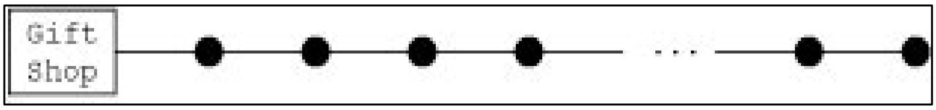
# Algorithm Analysis: The Big-O Notation

- Analyze algorithm after design
- **Delivering packages example**

Calculate the shortest distance from the shop to a particular destination.

  – 50 packages delivered to 50 different houses
  – 50 houses one mile apart, in the same area



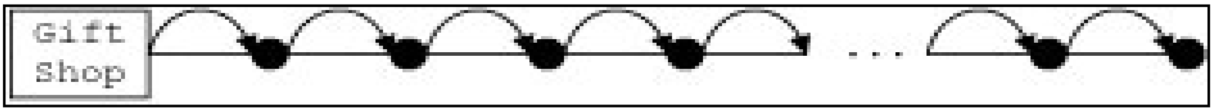

**Gift shop and each dot representing a house**

## Package delivering scheme 1

– Driver picks up all 50 packages
– Drives one mile to first house, delivers first package
– Drives another mile, delivers second package
– Drives another mile, delivers third package, and so on
– Distance driven to deliver packages

$$1+1+1+\ldots+1 = 50 \text{ miles}$$

– Total distance traveled: **50 + 50 = 100 miles**



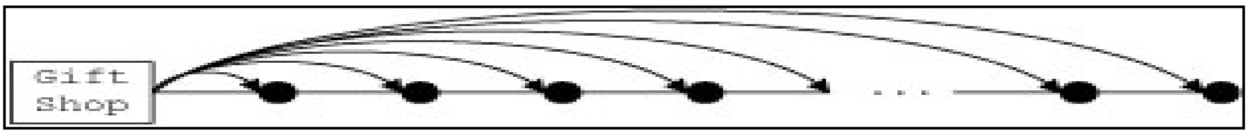

**Package delivering scheme 1**

## Package delivering scheme 2

– Similar route to deliver another set of 50 packages
- Driver picks up first package, drives one mile to the first house, delivers package, returns to the shop
- Driver picks up second package, drives two miles, delivers second package, returns to the shop

– Total distance traveled

$$2 * (1+2+3+\ldots+50) = 2550 \text{ miles}$$




**package delivery scheme 2**

– n packages to deliver to n houses, each one mile apart
– First scheme: total distance traveled

$$1+1+1+\ldots+n = 2n \text{ miles}$$

**Function of n**

– Second scheme: total distance traveled

$$2 * (1+2+3+\ldots+n) = 2*(n(n+1) / 2) = n^2+n$$

**Function of $n^2$** ($n^2$ is the dominant term in the above equation)

**Various values of $n$, $2n$, $n^2$, and $n^2 + n$**

| $n$ | $2n$ | $n^2$ | $n^2 + n$ |
|---|---|---|---|
| 1 | 2 | 1 | 2 |
| 10 | 20 | 100 | 110 |
| 100 | 200 | 10,000 | 10,100 |
| 1000 | 2000 | 1,000,000 | 1,001,000 |
| 10,000 | 20,000 | 100,000,000 | 100,010,000 |

**Analyzing an algorithm**

- Count number of operations performed by the algorithm, Not affected by computer speed

- **Example 1-1**
  - Illustrates fixed number of executed operations

```
cout << "Enter two numbers";                               //Line 1

cin >> num1 >> num2;                                       //Line 2

if (num1 >= num2)                                          //Line 3
    max = num1;                                            //Line 4
else                                                       //Line 5
    max = num2;                                            //Line 6

cout << "The maximum number is: " << max << endl;          //Line 7
```

The total number of operations performed by the above code is equal to 8

- **Example 1-2**
  - Illustrates dominant operations

```
cout << "Enter positive integers ending with -1" << endl;  //Line 1

count = 0;                                                 //Line 2
sum = 0;                                                   //Line 3

cin >> num;                                                //Line 4

while (num != -1)                                          //Line 5
{
    sum = sum + num;                                       //Line 6
    count++;                                               //Line 7
    cin >> num;                                            //Line 8
}

cout << "The sum of the numbers is: " << sum << endl;      //Line 9

if (count != 0)                                            //Line 10
    average = sum / count;                                 //Line 11
else                                                       //Line 12
    average = 0;                                           //Line 13

cout << "The average is: " << average << endl;             //Line 14
```

Line 1-4 has 5 operations,
Line 5-8 has 5 operations
Line 9-14 has 8 or 9 operations depending on whether line 11or line 13 executes.
Then the total number of operations, when the while loop executes $n$ times

$$5n + 15 \; or \; 5n + 14$$

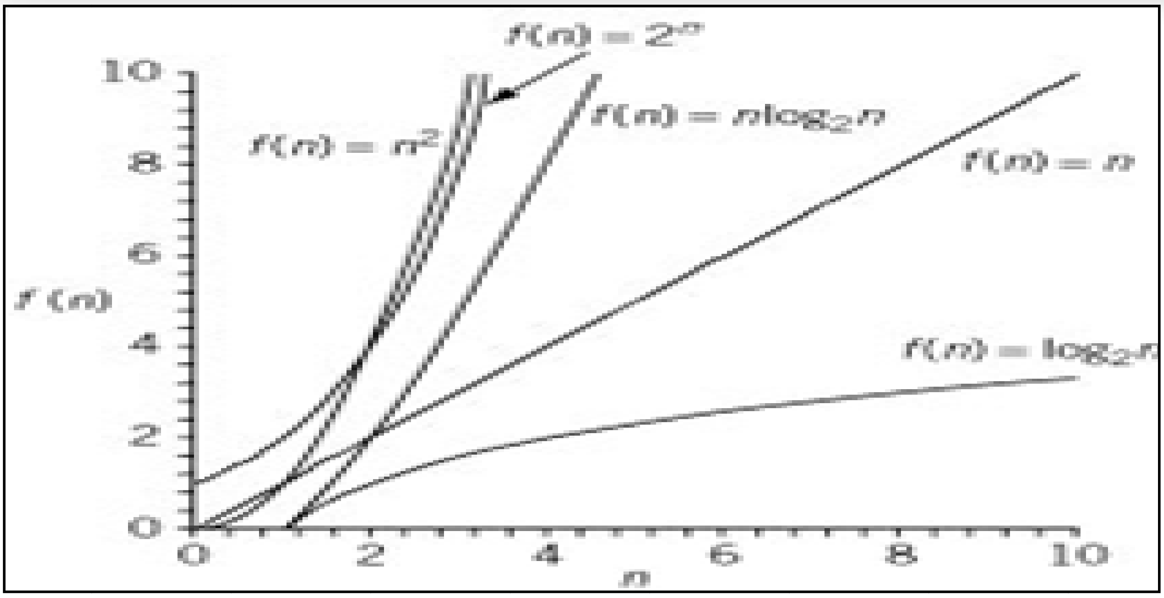
Note: one extra operation is executed to terminate the loop.
For very large$n$, the term $5n$ becomes the dominating term and 14 or 15 becomes insignificant.
Certain operations in different algorithms are dominant
Example:
In Matrix multiplication, the two operations are addition and multiplication, but the dominant is the multiplication, so we count the total number of multiplications operations.

**Search algorithm**

- n: represents list size
- f(n): count function
- Number of comparisons (dominant operation) in search algorithm.
- c: units of computer time to execute one operation
- cf(n): computer time to execute f(n) operations
- Constant c depends computer speed (varies)
- f(n): number of basic operations (constant)
- Determine algorithm efficiency
- Knowing how function f(n) grows as problem size grows

**Growth rates of various functions**

| $n$ | $\log_2 n$ | $n \log_2 n$ | $n^2$ | $2^n$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 2 |
| 2 | 1 | 2 | 2 | 4 |
| 4 | 2 | 8 | 16 | 16 |
| 8 | 3 | 24 | 64 | 256 |
| 16 | 4 | 64 | 256 | 65,536 |
| 32 | 5 | 160 | 1024 | 4,294,967,296 |




**Growth rate of functions in the previous table**

- Notation useful in describing algorithm behavior
  - Shows how a function *f(n)* grows as *n* increases without bound
- Asymptotic
  - Study of the function *f* as *n* becomes larger and larger without bound
  - Examples of functions
    - $g(n)=n^2$ (no linear term)
    - $f(n)=n^2 + 4n + 20$

As *n* becomes larger and larger
  - Term $4n + 20$ in $f(n)$ becomes insignificant
  - Term $n^2$ becomes dominant term

**Growth rate of $n^2$ and $n^2 + 4n + 20n$**

| $n$ | $g(n) = n^2$ | $f(n) = n^2 + 4n + 20$ |
|---|---|---|
| 10 | 100 | 160 |
| 50 | 2500 | 2720 |
| 100 | 10,000 | 10,420 |
| 1000 | 1,000,000 | 1,004,020 |
| 10,000 | 100,000,000 | 100,040,020 |

- Algorithm analysis
  - If function complexity can be described by complexity of a quadratic function without the linear term
    - We say the function is of $O(n^2)$ or Big-O *of* $n^2$
- Let *f* and *g* be real-valued functions
  - Assume *f* and *g* nonnegative
    - For all real numbers $n$, $f(n) >= 0$ and $g(n) >= 0$
- $f(n)$ is Big-O of $g(n)$: written $f(n) = O(g(n))$
  - If there exists positive constants $c$ and $n_0$ such that $f(n) <= cg(n)$ for all $n >= n_0$

**Some Big-O functions that appear in algorithm analysis**

| Function $g(n)$ | Growth rate of $f(n)$ |
|---|---|
| $g(n) = 1$ | The growth rate is constant and so does not depend on $n$, the size of the problem. |
| $g(n) = \log_2 n$ | The growth rate is a function of $\log_2 n$. Because a logarithm function grows slowly, the growth rate of the function $f$ is also slow. |
| $g(n) = n$ | The growth rate is linear. The growth rate of $f$ is directly proportional to the size of the problem. |
| $g(n) = n\log_2 n$ | The growth rate is faster than the linear algorithm. |
| $g(n) = n^2$ | The growth rate of such functions increases rapidly with the size of the problem. The growth rate is quadrupled when the problem size is doubled. |
| $g(n) = 2^n$ | The growth rate is exponential. The growth rate is squared when the problem size is doubled. |