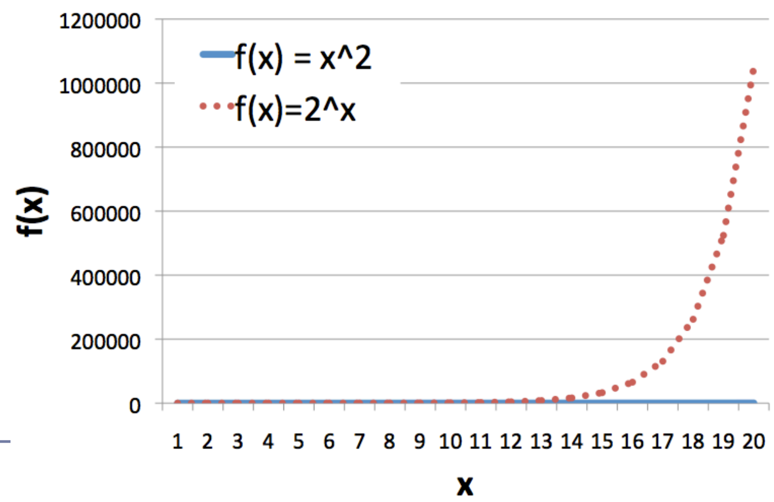
Computational Complexity

- Algorithms are **not** made to be used only once & are **not** made to be used for only one particular problem.
- How long does it take for the algorithm to find the solution when the input size increases ?
  - In particular, is it polynomial or exponential ?



Example

- Suppose A is an array of natural numbers (indexed from 0) and we want to sort it in ascending order using:
- For (i = 1 ; i < length(A) ; i ++) {

```
    valueToInsert = A[i]
    holePos = i
    while (valuePos > 0 and valueToInsert < A[holePos-1]) {
            A[holePos] = A[holePos-1]
        holePos = holePos-1
    }
    A[holePos] = valueToInsert
}
```

Example

- Suppose A is an array of natural numbers (indexed from 0) and we want to sort it in ascending order using:
- For (i = 1 ; i < length(A) ; i ++) {

```
    valueToInsert = A[i]
    holePos = i
    while (valuePos > 0 and valueToInsert < A[holePos-1]) {
            A[holePos] = A[holePos-1]
        holePos = holePos-1
    }
    A[holePos] = valueToInsert
}
```

- $t(n) = c_1.(1 + 2 + 3 + \ldots + n\text{-}1) + c_2.(n\text{-}1)$

  n: length(A)

Measuring Time Complexity

- Big Oh: $O(g(n))$
- Big Omega: $\Omega(g(n))$
- Theta: $\Theta(g(n))$

- Little Oh: $o(g(n))$
- Little Omega: $\omega(g(n))$

Big Oh: $O(g(n))$

- The set of all functions with **smaller** or the same order of growth as $g(n)$.
- A function $t(n)$ is in $O(g(n))$ if $t(n)$ is bounded above by some positive constant multiplication of $g(n)$ for large n
  - There's at least a positive constant c s.t.

$$0 \le t(n) \le cg(n) \text{ for all } n \ge n_0$$

Big Omega: $\Omega(g(n))$

- The set of all functions with **larger** or the same order of growth as $g(n)$.
- A function $t(n)$ is in $\Omega(g(n))$ if $t(n)$ is bounded below by some positive constant multiplication of $g(n)$ for large n
  - There's at least a positive constant c s.t.

$$0 \leq cg(n) \leq t(n) \text{ for all } n \geq n_0$$

Theta: $\Theta(g(n))$

- The set of all functions with the same order of growth as g(n).
- A function t(n) is in $\Theta(g(n))$ if t(n) is bounded both above and below by some positive constant multiplication of g(n) for large n
  - There's at least a positive constant c s.t.

$$0 \le c_1 g(n) \le t(n) \le c_2 g(n) \text{ for all } n \ge n_0$$

Little Oh, Little Omega

- Same as their Big Oh & Big Omega counterpart, but must hold for all positive constant multiplication.
  - In other words, strict upper / lower bound.
  - The order of growth is either always smaller or always lower, cannot be equal.
  - Little Oh implies Big Oh: If $t(n)$ is in $o(g(n))$, it will also be in $O(g(n))$.
  - Little Omega implies Big Omega: If $t(n)$ is in $\Omega(g(n))$, it will also be in $\omega(g(n))$.