sum} / \text{count})\)

---

### Variables

A variable is a named storage.

- A value can be stored into it, overwriting the previous value
- Its value can be copied

**Examples:**

- Set the value of \( A \) to 3
  - The variable \( A \) holds the value 3 after its execution
- Set the value of \( A \) to \((A+1)\)
  - Same as: add 1 to the value of \( A \) (\( A \) is now 4)

---

### Not too Strict on Syntax

- Pseudo-code is kind of a programming language without a rigid syntax, for example we can write:
  - Set the value of \( A \) to \((B+C)\)
  - as
    - Set \( A \) to \((B+C)\)
  - Or even:
    - Set the value of \( \text{sum} \) to 0
    - Set the value of \( \text{GPA} \) to 0
  - as
    - Set \( \text{sum} \) and \( \text{GPA} \) to 0
• The computing agent (computer) needs to communicate with the outside world:
  – INPUT operations allow the computing agent to receive from the outside world data values to use in subsequent computations.
  – OUTPUT operations allow the computing agent to communicate results of computations to the outside world.

Sequential Operations - Input/Output

Outside world

Input

Output

Input

General format:

Get a value for <variable>

The computing agent (computer) suspends executions and waits for an input value.

Input - Examples

• Examples:
  – Get value for grade
  – Get values for N, M
• Can write:
  – Get value for N₁
  – ...
  – Get value for N₁₀₀
• as
  – Get value for N₁, ..., N₁₀₀
Output

General format:

- Print the value of `<variable>`
- Print the message, "<text>"

The computing agent (computer) displays the value of the variable(s).

Output - Examples

- Examples:
  - Print the value of `grade`
  - Print the message, "Hello"
- Can write:
  - Print the value of `N1`
  - ...
  - Print the value of `N100`
- as
  - Print the values of `N1, ..., N100`

Example

- Write an algorithm to calculate the average of three numbers.

<table>
<thead>
<tr>
<th>Steps</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for <code>N1</code>, <code>N2</code>, and <code>N3</code></td>
</tr>
<tr>
<td>2</td>
<td>Set the value of <code>Average</code> to <code>(N1+N2+N3)/3</code></td>
</tr>
<tr>
<td>3</td>
<td>Print the value of <code>Average</code></td>
</tr>
<tr>
<td>4</td>
<td>Stop</td>
</tr>
</tbody>
</table>
Conditional Operations

If \(<\text{condition}>\) then
  operations for the then-part
Else
  operations for the else-part

1. Evaluate \(<\text{condition}>\) expression to see whether it is true or false.
2. If true, then execute operations in then-part
3. Otherwise, execute operations in else-part.

Conditions, or Boolean Expressions

• A condition is one whose value is true or false, for example:
  - \(3 > 2\) is greater than (true)
  - \(3 = 2\) is equal to (false)
  - \(A > 2\) is true if A's value is greater than 2 (at the time this is executed), false otherwise.

Conditions may be compounded

E1 or E2
true if at least one of them is true; false otherwise.
E.g. \(3 > 2\) or \(2 > 3\) is true

E1 and E2
true if both are true; false otherwise.
E.g. \(3 > 2\) and \(2 > 3\) is false

not E
true if E is false, false if E is true
**Example**

1. Get a value for $A$
2. If $A = 0$ then
3. Print the message, “The input is zero”
   Else
4. Print the message, “The input is not zero”

Example

1. Get a value for grade
2. If grade $< 1$ or grade $> 9$ then
3. Print the message, “Invalid grade”
   Else
4. Set the value of total to (grade + total)

**Iterative Operations - Repeat**

Repeat steps i to j until <condition> becomes true

- step i: operation
- step i+1: operation
- ...
- step j: operation

1. Execute steps i to j
2. Evaluate <condition>
3. If condition is false, go back to 1.
4. Otherwise, continue execution from step j+1.

Example

1. Get a value for count
2. Repeat steps 3 to 5 until (count >10)
3. Set square to (count * count)
4. Print the values of count and square
5. Add 1 to count
Repeat Loops

*What happens when it gets executed?*

If initial value for `count` is 8, we get printout

8   64
9   81
10  100

Repeat Loops

If initial value for `count` is 11, we get printout

11  121

Why?

Because the body is executed once before any test is done!

If need to execute loop 0 or more times we should use While-loops.

Iterative Operation - While

While `<condition>` remains true do steps i to j

1. Evaluate `<condition>`
2. If condition is true, execute steps i to j, then go back to 1.
3. Otherwise, if condition is false, continue execution from step j+1.
**Example**

1. Get a value for `count`
2. While `count` < 10 do
3.   Set `square` to `(count * count)`
4.   Print the values of `count` and `square`
5.   Add 1 to `count`
6. Stop

**While Loops**

*What happens when it gets executed?*

If `count` starts with 7, we get printout

- 7 49
- 8 64
- 9 81

What if `count` starts with 11?

Nothing is printed, loop is executed 0 times.

**Example: Multiply (via addition)**

<table>
<thead>
<tr>
<th>Steps</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for <code>N</code> and <code>M</code></td>
</tr>
<tr>
<td>2</td>
<td>Set the value of <code>Result</code> to 0</td>
</tr>
<tr>
<td>3</td>
<td>While <code>M</code> &gt; 0 do steps 4 and 5</td>
</tr>
<tr>
<td>4</td>
<td>Add <code>N</code> to the value of <code>Result</code></td>
</tr>
<tr>
<td>5</td>
<td>Subtract 1 from <code>M</code></td>
</tr>
<tr>
<td>6</td>
<td>Print the value of <code>Result</code></td>
</tr>
<tr>
<td>7</td>
<td>Stop</td>
</tr>
</tbody>
</table>
N * M Example
Suppose initially $N = 3$ and $M = 4$. During computation, the variable Result held the following values, in that order:

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

Infinite Loops
Danger:
A loop can be infinite due to non-changing conditions.

Example 1: Repeat until $2 > 3$
- Loop body
  - $2 > 3$ is false all the time.

Example 2: While $3 > 2$
- Loop body
  - $3 > 2$ true all the time.

Why do these two algorithms not terminate?

1. Set the value of $i$ to 1
2. While $i < 10$ do step 3
3. Print value of $i$
4. Stop

1. Set the value of $A$ to 1
2. While $A$ is an odd number do
3. Add 2 to the value of $A$
4. Print the value of $A$
5. Stop
**Addition Algorithm Revisited**

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the value of ( a_{m-1}, \ldots, a_0 )</td>
</tr>
<tr>
<td>2</td>
<td>Set the value of ( b_{m-1}, \ldots, b_0 )</td>
</tr>
<tr>
<td>3</td>
<td>Set the value of carry to 0</td>
</tr>
<tr>
<td>4</td>
<td>Set the value of ( i ) to 0</td>
</tr>
<tr>
<td>5</td>
<td>Repeat steps 6-8 until ( i ) greater than ( m-1 )</td>
</tr>
<tr>
<td>6</td>
<td>Set the value of ( c_i ) to ( a_i + b_i + \text{carry} )</td>
</tr>
<tr>
<td>7</td>
<td>If ( c_i \geq 10 ), then set ( c_i ) to ( c_i - 10 ) and carry to 1; otherwise set the value of carry to 0</td>
</tr>
<tr>
<td>8</td>
<td>Set value of ( i ) to ( i + 1 ) (look at next digit)</td>
</tr>
<tr>
<td>9</td>
<td>Set ( c_m ) to carry</td>
</tr>
<tr>
<td>10</td>
<td>Print out the final answer ( c_m, c_{m-1}, \ldots, c_0 )</td>
</tr>
<tr>
<td>11</td>
<td>Stop.</td>
</tr>
</tbody>
</table>

**Summary of Pseudocode**

**Sequential**
- Set the value of variable to expression

**Input and Output**
- Get a value ……; Print ……

**Conditional**
- If a condition is true then  
  - the first set of operations
- else  
  - the second set of operations

**Summary of Pseudocode**

**Iterative:**
- Repeat until a condition becomes true  
  - the loop body

- While a condition remains true do  
  - the loop body
Exercises

I. Compute the average of 3 grades (1-9); if any one is 0 or negative, a message “Bad data” is printed

Get values for x, y, z
If x < 1 or y < 1 or z < 1 then
  Print message, “Bad data”
Else
  Set Average to (x + y + z) / 3
  Print the value of Average
Stop

Exercises

II. Compute the sum of n integers where n > 0

Get value for n, the number of integers
Get values for I1, I2, …, In, a list of n integers
Set the value of Sum to 0
Set the value of k to 1
Repeat until k > n
  Add Ik to Sum
  Add 1 to k
End of the loop
Print the value of Sum
Stop

Exercises

III. What does the following algorithm do?

Repeat until A > 0
  Print message, “Enter an integer”
  Get a value for A
End of the loop
Stop

IV. Write an algorithm that does the same but using a while loop instead of a repeat loop.
RECALL: Algorithms & Computing Agents
If we can discover an algorithm to perform a task, we can instruct a computing agent to execute it to solve the problem for us.

Algorithmic Problem Solving
Algorithm discovery
The process of finding a solution to a given problem

Typical Steps:
1. Understand the problem
2. Divide it into sub-problems
3. Sketch and refine, probably repeatedly
4. Test the correctness

Sequential search: an Example
Find the phone number of a given Name in an (unsorted) list of names and their phone numbers

<table>
<thead>
<tr>
<th>Names</th>
<th>Phone numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>$T_1$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>$T_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$N_{1000}$</td>
<td>$T_{1000}$</td>
</tr>
</tbody>
</table>
Sequential search: 1st Attempt

1. Get values for Name, \(N_1, \ldots, N_{1000}\), \(T_1, \ldots, T_{1000}\)
2. If Name = \(N_1\) then print the value of \(T_1\)
3. If Name = \(N_2\) then print the value of \(T_2\)
...
1000. If Name = \(N_{999}\) then print the value of \(T_{999}\)
1001. If Name = \(N_{1000}\) then print the value of \(T_{1000}\)
1002. Stop

Sequential search: Using A Loop

Get values for Name, \(N_1, \ldots, N_{1000}\), \(T_1, \ldots, T_{1000}\)
Set the value \(i\) to 1 and the value of Found to NO
Repeat until Found = Yes or \(i > 1000\)
   If Name = \(N_i\) then
       Print the value of \(T_i\)
       Set the value of Found to YES
   Else
       Add 1 to the value of \(i\)
End of loop
Stop

Selection: Find The Largest Number

Given a list of variables \(A_1, A_2, \ldots, A_n\), find the largest value and its (first) location

<table>
<thead>
<tr>
<th>Location</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

The largest is 8 at location 3

Idea (sketch): Go through the entire list, at each iteration find the largest-so-far and record its location
To begin with, set `largest-so-far` to (the value of) \( A_1 \)
set `location` to 1
set \( i \) to 2

Compare \( A_1 \) and \( A_2 \)
`largest-so-far` still holds the value of \( A_1 \)
set \( i \) to \( i+1 \)

Compare \( A_1 \) and \( A_3 \)
`largest-so-far` now holds the value of \( A_3 \)
`location` is 3
set \( i \) to \( i+1 \)
Continue the similar process until \( i = 8 \)

**Selection: Find The Largest Number**

- Get a value for \( n \), the size of the list
- Get values for \( A_1, A_2, \ldots, A_n \), the list to be searched
- Set \( \text{largest\_so\_far} \) to \( A_1 \) and set \( \text{location} \) to 1
- Set the value of \( i \) to 2
- While \( i \) is less or equal to \( n \) do
  - If \( A_i > \text{largest\_so\_far} \) then
    - Set the value of \( \text{largest\_so\_far} \) to \( A_i \)
    - Set the value of \( \text{location} \) to \( i \)
    - Add 1 to the value of \( i \)
  - End of loop
- Print the values of \( \text{largest\_so\_far} \) and \( \text{location} \)

**Algorithmic Problem Solving: Summary**

- Two examples of algorithmic problem solving
  - Sequential search
    - Q: On the average, how many comparisons (of names) does the algorithm make?
  - Selection
    - Q: Design a similar algorithm to find
      - the smallest value and its first location
      - the largest and all the locations holding it