Note: in this lecture “function” almost always refers to a mathematical function, as in \( f(x) = x + 101 \)

Sections 6.1, 6.2, 6.4 (optional), 6.6 (not 6.6.3)

An algorithm is a clearly specified set of instructions a computer follows to solve a problem.

An algorithm should be
- correct
- efficient: not use too much time or space

Algorithm analysis: determining how much time and space a given algorithm will consume.

Note that two very different algorithms can solve the same problem
- bubble sort vs. quicksort
- List insert in an array-based implementation vs. a linked-list-based implementation.

How do we know which is faster (more efficient in time)?

Why not just run both on same data and compare?

Could measure the time each one takes to execute, but that is subject to various external factors
- multitasking operating system
- speed of computer
- language solution is written in (compiler)

Need a way to quantify the efficiency of an algorithm independently of execution platform, language, or compiler.
Estimating execution time

- The amount of time it takes an algorithm to execute is a function of the input size.
- We use the **number of statements executed** (given a certain input size) as an approximation of the execution time.
- Count up statements executed for a program or algorithm as a function of the amount of data
  - For a list of length N, it may require executing $3N^2+2N+125$ statements to sort it using a given algorithm.

Counting statements

- Each single statement (assignment, output) counts as 1 statement
- A boolean expression (in an if stmt or loop) is 1 statement
- A function call is equal to the number of statements executed by the function.
- A loop is basically the number of times the loop executes times the number of statements executed in the loop.
  - usually counted in terms of N, the input size.

Counting statements example

```java
int total(int[] values, int numValues)
{
    int result = 0;
    for(int i = 0; i < numValues; i++)
        result += values[i];
    return result;
}
```

- What does N (input size) represent in this case?
  - the number of values in the array (==numValues)
- Tally up the statement count:
  - int result = 0; (1)  - result += values[i]; (N)
  - int i=0; (1)  - return result; (1)
  - i < numValues (N+1)
  - i++ (N)

Comparing functions

- Is $3N+4$ good? Is it better (less) than
  - $5N+5$ ?
  - $N+1,000$ ?
  - $N^2 + N + 2$ ?
- Hard to say without graphing them.
- Even then, are the differences significant?
Comparing functions

- When comparing these functions in algorithm analysis
  - We are concerned with very large values of N.
  - We tend to ignore all but the “dominant” term.

  At large values of N, 3N dominates the 4 in 3N+4
  
  - We also tend to ignore the constant factor (3).
  - We want to know which function is growing faster (getting bigger for bigger values of N).

Classifications of (math) functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Formula</th>
<th>Big O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>f(x)=b</td>
<td>O(1)</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>f(x)=log_b(x)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Linear</td>
<td>f(x)=ax+b</td>
<td>O(n)</td>
</tr>
<tr>
<td>Linearithmic</td>
<td>f(x)=x log_b(x)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Quadratic</td>
<td>f(x)=ax^2+bx+c</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>Exponential</td>
<td>f(x)=b^x</td>
<td>O(2^n)</td>
</tr>
</tbody>
</table>

- Last column is “big Oh” notation, used in CS.

Function classifications

- Constant        | f(x)=b                | O(1)          |
- Logarithmic     | f(x)=log_b(x)         | O(log n)      |
- Linear          | f(x)=ax+b             | O(n)          |
- Linearithmic    | f(x)=x log_b(x)       | O(n log n)    |
- Quadratic       | f(x)=ax^2+bx+c        | O(n^2)        |
- Exponential     | f(x)=b^x              | O(2^n)        |

Last column is “big Oh” notation

Comparing functions

- For a given function expressing the time it takes to execute a given algorithm in terms of N,
  - we ignore all but the dominant term and put it in one of the function classifications.

- Which classifications are more efficient?
  - The ones that grow more slowly.
Comparing functions

- Graph 1

- Graph 2

Comparing functions

- Assume N is 100,000, processing speed is 1,000,000,000 operations per second

<table>
<thead>
<tr>
<th>Function</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^N$</td>
<td>$3.2 \times 10^{30086}$ years</td>
</tr>
<tr>
<td>$N^2$</td>
<td>3171 years</td>
</tr>
<tr>
<td>$N^3$</td>
<td>11.6 days</td>
</tr>
<tr>
<td>$N \log N$</td>
<td>10 seconds</td>
</tr>
<tr>
<td>$N$</td>
<td>0.0017 seconds</td>
</tr>
<tr>
<td>square root of $N$</td>
<td>$3.2 \times 10^{-7}$ seconds</td>
</tr>
<tr>
<td>$\log N$</td>
<td>$1.2 \times 10^8$ seconds</td>
</tr>
</tbody>
</table>

Formal Definition of Big O

“Order F of N”

- $T(N)$ is $O(F(N))$ if there are positive constants $c$ and $N_0$ such that $T(N) \leq cF(N)$ when $N \geq N_0$
  - $N$ is the size of the data set the algorithm works on
  - $T(N)$ is the function that characterizes the actual running time of the algorithm (like $3N+4$)
  - $F(N)$ is a function that characterizes an upper bounds on $T(N)$. It is a limit on the running time of the algorithm. (The typical Big O functions)
  - $c$ and $N_0$ are constants. We pick them to make the definition work.
Example using definition

- Given $T(N) = 3N + 4$, prove it is $O(N)$.
  - $F(N)$ in the definition is $N$
  - We need to choose constants $c$ and $N_0$ to make $T(N) \leq cF(N)$ when $N \geq N_0$ true.
  - Lets try $c = 4$ and $N_0 = 5$.
  - Graph on next slide shows: $3N+4$ is less than $4N$ whenever $N$ is bigger than 5

Demonstrating $3N+4$ is $O(N)$

vertical axis: execution time

horizontal axis: $N$, number of elements in data set

Big O is an upper bound
(any upper bound)

- If $T(N)$ is $O(F(N))$, then for big values of $N$, $cF(N)$ is greater than $T(N)$.
  - If $T(N)$ is $O(n)$, it is also $O(n^2)$.
  - If $T(N)$ is $O(n)$, it is also $O(n^3)$.
  - If $T(N)$ is $O(n)$, it is also $O(n \log n)$.
- But these statements are not very informative.
  - If $T(N)$ is $O(n)$, it may or may not be $O(1)$.
  - We prefer to state the tightest bound possible when analyzing algorithms.

Best, Average, Worst case analyses

Because certain data values may affect execution time (still functions of $N$):

- Best case: fewest possible statements executed considering all possible cases of input of size $N$
- Average case: average number of statements executed for all possible cases of input of size $N$
- Worst case: maximum number of statements that could be executed for all possible cases of input of size $N$

Consider examples 1 and 2 that follow:
Example 1:

```cpp
bool findNum(double[] values, int numValues, double num) {
    int found = false;
    for(int i = 0; i < numValues; i++)
        if(values[i] == num)
            found = true;
    return found;
}
```

- T(N) is O(F(N)) for what function F?
  - best case?
  - average case?
  - worst case?

Note: all have the same efficiency function for this code.

Example 2:

```cpp
bool findNum(double[] values, int numValues, double num) {
    for(int i = 0; i < numValues; i++)
        if(values[i] == num)
            return true;
    return false;
}
```

- T(N) is O(F(N)) for what function F?
  - best case?
  - average case of all inputs that return true?
  - worst case?

Example 3:

```cpp
Matrix Matrix::add(Matrix rhs) {
    Matrix sum = new Matrix(numRows(), numCols(), 0);
    for(int row = 0; row < numRows(); row++)
        for(int col = 0; col < numCols(); col++)
    return sum;
}
```

- T(N) is O(F(N)) for what function F?

Is there a best case or worst case?

Example 4:

```java
public void selectionSort(double[] data, int numValues) {
    int n = numValues;
    int min;
    double temp;
    for(int i = 0; i < n; i++)
        for(int j = i+1; j < n; j++)
            if(data[j] < data[min])
                min = j;
        temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
```

- T(N) is O(F(N)) for what function F?

Note: 1+2+3+...+N = N*(N+1)/2
Example 5:

\[
\text{public int func(int[] list, int length)}\{
\text{    int total = 0;}
\text{    for(int i = 0; i < length; i++){}
\text{        total += countDups(list[i], list);}
\text{    }\}
\text{    return total;}
\text{ }\}
\text{// method countDups is O(N) where N is the}
\text{// length of the array it is passed}
\]

- T(N) is O(F(N)) for what function F?