Algorithm Analysis ~ 01

CSE/IT 122 ~ Algorithms & Data Structures

What Is An Algorithm?

What Is An Algorithm?

- Definition: An algorithm is a definite procedure for solving a problem using a *finite* number of steps

Sum the first n Integers

As a C function, we might write:

```
unsigned long long sum_forloop(unsigned long n){
    unsigned long i = 1;
    unsigned long long sum = 0.0;

    for( ; 1 <=n; i++)
        sum += i;
    return sum;
}
```

Sum the first n Integers

As **pseudo-code**, we might write:

```
Sum(n)
    i := 1
    sum := 0

    for i := 1 to n
        sum := sum + 1
    next i

    return sum
```

The Running Time of a Program

- Running Time, denoted by **T(n)**, where *n* is the input size
- **T(n)** is the number of instructions executed on an idealized computer
- No units for **T(n)**

Algorithm Analysis

➔ How many instructions would our initial code execute before completing?

```
unsigned long long sum_forloop(unsigned long n){
    unsigned long i = 1;
    unsigned long long sum = 0.0;

    for( ; 1 <=n; i++)
        sum += i;
    return sum;
}
```

Algorithm Analysis

- Can we do better than T(n) = n to sum up the first n integers?
- Is there, perhaps a very simply way to arrive at that answer?
- A formula???

Sum The First N Integers ~ Try 2

Of course we can do better (We are computer scientists!)

```
Long sum(long sum){
    Return (n * (n + 1)) / 2; // cost $c_1$; one time
}
```

Square Root ~ Brute Force

- ➔ Give me a brute force solution to finding a square root

Square Root ~ Brute Force

- Give me a brute force solution to finding a square root
- An upper bound is (n/2*step) times
- Why?
- Is $x^{0.5} < x/2$?
- Yes. Show:
  - Since $x^2 - 4x = x(x - 4) > 0$ when $x > 4$
  - Solve $x^2 > 4x$. Take square roots of both sides
  - $x > sqrt(4x)$
  - $x/2 > sqrt(x)$

Square Root ~ Binary Search

- Find the square root using a binary search
  - **Algorithm:**
    - Start with *start* = 0, *end* = x
    - Do the following while *start* is smaller than or equal to *end*
      - Compute *mid* as (*start* + *end*) / 2
      - Compare *mid***mid* with x
      - If x is equal to *mid***mid*, return mid
      - If x is greater, do binary search between *mid+1* and *end*
      - If x is smaller, do binary search between *start* and *mid*
- Much better performance than brute force
- Can we do better?

Square Root ~ Newton's Method

- Let **P(x)** be a polynomial of degree *n*
- A **root** is a solution to **P(x)** = 0
- Apply this to square roots:
  - $x^2 - 99 = 0$ is a polynomial
  - $x^2 = 99$
  - x = sqrt(99)
  - Solving for *x* gives you the square root of 99

Square Root ~ Newton's Method

➔ Assume you know x_0 is close to the solution of the equation $f(x) = 0$, where $f(x)$ is a differentiable equation

➔ Then the tangent line F to the graph of f at the point with x-coordinate x_0 will ordinarily intersect the x-axis at a point whose x-coordinate, x_1 is closer to the solution than x_0

Square Root ~ Newton's Method

- ➔ Using the point-slope equation of the tangent line F, we can write:
  - $y - f(x_0) = f'(x_0)(x - x_0)$
  - Using the point $(x_0, f(x_0))$ and the slope of the line is the derivative
- ➔ If the tangent line F intersects the x-axis at $(x_1, 0)$ then:
  - $0 - f(x_0) = f'(x_0)(x_1 - x_0)$
  - If $f'(x)$ does not equal 0,
  - $(x_1 - x_0) = - f(x_0)/f'(x_0)$ and
  - $X_1 = x_0 - f(x_0)/f'(x_0)$
- ➔ Then simply repeat this process to get closer to x_1

Square Root ~ Newton's Method

- As you repeat, you produce a sequence of numbers
  - $x_0, x_1, x_2, \dots, x_n, \dots$
  - determined by the formula $x_{n+1} = x_n - f(x_n)/f'(x_n)$
- For any *k*, finding the root of $x^2 - k = 0$ will find the sqrt(k)
- What is the derivative?
  - 2x
- Thus, our formula is:
  - $X_{n+1} = x_n - (x_n * x_n - k) / (2 * x_n)$

Proving the Sum of the First N Positive Integers

- ➔ Why does the sum of the first *n* positive integers equal n(n+1) / 2?

Proving the Sum of the First N Positive Integers

Example:

- ➔ 1+2+3+4+5+6+7+8+9+10
  - Notice the following:
  - 1 + 10 = 11
  - 2 + 9 = 11
  - 3 + 8 = 11
  - 4 + 7 = 11
  - 5 + 6 = 11
- ➔ This happens 5 times so 5 * 11 = 55
- ➔ Same as the formula 10(10 + 1) / 2
- ➔ This is not a *proof* for all *n*. Only shows it for the case n = 10