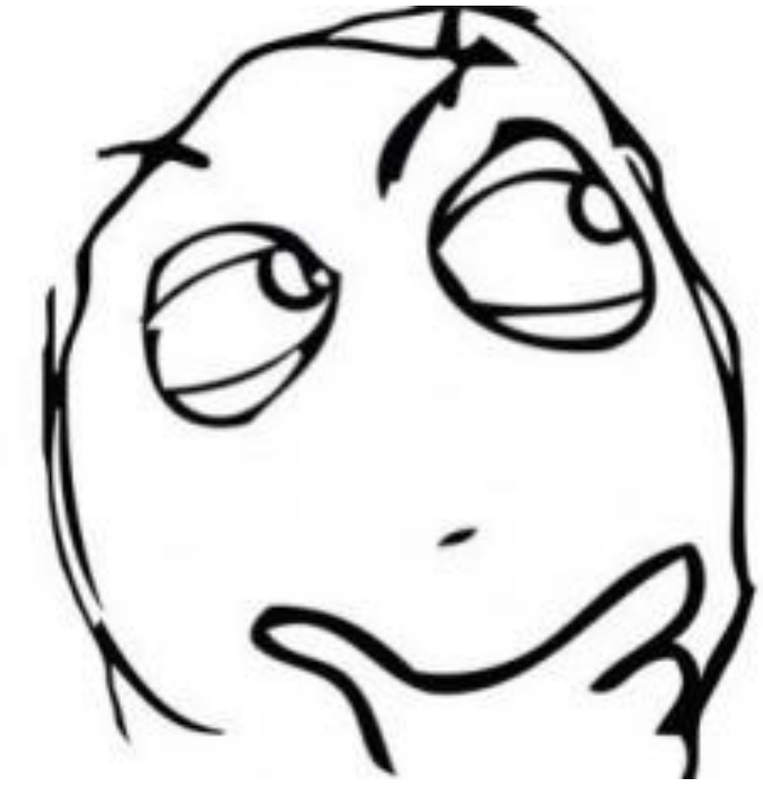
Complexity Analysis

What is it?



Linear Search
Vs
Binary Search

Mathematical Significance

Big Oh

Given nonnegative functions $f, g : \mathbb{R} \to \mathbb{R}$, we say that

$$f = O(g)$$

$$\limsup_{x \to \infty} f(x)/g(x) < \infty.$$

Big Omega

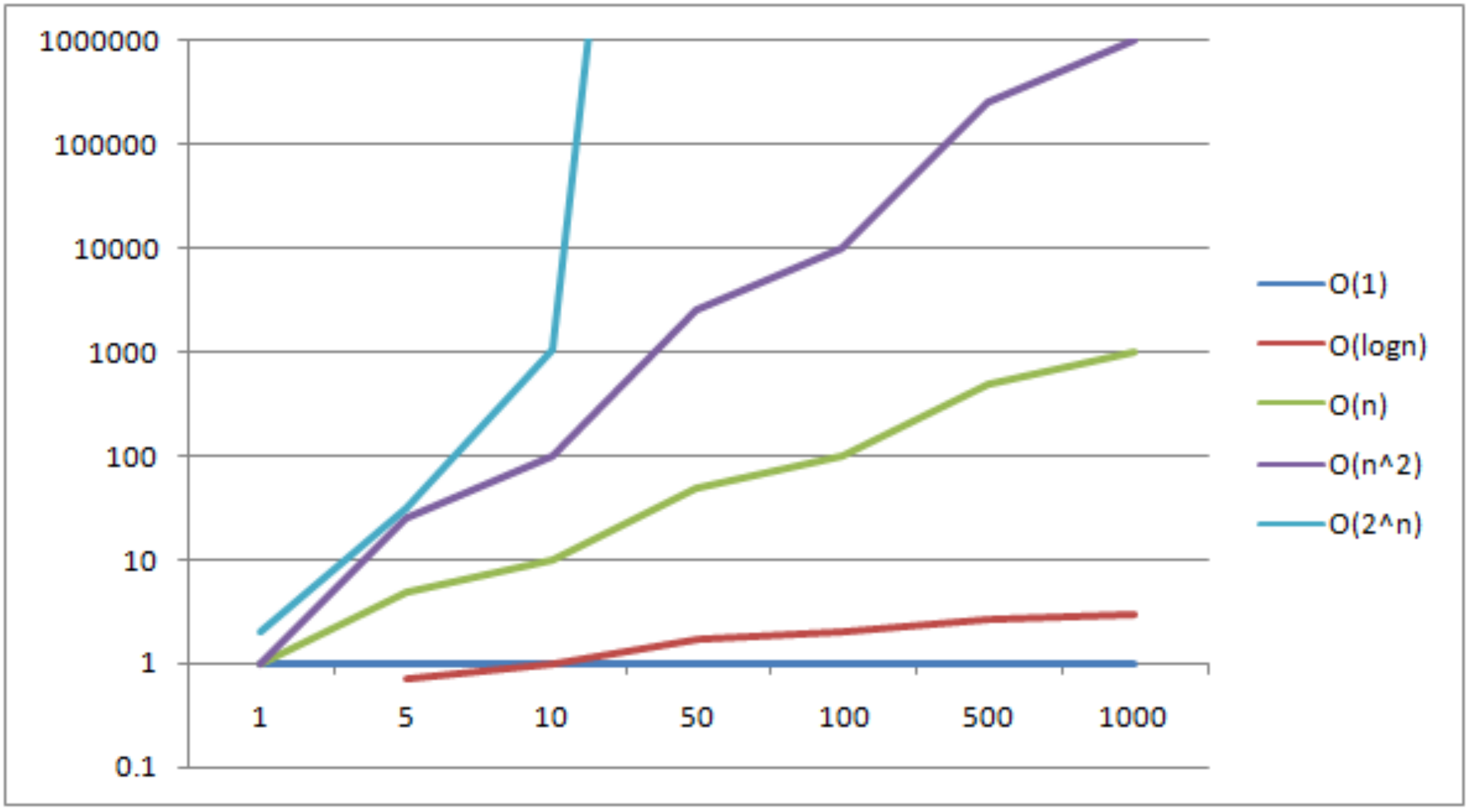
In other words, $f(x) = \Omega(g(x))$ means that $f(x)$ is greater than or equal to $g(x)$, except that we are willing to ignore a constant factor and to allow exceptions for small x.

If all this sounds a lot like big-Oh, only in reverse, that's because big-Omega is the opposite of big-Oh. More precisely,

Theta

Sometimes we want to specify that a running time $T(n)$ is precisely quadratic up to constant factors (both upper bound *and* lower bound). We could do this by saying that $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$, but rather than say both, mathematicians have devised yet another symbol, Θ, to do the job.

Graphical representation



Space Complexity or Auxiliary Space?

Auxiliary Space is the extra space or temporary space used by an algorithm.

Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

For example, if we want to compare standard sorting algorithms on the basis of space, then Auxiliary Space would be a better criteria than Space Complexity. Merge Sort uses O(n) auxiliary space, Insertion sort and Heap Sort use O(1) auxiliary space. Space complexity of all these sorting algorithms is O(n) though.

Questions!

```
for(int I = 1; I <= N;){
for(int j = 1; j <= k; j++){
// some operation taking time c.
}
I += k;
}
```

Take as input N, a number. Take N more inputs to form an array. Take as input M, a number. Take M more inputs to form an array. Write a function which returns intersection of both arrays. Target complexity is O((n + m)logm)) where m is the size of smaller array.

Selection Sort

Worst complexity: n^2

Average complexity: n^2

Best complexity: n^2

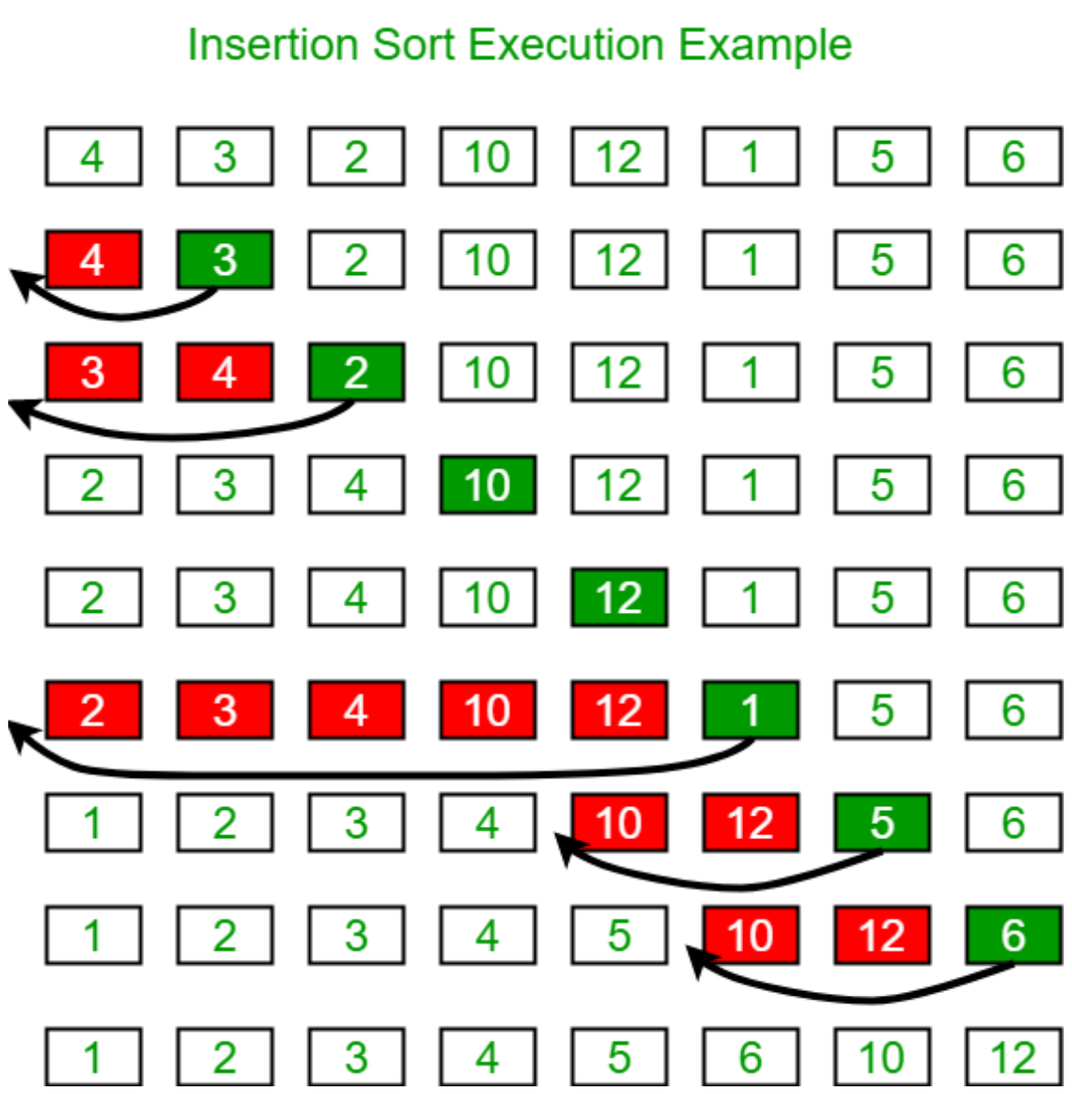
Space complexity: 1

Method: Selection

Stable: No

The good thing about selection sort is it never makes more than O(n) swaps and can be useful when memory write is a costly operation.

Insertion Sort



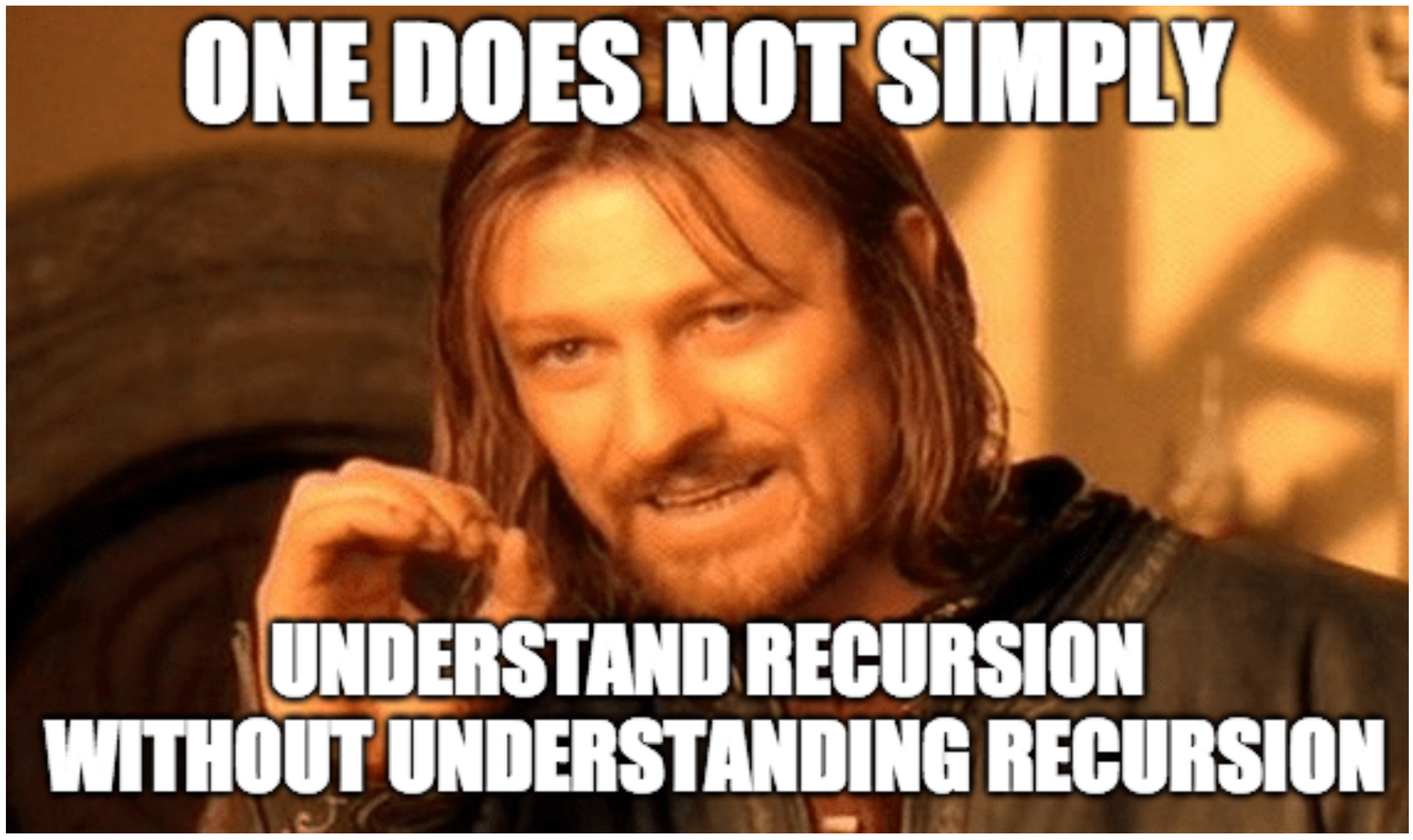
Time Complexity: O(n*2)

Auxiliary Space: O(1)

Boundary Cases: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

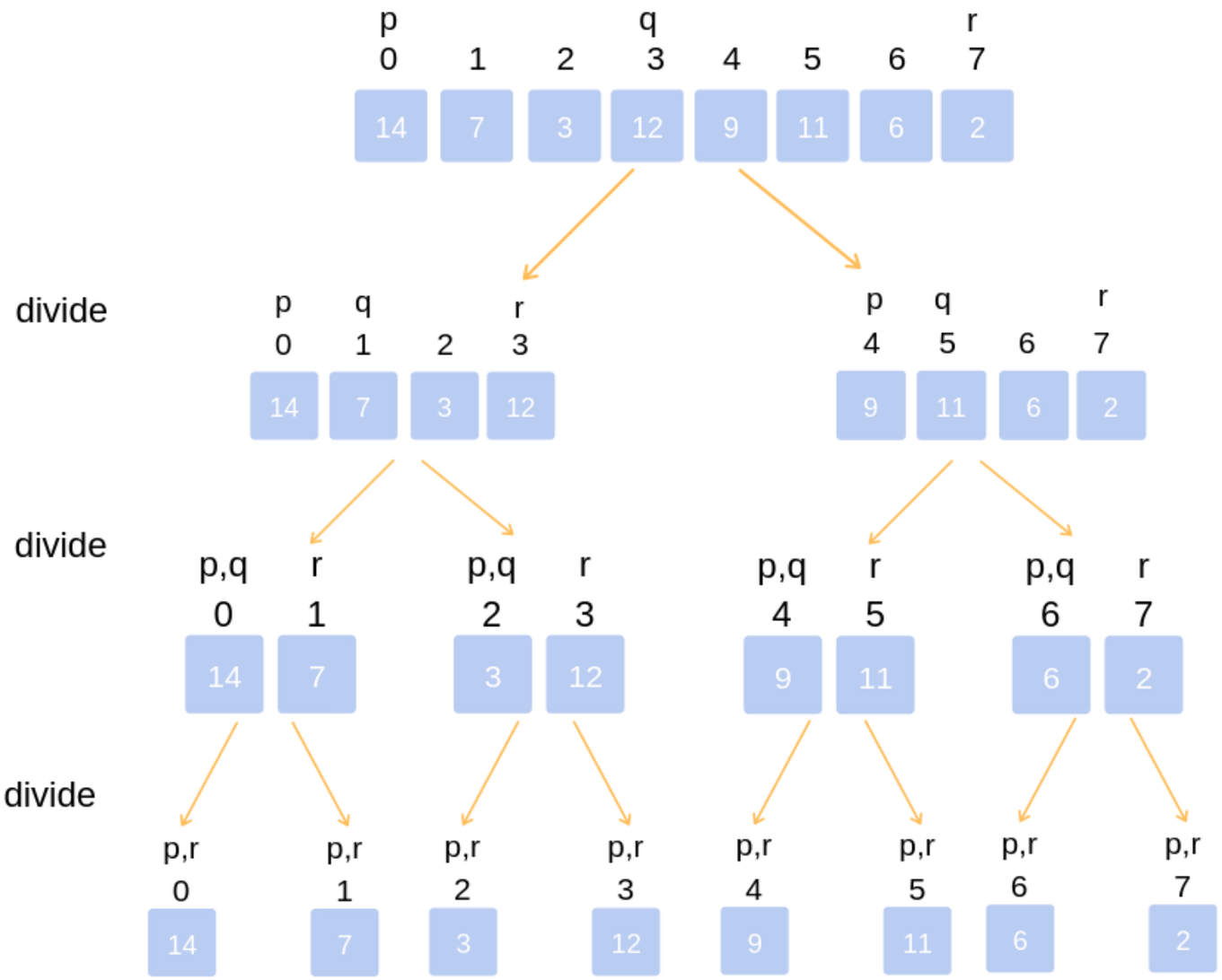
Sorting In Place: Yes

Recursion…



What is Stack Overflow…?

Merge Sort



Worst complexity: n*log(n)

Average complexity: n*log(n)

Best complexity: n*log(n)

Space complexity: n

Method: Merging

Stable: Yes

$$T_n = 2T_{n/2} + n - 1$$

Quick Sort

Time taken by QuickSort in general can be written as following.

$$T(n) = T(k) + T(n-k-1) + \theta(n)$$

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

$$T(n) = T(0) + T(n-1) + \theta(n)$$

which is equivalent to

$$T(n) = T(n-1) + \theta(n)$$

The solution of above recurrence is $\theta(n^2)$.

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

$$T(n) = 2T(n/2) + \theta(n)$$

The solution of above recurrence is θ(nLogn).

Solving divide and conquer recurrences

The Akra-Bazzi Formula to the rescue!

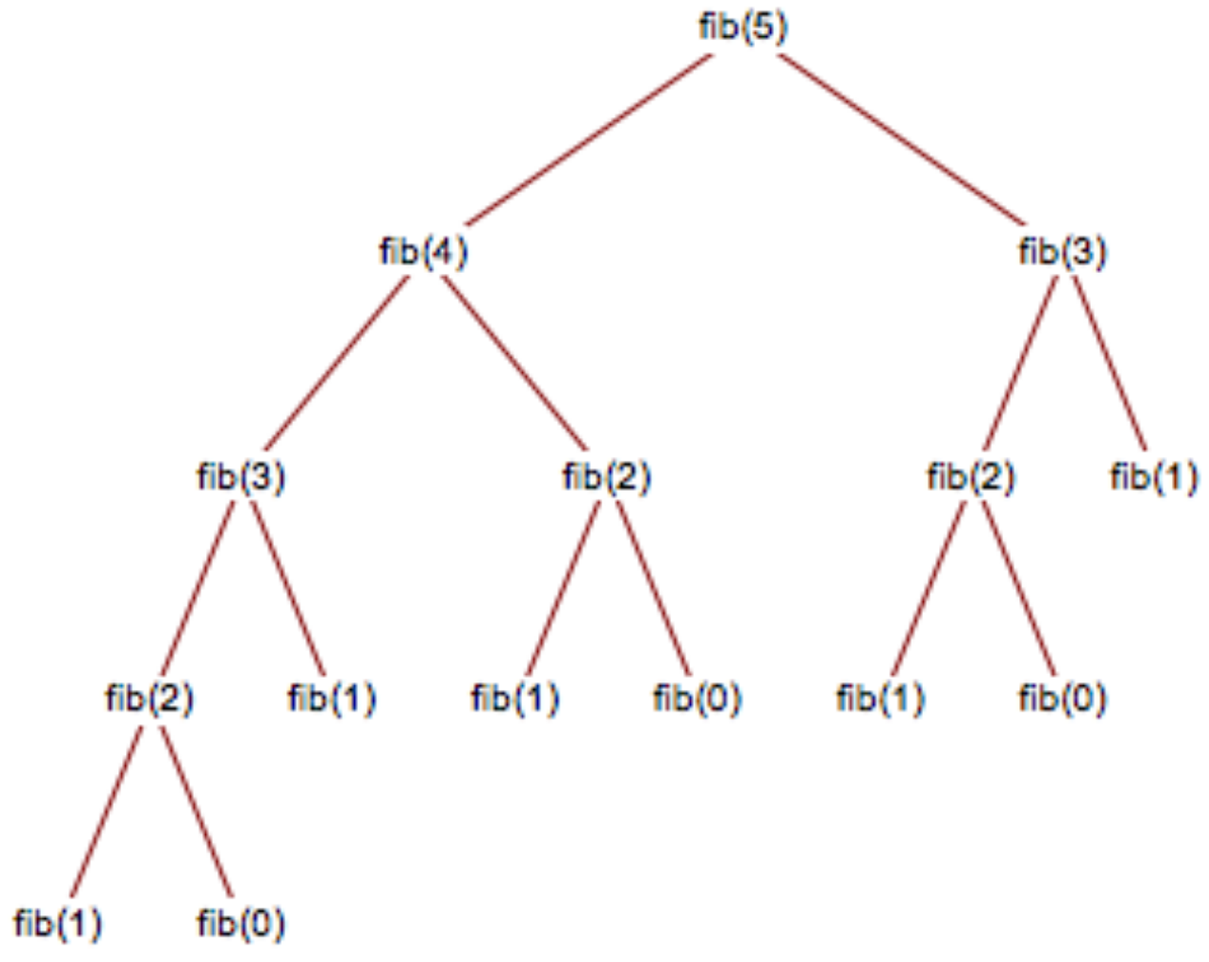
$$T(n) = \sum_{i=1}^{k} a_i T(b_i n) + g(n)$$

$$T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}}\, du\right)\right)$$

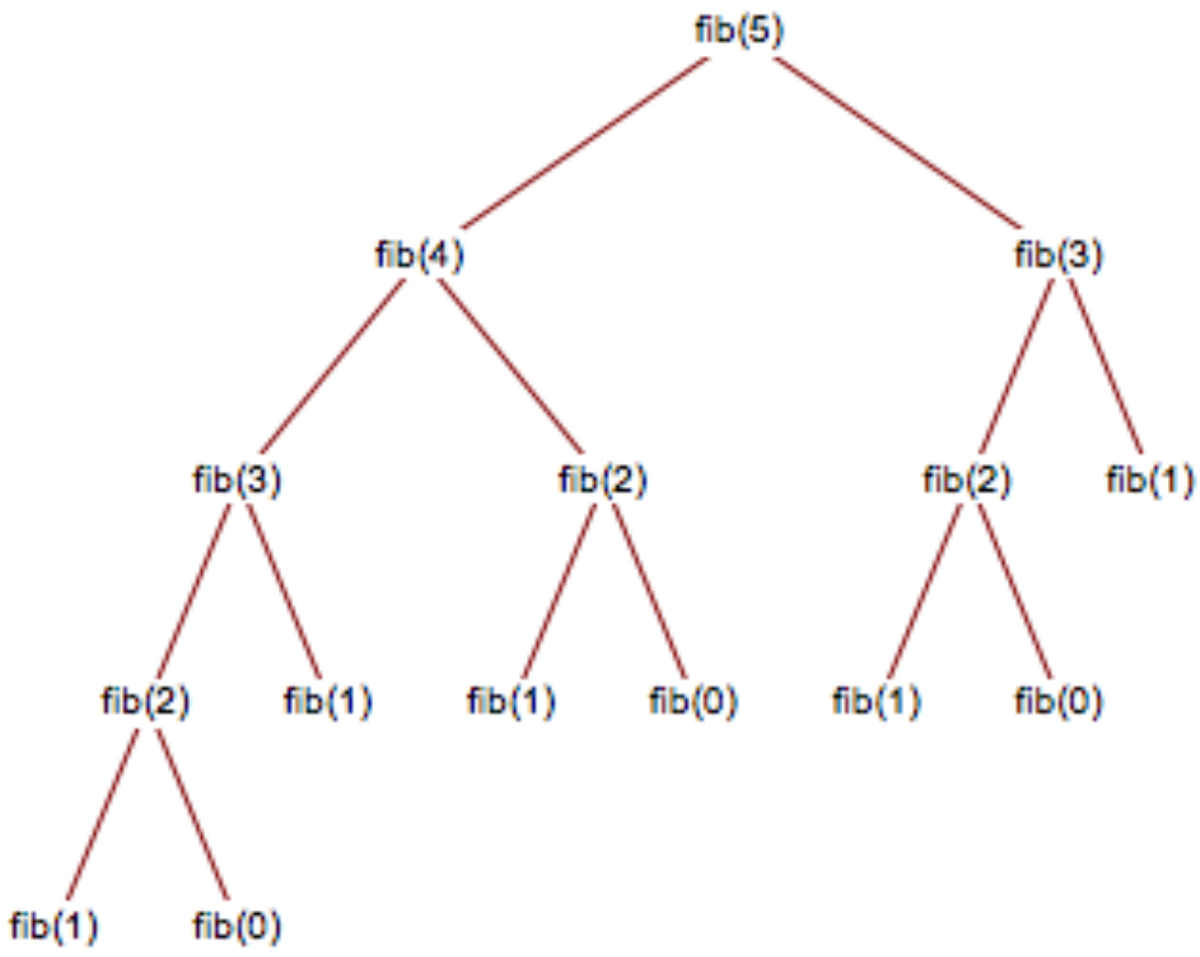
Advanced Problems

- T(N) = 2T(N/2) + NLogN
- T(N) = 2T(N/2) + (8/9)T(3N/4) + N^2
- T(N) = 3T(N/3) + 4T(N/4) + N^2

Space Complexity of Recursive algorithms?



Fibonacci using Recursion



$$F(n) = F(n-1) + F(n-2)$$

$$T(n) = O(1.6180)^n$$

Solving Linear Recurrences

Recursive vs Iterative approach!

When to use which one?

NP complete problems when asked to run in polynomial time