Introduction to Algorithms

Written by Rachel Singh, last updated Oct 16, 2018

About

In the realm of computer science, we often care about how fast our software is.

Boiling this down, we can actually investigate the quickness of a program based on the *efficiency* of the *algorithms* it uses.

In this section, we will look at algorithms, complexity, and growth rates from a mathematical perspective, but it will help you out with future classes like Data Structures & Algorithm Analysis.

Topics

1. What's an algorithm?

2. Coming up with algorithms.

3. (Aside): Testing

What's an Algorithm?

1. What's an Algorithm?

In mathematics and computer science, an algorithm (/ˈælgərɪðəm/ (About this sound listen)) is an unambiguous specification of how to solve a class of problems. Algorithms can perform calculation, data processing and automated reasoning tasks.

From https://en.wikipedia.org/wiki/Algorithm

Notes

1. What's an Algorithm?

Generally, an algorithm will have **inputs** and **outputs**, and the contents of the algorithm is a series of instructions that solve some problem.

In math, we also have functions:

$$output = f(input)$$

And in programming languages:

```
int Sum( int a, int b )
```

Notes

1. What's an Algorithm?

Characteristics of an algorithm:

- **Input:** It receives input
- **Output:** It produces output
- **Precision:** The steps are precisely stated.
- **Determinism:** The intermediate results of each step of execution are unique and determined only by the inputs and the results of the preceding steps.
- **Finiteness:** The algorithm terminates; that is, it stops after finitely many instructions have been executed.
- **Correctness:** The output produced by the algorithm is correct; that is, the algorithm correctly solves the problem.
- **Generality:** The algorithm applies to a set of inputs.

From Discrete Mathematics, Johnsonbaugh, pg 181-182

Notes

- **Input**
- **Output**
- **Precision**
- **Determinism**
- **Finiteness**
- **Correctness**
- **Generality**

1. What's an Algorithm?

What's not an algorithm?

- A function that only adds 2 + 5, and doesn't work for other integers. (Not general)
- A function called Sum with inputs a and b, and returns a*b as the result. (Not correct; badly named)
- A function that searches through a list of numbers at random, trying to find the number given as the input. (Not finite, not deterministic?)

Notes

- **Input**
- **Output**
- **Precision**
- **Determinism**
- **Finiteness**
- **Correctness**
- **Generality**

Coming up with Algorithms

2. Coming up with Algorithms

When we're coming up with an algorithm, we will usually want to think of it as if we're making a **function** in a programming language: We need to specify the input(s) and the output(s), and then come up with a list of instructions.

Notes

2. Coming up with Algorithms

Example: Come up with an algorithm (or function) that returns the larger of two numbers.

Specify inputs and outputs and instructions.

Notes

2. Coming up with Algorithms

Example: Come up with an algorithm (or function) that returns the larger of two numbers.

```
int GetMax( int a, int b )
{
   if ( a > b )     return a;
   else             return b;
}
```

(C++ code)

Notes

2. Coming up with Algorithms

Example: Come up with an algorithm that will return the sum of all numbers in a list.

Notes

2. Coming up with Algorithms

Example: Come up with an algorithm that will return the sum of all numbers in a list.

```
def Sum( myList ):
    sumVal = 0

    for num in myList:
        sumVal += num

    return sumVal
```

(Python code)

Notes

Testing

3. Testing

After you create an algorithm or a function in a program, how can you validate that it works as intended?

We can come up with **test cases**, which specify “for these inputs, I expect those outputs”.

Notes

3. Testing

For example, say we see the function declared:

```
int Sum( int, int );
```

But we can’t see inside of it. We can still test it out by writing test cases and then calling the function…

Notes

3. Testing

For example, say we see the function declared:

```
int Sum( int, int );
```

Input 1	Input 2	Expected Output	Actual Output
1	1	2	
0	3	3	
-4	4	0	
100	100	200	
5	-10	-5	

Now we have a series of test cases that we can use to validate the function. We run the function, and compare its **actual output** to the **expected output** we had pre-calculated.

Notes

3. Testing

Example: Write test cases for the following

```
bool IsOverdrawn( float money );
```

Input 1	Expected Output	Actual Output

Afterward, try writing a function to define this function as you think it would work. https://www.onlinegdb.com/online_c++_compiler

Notes

3. Testing

Example: Write test cases for the following

```
bool IsOverdrawn( float money );
```

Input 1	Expected Output	Actual Output
100.00	false	
1.00	false	
-5.00	true	
0.10	false	
0.00	false	

Did all the tests pass?

Notes

Conclusion

Next time we will talk more about algorithm analysis.