Lecture 1: Pseudocode and Algorithm Basics

CS3310

Algorithms

What is an algorithm?

- A simple, *unambiguous*, mechanical procedure to solve some *problem*.

Important terms:

- A **problem** is a question which we want to answer.
    - Example: Sort a list *S* of *n* numbers in nondecreasing order.
- A **parameter** is a variable that is not assigned a specific value in the description of our problem, i.e. *S* and *n* in the above *problem* are *parameters*.
- An **instance** of a *problem* is a single assignment of values to its *parameters*:
    - S = [10, 7, 11, 5, 13, 8] and $n = 6$ is one possible *instance* to our *problem*.
- A **solution** or **output** to an *instance* of a *problem* is the answer that our algorithm should return.
    - [5, 7, 8, 10, 11, 13] is the *solution* to the above *instance*.

Algorithms

- With a very small input, most problems can be solved relatively quickly by humans.

Sort this list:
{ 5, 8, 7, 1, 2, 6 }

Algorithms

- However, with larger inputs, a computer is required if we’re interested in efficiency

Sort this list:
{ 55, 18, 7, 21, 2, 608, 110, 310, 910, 11, 12, 15, 61, 190, 75, 81, 14, 91, 80, 15, 43, 8, 100, 13, 17, 1, 8, 5, 12, 101, 214, 70, 16, 300, 9, 24, 37, 715, 212, 1, 34, 4, 0, 82, 73, 46, 54, 72, 102, 201, 305, 200, 60, 401, 501, 312, 502, 400, 231, 342, 320, 208, 876, 403, 109, 504, 31, 372, 462, 429, 480, 501, 430, 720, 58, 73, 24, 807, 819, 650, 460, 306, 203, 27, 189 }

Algorithms

- To tell a computer how to solve a problem, we need to produce a step-by-step procedure, called an **algorithm**, which solves *every* **instance** of a specific **domain**.
- The **domain** is the set of all *instances* a specific algorithm should solve.
  - For example, an *instance* of the sorting problem with $n < 0$ would not be part of a sorting algorithm's *domain*.
- It is crucial that every instance of the domain can be solved by our algorithm. If there is even one instance it does not solve, our algorithm is incorrect.

Designing an Algorithm

- There are multiple ways to communicate what an algorithm does:
    - Describe it in English
    - Implement the algorithm in a specific programming language
    - Write pseudocode

Designing an Algorithm

- There are multiple ways to communicate what an algorithm does:
    - Describe it in English
    - Implement the algorithm in a specific programming language
    - **Write pseudocode**

- Why write pseudocode instead of actual code?
    - It is simpler than describing it in English, as more complicated algorithms can lead to very long, confusing descriptions.
    - Readers with backgrounds in many different programming languages will look at an algorithm. Pseudocode should be written so it can easily be translated into any programming language.

Pseudocode

- While there is no universal standard format for pseudocode, it is usually best to be consistent within a specific project, book, paper, etc.
- I will use the pseudocode format the book uses (for the most part). It is similar to C++ with a few differences.
    - Data types that don't actually exist in C++ are used when their name clarifies a variable's purpose:
        - *index* (instead of *int*): an integer that refers to an index in an array
        - *keytype*: an element in an ordered array or list. (i.e. Instead of sorting an array of *ints* or *chars*, our pseudocode sorts an array of *keytypes*)
        - *number*: any numeric variable that could be either an *int* or a *float*
- **Important**: Arrays are indexed from 1 to n rather than from 0 to n - 1

Pseudocode

- When an `&` exists after the data type of a parameter, that variable is being passed by reference. The book often uses reference variables instead of return values
- Code is sometimes simplified into easier-to-understand English:

➢ `if (low ≤ x ≤ high)` **instead of** `if(low <= x && x <= high)`

➢ `exchange x and y` **instead of**

```
temp = x;
x = y;
y = temp;
```

Sometimes loops will take the following, simplified form:

```
repeat (n times)
```

I will often remove curly braces for the sake of space.

Example Pseudocode

What does the following algorithm do?

```
number alg1 (int n, const number S[])
{
    index i;
    number result;

    result = 0;
    for (i = 1; i <= n; i++)
        result = result + S[i];
    return result;
}
```

Example Pseudocode

What does the following algorithm do?

```
number alg1 (int n, const number S[])
{
    index i;
    number result;

    result = 0;
    for (i = 1; i <= n; i++)
        result = result + S[i];
    return result;
}
```

- Assume `n` is the # of elements in the array `S`:
    - `alg1` adds all the numbers in `S` and returns the result.

Example Pseudocode

How about this algorithm?

```
void alg2 (int n, const keytype S[], keytype x, index &location)
{
     location = 1;
     while (location <= n && S[location] != x)
          location ++;
     if (location > n)
          location = 0;
}
```

Example Pseudocode

How about this algorithm?

```
void alg2 (int n, const keytype S[], keytype x, index &location)
{
     location = 1;
     while (location <= n && S[location] != x)
          location ++;
     if (location > n)
          location = 0;
}
```

Determines if `x` is in S. Returns its location (via a reference variable) if it is, 0 otherwise.

Example Pseudocode

What does the following algorithm do?

➢ Arrays A, B, and C are $n \times n$ sized

```
void alg3 (int n, const number A[][], const number B[][], number C[][])
{
     index i, j, k;

     for (i = 1; i <= n; i++)
          for (j = 1; j <= n; j++)
               C[i][j] = 0;
               for (k = 1; k <= n; k++)
                    C[i][j] = C[i][j] + A[i][k] * B[k][j];
}
```

Example Pseudocode

What does the following algorithm do?

➢ Arrays A, B, and C are $n \times n$ sized

```
void alg3 (int n, const number A[][], const number B[][], number C[][])
{
    index i, j, k;

    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            C[i][j] = 0;
            for (k = 1; k <= n; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
}
```

- Multiplies arrays A and B, placing the result in C.
- **Note**: All pseudocode in this class assumes arrays are passed by reference by default.

Sequential Search

When describing an algorithm, we write out the problem, parameters, and outputs first:

Problem: Is the key *x* in the array S of *n* keys?

Parameters: Positive int *n*, array of keys S indexed from 1 to n, a key *x*

Outputs: the location of *x* in S, or 0 if x is not in S

```
void seqsearch (int n, const keytype S[], keytype x, index &location)
    location = 1;
    while (location <= n && S[location] != x)
        location ++;
    if (location > n)
        location = 0;
```

In-Class Exercise

1. Write a pseudocode algorithm that finds the largest number in a list of numbers.
2. Write a pseudocode algorithm that prints out all the subsets of three elements of a set of n elements. The elements of this set are stored in a list that is a parameter to the algorithm.