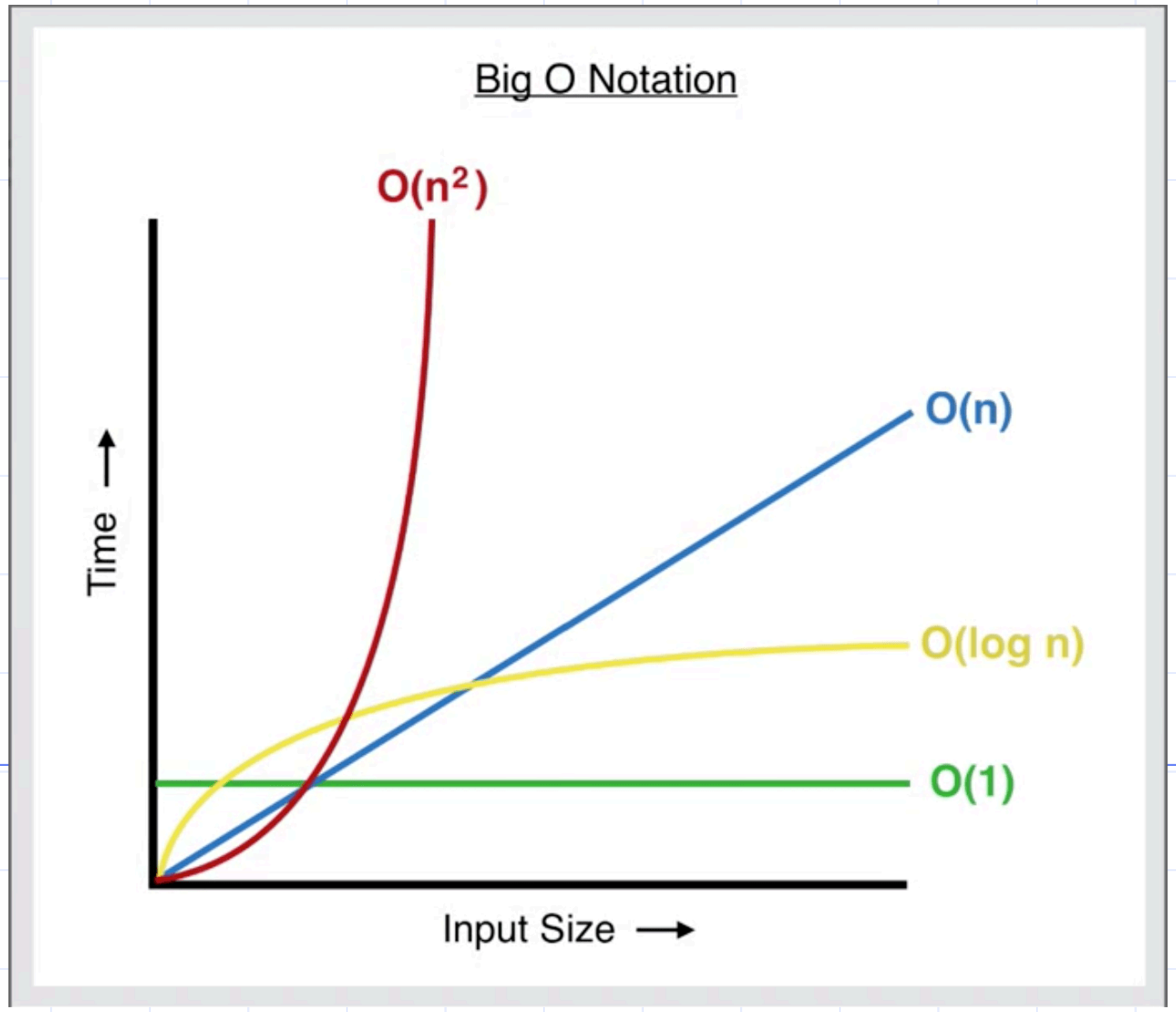
Time Complexity of Algorithms



Time Complexity

- Total amount of time an algorithm requires to complete
- Simplest case: **constant time complexity**
  - algorithm takes same amount of time, no matter the input size

```
int sumFirstTwo(int arr[], int n) //n=currSize
{
  return arr[0]+arr[1]; // always 1+ and 1 ret
}
```

Constant Time Complexity

- another constant time complexity example:

```
int sumFirst10(int arr[], int n)
{
  int sum=0; // 1 init
  for(int i=0; i<10; i++)// 1 init, 10 <, 10 ++
    sum+=arr[i]; // always 1 += (done 10 times)
  return sum; // 1 ret
}
```

- total units of work: 1 + 1 + 10 + 10 + 10 + 1 = 33
  - still constant !!!!

Linear Time Complexity

- another constant time complexity example:

```
int sumAll(int arr[], int n)
{
  int sum=0; // 1 init
  for(int i=0; i<n; i++)// 1 init, n <, n ++
    sum+=arr[i]; // always 1 += (done n times)
  return sum; // 1 ret
}
```

- total units of work: 1 + 1 + n + n + n + 1 = 3n + 3
  - **NOT** constant, but is linear in n (size of input)

Quadratic Time Complexity

```
int sumDist(int arr[], int n)
{
  int sum=0; // 1 init
  for(int i=0; i<n; i++)// 1 init, n <, n ++
    for(int j=0; j<n; j++)// (1 init, n <, n ++) n times
     sum = sum + abs(arr[i]-arr[j]); //(1+=, 1-, 1+) n² times
  return sum; //1 ret
}
```

- total: $1 + (1+n+n) + (n+n^2+ n^2) + (n^2 + n^2 + n^2) + 1$
- … $= 5n^2 + 3n + 3$
  - **NOT** linear or constant, but quadratic in n

Which Complexity is best?

- Which of these run times (time complexities) is shortest?
  - $2n^2 + 1024n + 6872$
  - $4n^2 - 4n + 5$
  - $9000n + 987654$
- answer: it depends on n !!!!
  - consider n=500, 520, 540 , 4097, 4107, 4117
  - remember, n could be an array size!

Which Complexity is best?

n	9000n+987654	4n2-4n+5	2n2+1024n+6872
520	5,667,654	1,079,525	1,080,152
521	5,676,654	1,083,685	1,083,258
522	5,685,654	1,087,855	1,086,368
…	…	…	…
4107	37,950,654	67,453,373	37,947,338
4108	37,959,654	67,486,229	37,964,792

Asymptotic Complexity

- Turns out that, in practice, the following two are essentially equivalent:
  - $2n^2 + 1024n + 6872$
  - $4n^2 - 4n + 5$

- Why? because we model them with **big-O**, i.e. O(f(n)):
  - $g(n) \in O(\ f(n)\ )$ i.e. g(n) is in big-O of f(n) iff:
    - for constants c and n_0 :
    - $\forall\ n > n_0,\ g(n) < c\ f(n)$

Asymptotic Complexity …

- Turns out that, in practice, the following two are equivalent:
  - $2n^2 + 1024n + 6872$, which is $O(n^2)$
  - $4n^2 - 4n + 5$, which is also $O(n^2)$

- Which means that both increase in run time at a rate of no more than cn^2
- $9000n + 987654$ is $O(n)$, technically also $O(n^2)$
- $O(n)$ is better than $O(n^2)$, provided the bound is tight

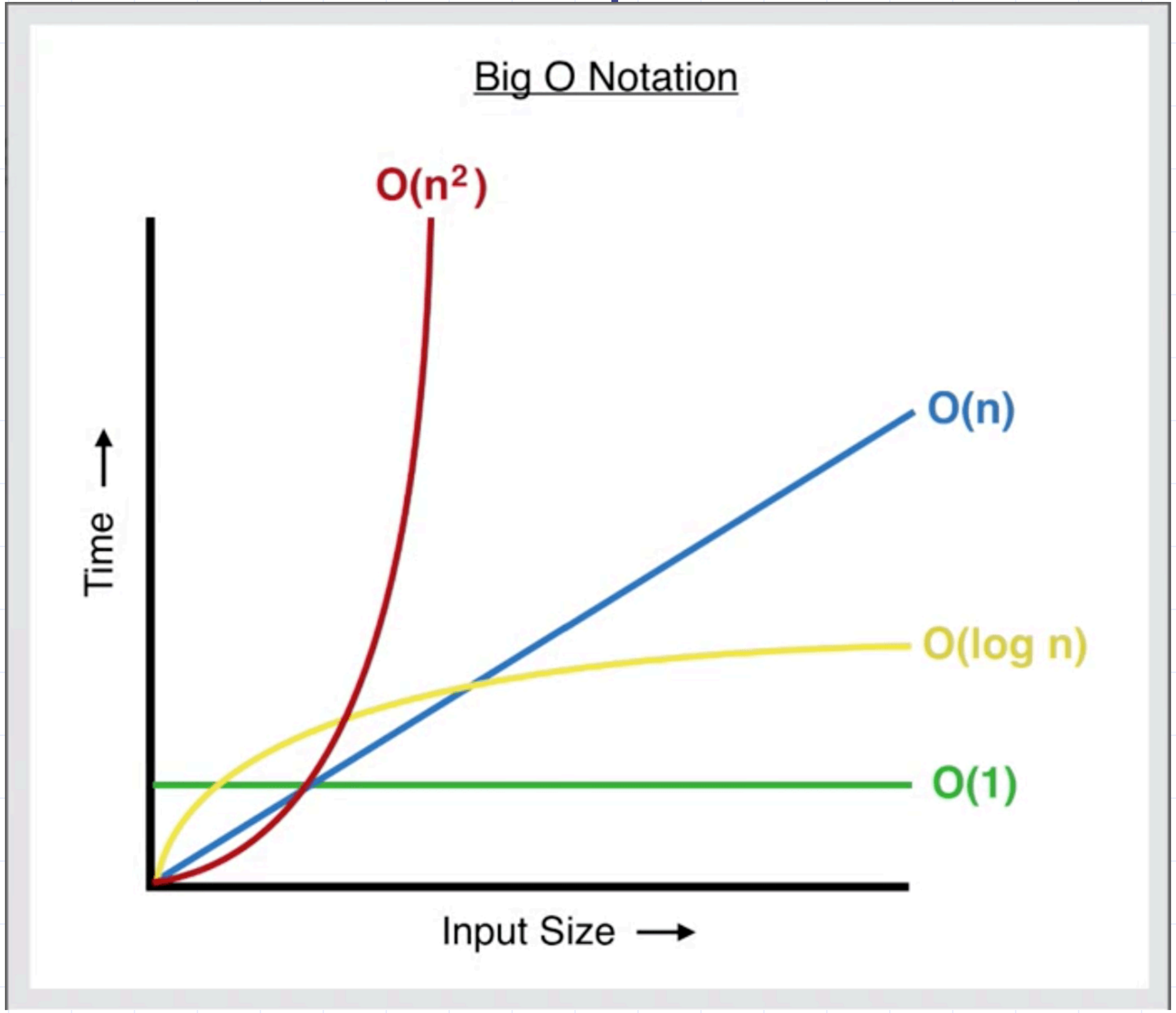
Which Complexity is best?

- Which of these run times (time complexities) is shortest?
  - $2n^2 + 1024n + 6872$ … $O(n^2)$
  - $4n^2 - 4n + 5$ … $O(n^2)$
  - $9000n + 987654$ … $O(n)$

- For large n (bigger program inputs)
  - $9000n + 987654$ is the best!

Asymptotic Complexity …

- Some common run times plotted:



Asymptotic Complexity Matters!

- Consider a dual core 2.5GhZ CPU (5GhZ "total")

n	log n	n log n	n^2
10 $2*10^{-9}$ sec	3.32 $6*10^{-10}$ sec	33.2 $6*10^{-9}$ sec	100 $2*10^{-8}$ sec
10^2 $2*10^{-8}$ sec	6.64 $1.2*10^{-9}$ sec	666.4 $13*10^{-8}$ sec	10000 $2*10^{-6}$ sec
10^3 $2*10^{-7}$ sec	9.96 $2*10^{-9}$ sec	9960 $19*10^{-7}$ sec	1000000 $2*10^{-4}$ sec
10^4 $2*10^{-6}$ sec	13.28 $2.6*10^{-9}$ sec	132800 $27*10^{-6}$ sec	10^8 $2*10^{-2}$ sec
10^9 2 sec	29.8 $6*10^{-9}$ sec	$29.8*10^9$ 60 sec	10^{18} 2315 days!

> 6 years!!!!

Asymptotic Analysis Example

```
int sumDist(int arr[], int n)
{
  int sum=0; // O(1)
  for(int i=0; i<n; i++)// O(n)
    for(int j=0; j<n; j++)// O(n), done O(n) times
     sum = sum + abs(arr[i]-arr[j]); //O(1), done O(n²)times
  return sum; //O(1) ret
}
```

- total: $O(1) + O(n) + O(n^2) + O(n^2) + O(1)$
- … $= O(n^2)$ // note abuse of = in notation, should really be $\in$

Asymptotic Analysis Example

```
int maxDist(int arr[], int n)
{
  int big=0; // O(1)
  for(int i=0; i<n; i++)// O(n)
    for(int j=0; j<n; j++)// O(n), done O(n) times
     if (abs(arr[i]-arr[j]) > big) //O(1), done O(n²)times
       big = abs(arr[i]-arr[j]); //O(1), done O(n²)times
  return big; //O(1) ret
}
```

- total: $O(1) + O(n) + O(n^2) + O(n^2) + O(n^2) + O(1)$
- … $= O(n^2)$

Asymptotic Analysis Example

```
int sumDist(int arr[], int n)
{
  int sum=0;
  for(int i=0; i<n; i++)
    for(int j=0; j<i; j++)
     sum = sum + abs(arr[i]-arr[j]);
  return sum;
}
```

- total: $O(1) + O(n) + O(n^2) + O(n^2) + O(1)$
- $O(n^2)$

Asymptotic Analysis Example

```
int binSearch(int target, int arr[], int n)
{ // note: we assume target is in arr
  int left=0, right=n-1, middle=n/2;
  while (arr[middle]!=target) {
    if (arr[middle] > target) right=middle-1;
    else left = middle+1;   //cut out left half
    middle= (left+right)/2;//cut out right half
  return middle;
}
```

- total: O(1) + O(log n) = O(log n)

Asymptotic Analysis Example

```
int binSearch(int target, int arr[], int n)
{ // note: we assume target is in arr
  int left=0, right=n-1, middle=n/2;
  while (arr[middle]!=target) {
    if (arr[middle] > target) right=middle-1;
    else left = middle+1;   //cut out left half
    middle= (left+right)/2;//cut out right half
  return middle;
}
```

- $O(\log_2 n)$

Other Asymptotic Complexity

- $g(n) \in O(f(n))$ represents an "upper bound"
- $g(n) \in \Omega(f(n))$ represents a "lower bound"
  - for constants c and n_0 :
  - $\forall\ n > n_o,\ g(n) > c\ f(n)$
- $g(n) \in \Theta(f(n))$ represents a "tight bound"
  - $g(n) \in O(f(n))$ **and** $g(n) \in \Omega(f(n))$
  - not always possible to find this!