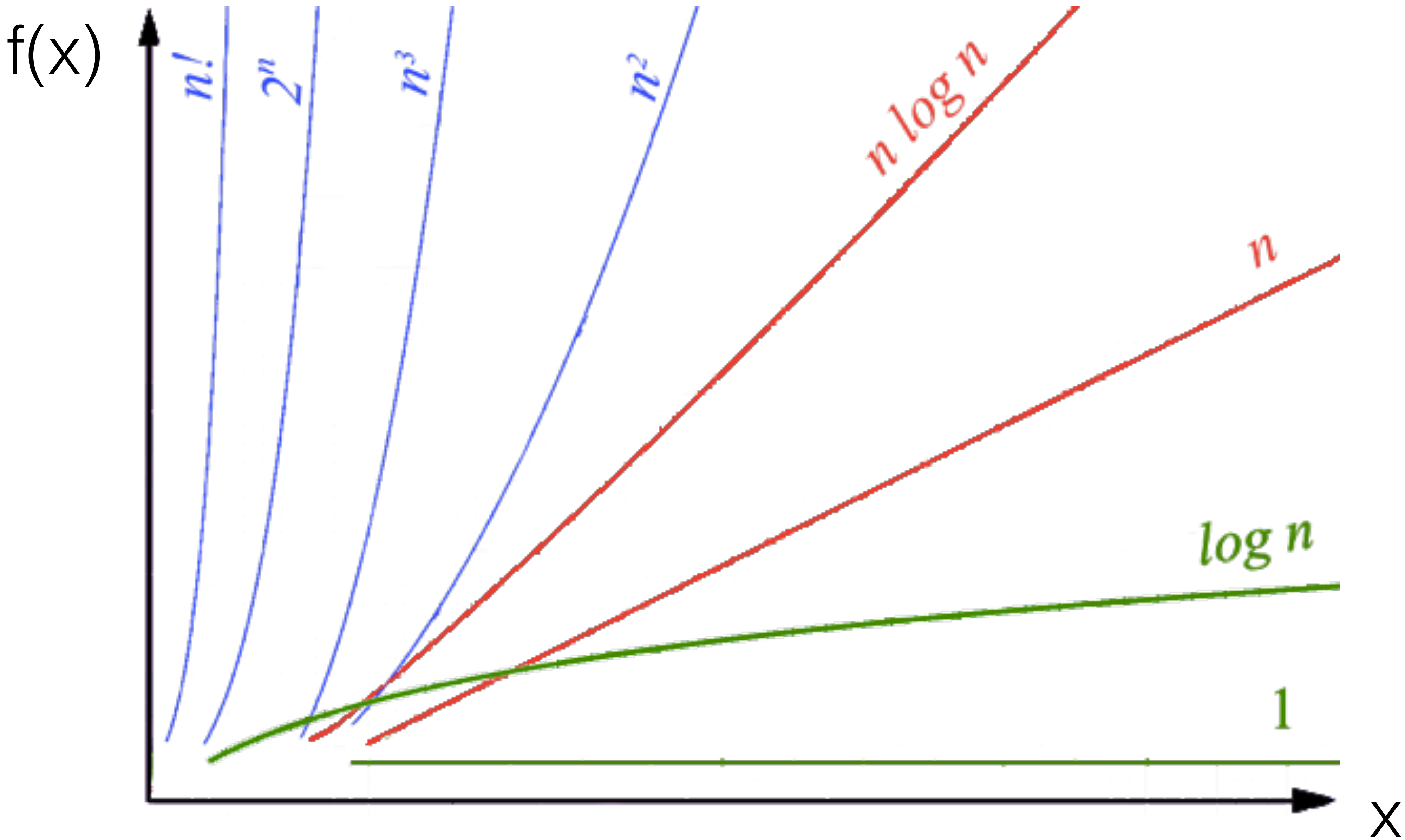
Algorithmic Complexity

Historical Perspective

1936: The Turing Machine
1940-50: Development of digital computers
1960s: Hartmanis and Stearns develop computational complexity.
P and NP classes, P=NP problem.
1970s: Cook and Karnap: NP-completeness
Cook-Levin Theorem
1980s: Combinatorial techniques introduced, circuits.
1990s: A new model of computation: the quantum computer.

Mathematical Interlude: How functions grow



Remember: the "big-O" notation:

$$O(1) < O(\log_2 N) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < \ldots < O(2^N) < \ldots$$

?

Algorithms

How much time will and algorithm take for producing the result? —> **Time Complexity**

How much memory will an algorithm need? —> **Space Complexity**

We are interested in the **asymptotic** behaviour of the algorithms —> “big-O” notation.

We would like to estimate the asymptotic time/space requirements of an algorithm, given the size N of the input problem. Moreover, we would like to give an answer which is independent from the kind of computer we are using.

Example:
How much time/space do I need for multiplying two vectors of length N?

Other "Big-" Notations

$O(f(N))$ **Upper bound**

$\Omega(f(N))$ **Lower bound**

$\Theta(f(N))$ **Double-sided bound**

$$f(N) \in O(g(N)) \text{ if } \lim_{N\to\infty} \frac{|f(N)|}{|g(N)|} < \infty$$

$$f(N) \in \Omega(g(N)) \; if \; \exists C, k : |f(N)| \geq |g(N)| \forall N > k$$

$$f(N) \in \Theta(g(N)) \; if \; f \in O(g(N)) \wedge f \in \Omega(g(N))$$

Examples

- Simple operations
- Nested Loops
- Arrays vs Linked Lists
- Binary Search / Divide and Conquer Algorithms.

- What about the time complexity of recursive algorithms?

The Master Theorem

Consider a recursive equation with the following form:

$$T(N) = aT(N/b) + f(N)$$

where a>=1 and b>1 and f is asymptotically positive.
The possible solutions to the equation are (3 cases):

$$f(N) \in O(N^c), c < \log_b a \Rightarrow T(N) \in \Theta(N^{\log_b a})$$

$$f(N) \in \Theta(n^c \log^k N), c = \log_b a \Rightarrow T(N) \in \Theta(n^c \log^{k+1} n)$$

$$f(N) \in \Omega(N^c), c > \log_b a, af(N/b) \leq kf(N) \Rightarrow T(N) \in \Theta(f(N))$$

NOTE: The Master theorem does not exhaust all the possible functional forms.

The Master Theorem: Application to the Binary Search

```
int binary_search(int array[], int left, int right ,int item)
{
    int middle = ( left + right ) / 2;
   if (array[middle] > item) return binary_search(array, left, middle - 1,item);
   else if (array[middle] < item) return binary_search(array, middle + 1, right,item);

   else return middle;
}
```

f(N)

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

Apply Master Theorem in the second case with:

$$c = \log_b a$$

a=1

b=2

c=0

k=0

The result is: $T(N) \in \Theta(\log_2 N)$

The Master Theorem: Application to the Towers of Hanoi

```
void solve(int count, char source, char dest, char spare){

  if (count == 1) cout << "Move from " << source << " to " << dest << endl;

  else {
    solve(count-1,source,spare,dest); //move n-1 disks from A to C
    solve(1,source,dest,spare);       //move the remaining disk from A to B
    solve(count-1,spare,dest,source); //move n-1 disks from C to B ...
  }
}
```

$$T(N) = T(N-1) + T(N-1) + 1 = 2T(N-1) + 1$$

Substituting $n = \log_2 m$

we can bring the equation in a form suitable to the Master Theorem, which in turn proves that:

$$T(N) = \Theta(2^N)$$

The legend says that the monks in a remote temple have to bring 64 disks from the first peg to the third. When done, the world will end. If it takes 1s for them to bring a disk from one peg to the other, should we worry about the end of the world? The answer tells you something about the power of exponentials!