Algorithms

- 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.

Algorithms

- 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?
Estimating execution time

- We use the **number of statements executed** as an approximation of the execution time.
- The amount of time it takes an algorithm to execute is a function of the input size.
- Count up statements in a program or method or algorithm as a function of the amount of data
  - For a list of length \( N \), it may take \( 3N^2 + 2N + 125 \) 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.
  - but 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 is \( N \) (input size) 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)

Total = \( 3N + 4 \)

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).

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 \( f(x) = b^x \) \( O(2^x) \)

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 1](image-url)
Comparing functions

- Graph 2

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^N</td>
<td>3.2 x 10^3008 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^4</td>
<td>10 seconds</td>
</tr>
<tr>
<td>N log N</td>
<td>0.0017 seconds</td>
</tr>
<tr>
<td>N</td>
<td>0.0001 seconds</td>
</tr>
<tr>
<td>square root of N</td>
<td>3.2 x 10^{-7} seconds</td>
</tr>
<tr>
<td>log N</td>
<td>1.2 x 10^{-4} seconds</td>
</tr>
</tbody>
</table>
```

Comparing functions

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

Formal Definition of Big O

- T(N) is O(F(N)) if there are positive constants c and N₀ such that T(N) <= cF(N) when N >= N₀
- 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
- 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 N₀ 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 N₀ to make T(N) <= cF(N) when N >= N₀ true.
  - Lets try c = 4 and N₀ = 5.
  - Graph on next slide shows: 3N+4 is less than 4N when N is bigger than 5
Demonstrating 3N+4 is O(N)

- T(N), actual function of time.
  - In this case 3N + 4
- F(N), approximate function of time.
  - In this case N
- c * F(N), in this case, c = 4, c * F(N) = 4N

Best, Average Worst case analyses

- Best case: fewest possible statements executed
  - least interesting
- Average case: number of statements executed for most cases of input, or normal cases
  - pick an input set that is randomly distributed
- Worst case: maximum number of statements that could be executed
  - pick input set that would require the most statements to be executed.

Best, Average Worst case analyses

Example 1:

```c
bool findNum(double[] values, int numValues, double num) {
    for(int i = 1; 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?
  - worst case?
Example 2:

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?

Example 3:

public void selectionSort(double[] data, int numValues)
{ int n = numValues;
    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;
} // end of outer loop, i

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

Example 4:

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

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

Example 5:

- Insert (and remove) for List_3358
  - implemented using arrays (in class: see below)
  - implemented using linked lists
- These operations are O(__) ?

```java
void List_3358::remove() {
    assert(!atEOL() && !isEmpty());
    for (int i=cursor; i < currentSize-1; i++)
        values[i] = values[i+1];
    currentSize--;
    if (isEmpty())
        cursor = EOL;
}
```