DANIEL GRAHAM PHD

ALGORITHMS

WHY STUDY ALGORITHMS?

FOR INTELLECTUAL STIMULATION

"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.."

– FRANCIS SULLIVAN (THE JOY OF ALGORITHMS)

TO BECOME A PROFICIENT PROGRAMMER.

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he[she] considers his[her] code or his[her] data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

– LINUS TORVALDS (CREATOR OF LINUX)

THEY MAY UNLOCK THE SECRETS OF LIFE AND OF THE UNIVERSE.

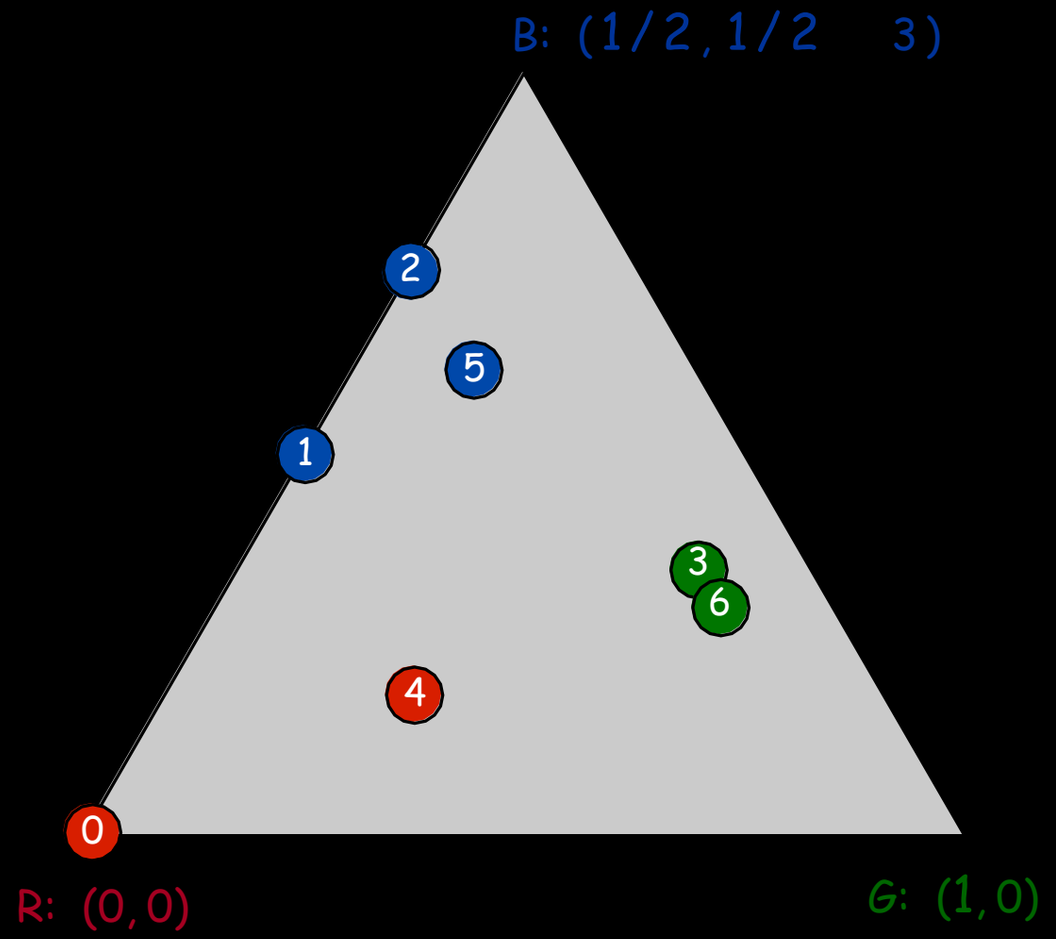
"Computer models mirroring real life have become crucial for most advances made in chemistry today.... Today the computer is just as important a tool for chemists as the test tube."

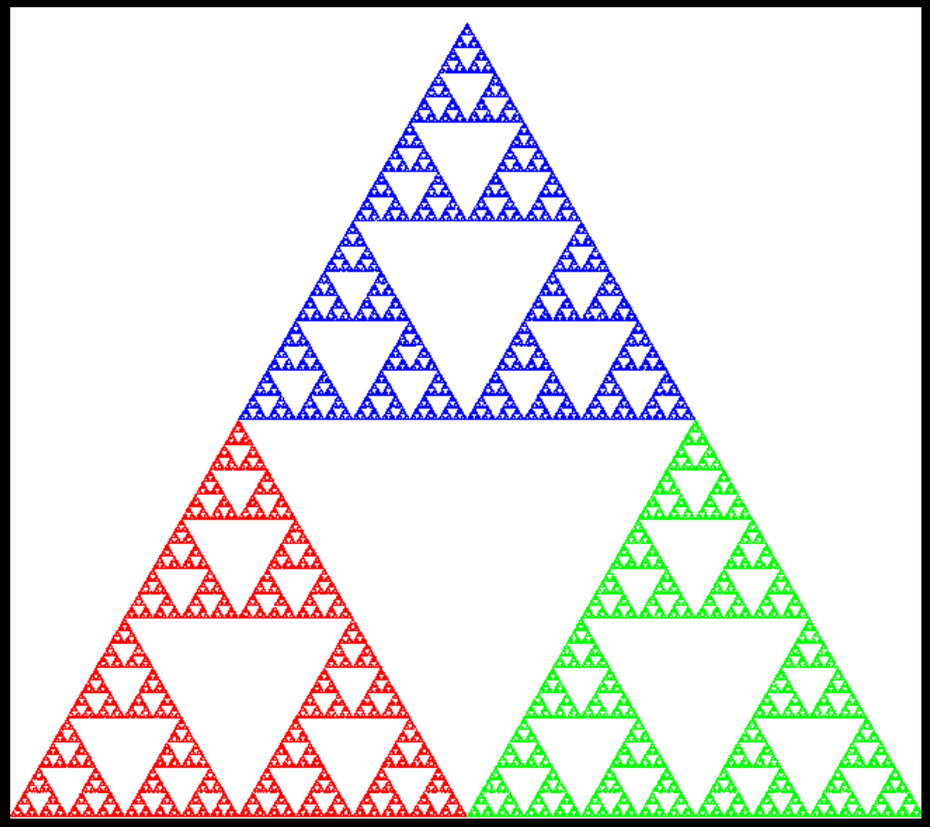
– DURING A NOBEL PRIZE SPEACH IN 2013

Chaos game. Play on equilateral triangle, with vertices R, G, B.

Start at R.
Repeat the following **N** times:
  pick a random vertex
  move halfway between current point and vertex
  draw a point in the color of the vertex

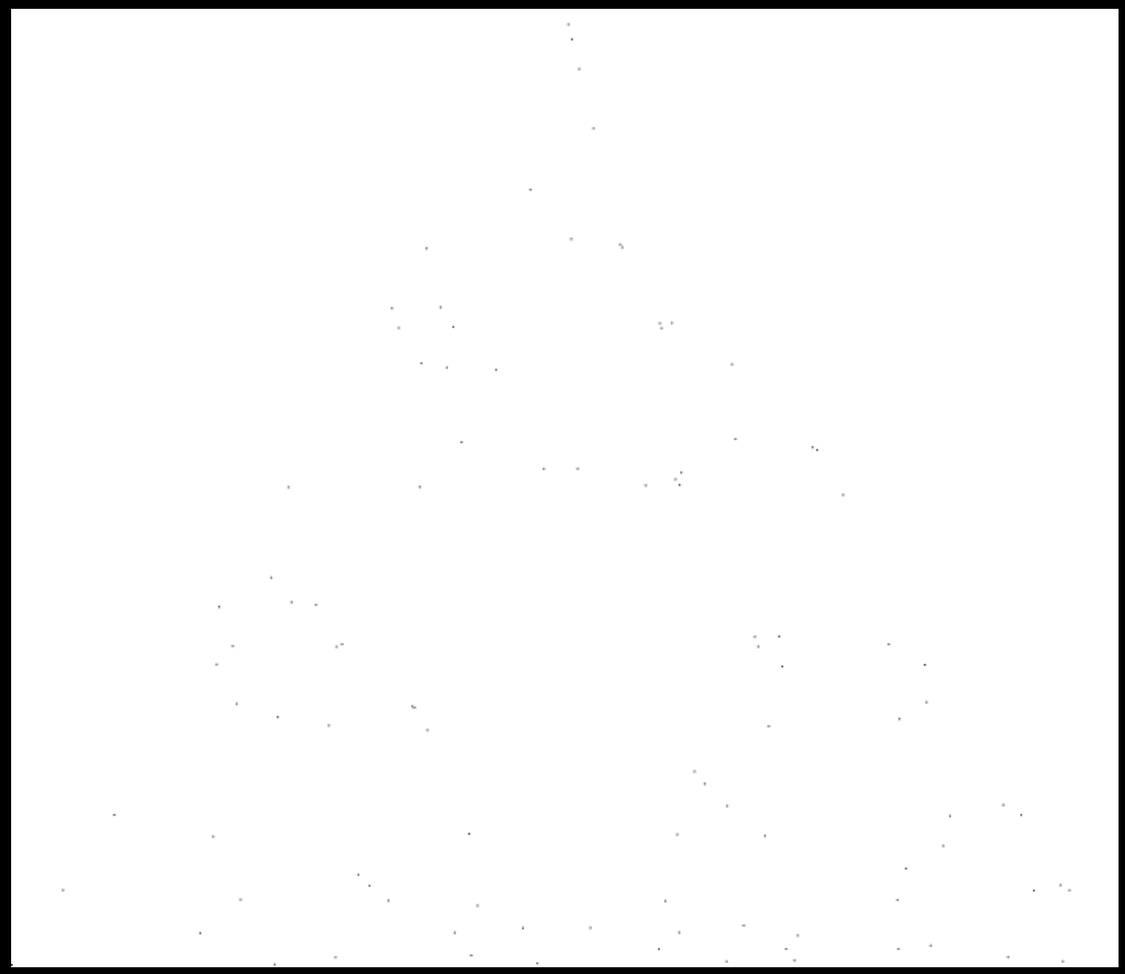
STARTING AT RED
GIVEN SEQUENCE:
B B G R B G





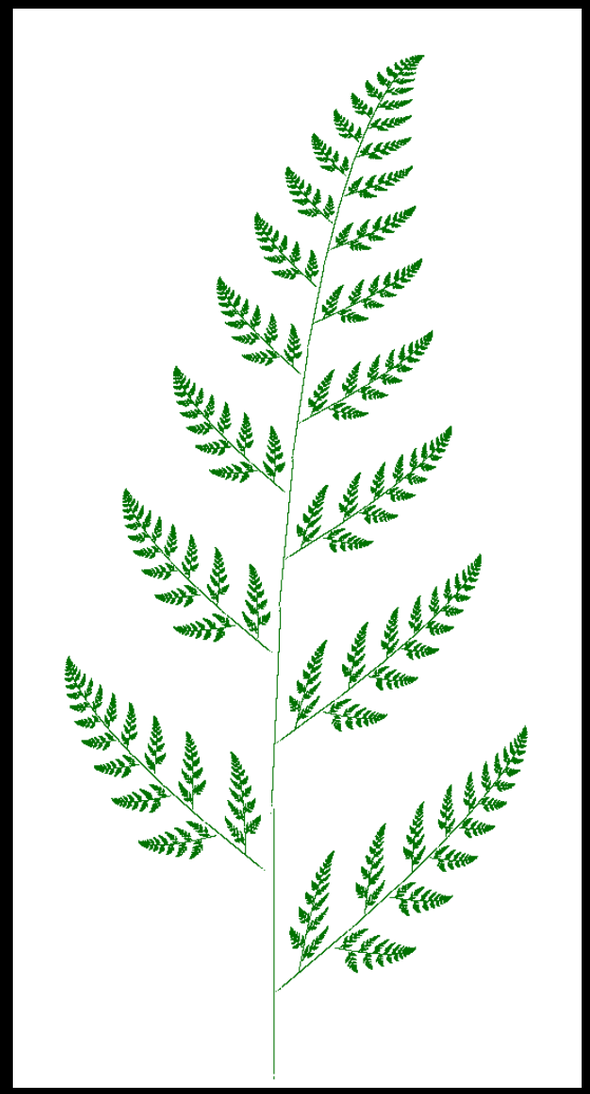
R G

Sierpinski triangle



- Chaos game with Different Rules

Probablity	new x	new y
2%	0.5	0.27y
15%	-.14x + .26y + .57	.25x + .22y - .04
13%	.17x - .21y + .41	.22x + .18y + .09
70%	.78x + .03y + .11	-.03x + .74y + .27



"Algorithms: a common language for nature, human, and computer."

–AVI WIGDERSON

FOR PROFIT

Google

Apple Computer

facebook

Cisco Systems

IBM

Nintendo

Jane Street

Morgan Stanley

NETFLIX

Adobe

RSA Security

D E Shaw & Co

ORACLE

Pandora

Akamai

YAHOO!

amazon.com

Microsoft

Pixar Animation Studios

LOGISTICS

Programming assignments. 50%

- Don’t use online solution (Honor Violation)
- Online Submission system. (More info to come)

Write Up. 10%

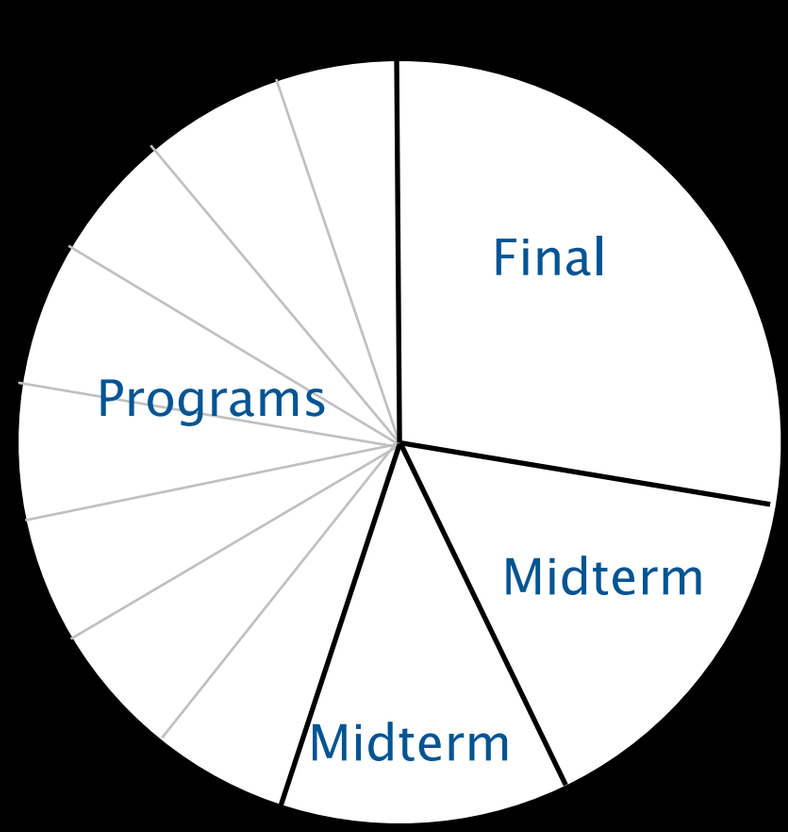
- Due on Thursday at 10pm on Collab.
- No late submission accepted.

Exams. 15% + 12.5%+ 12.5%

- Midterm x2 (in class).
- Final (to be scheduled by Registrar).

Staff discretion.

- Contribute to Piazza discussion forum.
- Attend and participate in precept/lecture.



Collab discussion forum.

- Low latency, low bandwidth.

Office hours.

- High bandwidth, high latency.
- See web for schedule using system written by Max

Download the Project Portfolio.

- Homework Problems will be upload to a git repository.
- Every week you will need to pull from this repository to get that weeks homework problems.

Monday and Wednesday

4pm - 6pm

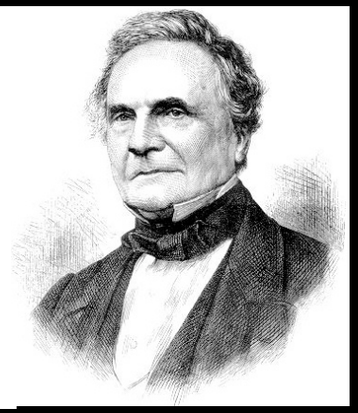
OFFICE HOURS ARE FOR EVERYONE
COME BY EVEN IF YOU WANT TO CHAT

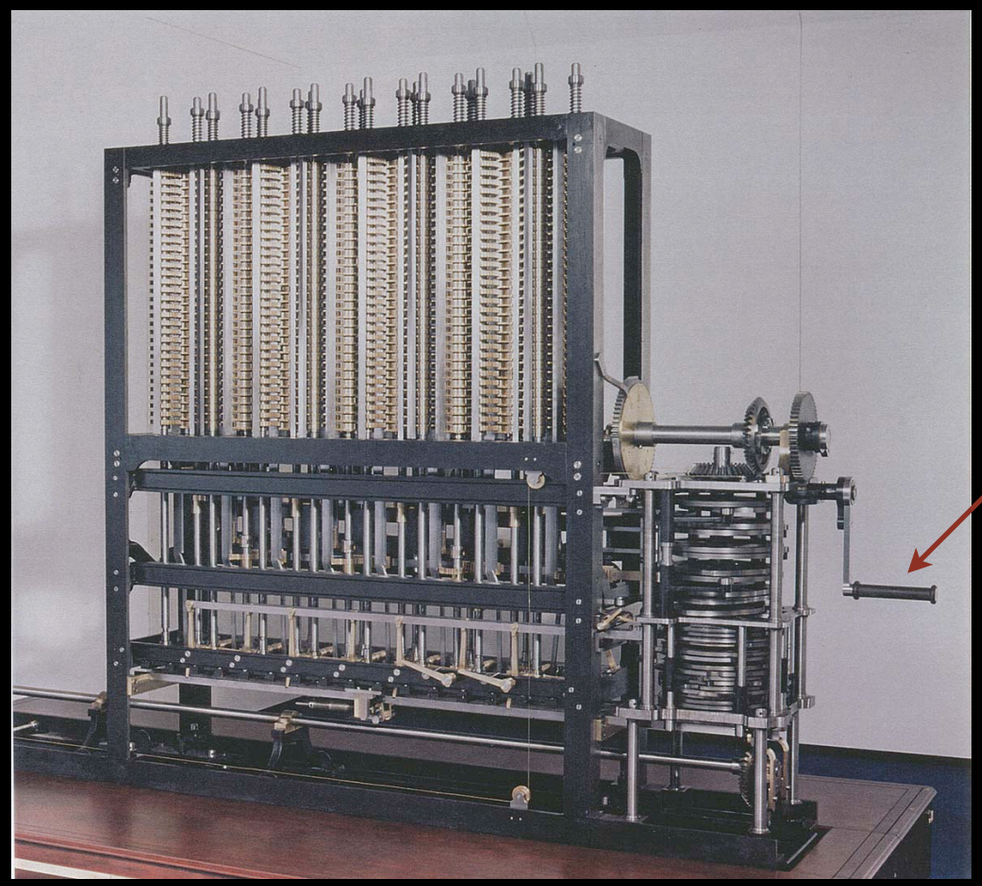
Is my algorithm better than yours?

How long will it take to run?

How much memory will it use?

" As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time? " — *Charles Babbage (1864)*





how many times do you have to turn the crank?

Ada Lovelace The First Programer of the Analytic Engine

APPROACH ONE EMPIRICAL

- Let's apply an scientific method:
  - Observe: Run the program and time it.
  - Hypothesize: Create a model for the behavior
  - Predict: Use the model to determine value in the future
  - Verify: Apply model to future value check to make sure that predictions agree

- Let’s apply this empirical approach to the three sum problem.

3-SUM. Given *N* distinct integers, how many triples sum to exactly zero?

```
% more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

% java ThreeSum 8ints.txt
4
```

	a[i]	a[j]	a[k]	sum
1	30	-40	10	0
2	30	-20	-10	0
3	-40	40	0	0
4	-10	0	10	0

ANY THOUGHTS ON HOW WE MIGHT SOLVE THIS?

BRUT FORCE SOLUTION

```
public class ThreeSum
{
   public static int count(int[] a)
   {
      int N = a.length;
      int count = 0;
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
               if (a[i] + a[j] + a[k] == 0)
                  count++;
      return count;
   }

   public static void main(String[] args)
   {
      In in = new In(args[0]);
      int[] a = in.readAllInts();
      StdOut.println(count(a));
   }
}
```

check each triple

for simplicity, ignore integer overflow

IS THIS GOOD SOLUTION

- Let's apply an scientific method:
  - Observe: Run the program and time it.
  - Hypothesize: Create a model for the behavior
  - Predict: Use the model to determine value in the future
  - Verify: Apply model to future value check to make sure that predictions agree

HOW DO WE TIME THE ALGORITHM?

Q. How to time a program?

A. Stop WATCH

USES CLOCK IN THE SYSTEM

public class Stopwatch	
Stopwatch()	*create a new stopwatch*
double elapsedTime()	*time since creation (in seconds)*

```
public class Stopwatch
{
   private final long start = System.currentTimeMillis();

   public double elapsedTime()
   {
      long now = System.currentTimeMillis();
      return (now - start) / 1000.0;
   }
}
```

implementation (part of stdlib.jar)

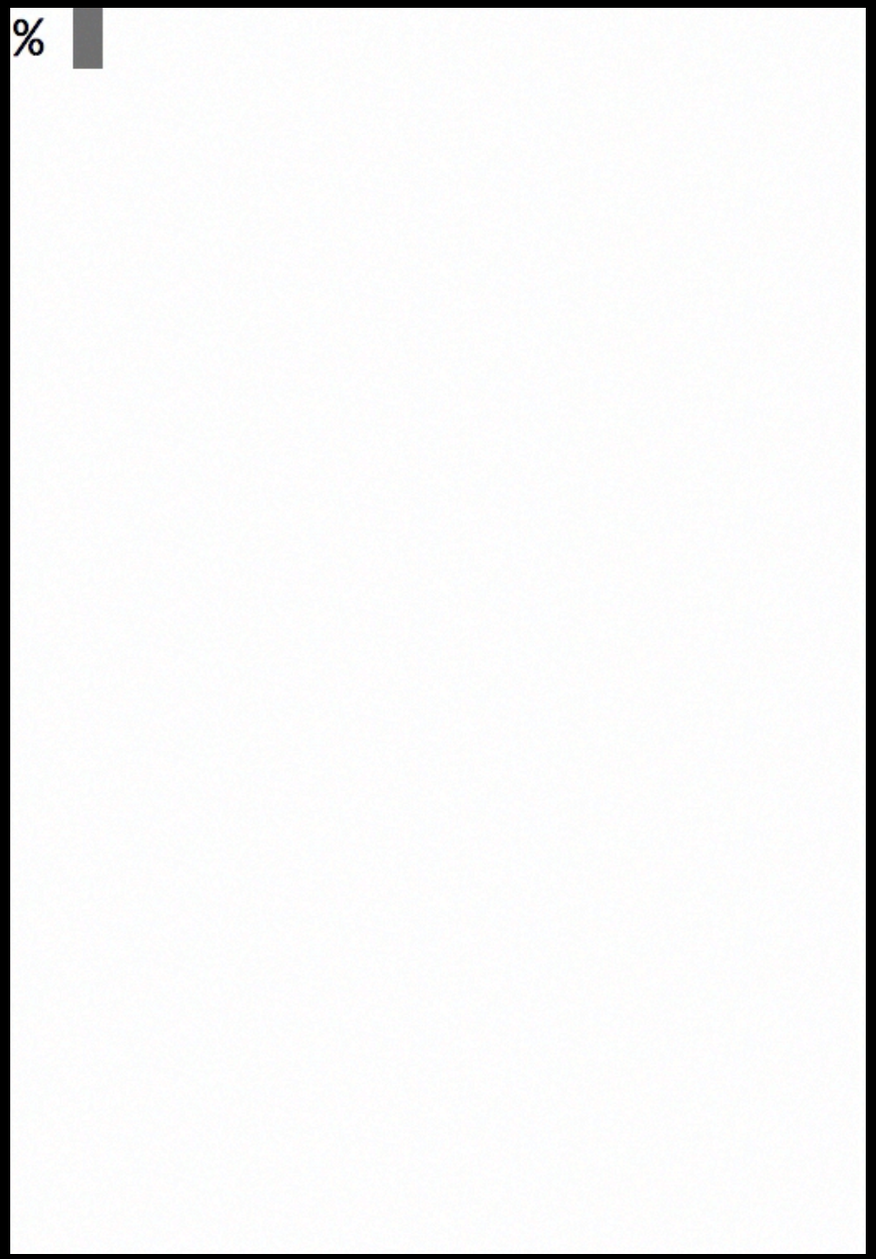
Q. How to time a program

A. Automatic.

public class Stopwatch	(part of stdlib.jar)
Stopwatch()	*create a new stopwatch*
double elapsedTime()	*time since creation (in seconds)*

```
public static void main(String[] args)
{
   In in = new In(args[0]);
   int[] a = in.readAllInts();
   Stopwatch stopwatch = new Stopwatch();
   StdOut.println(ThreeSum.count(a));
   double time = stopwatch.elapsedTime();
   StdOut.println("elapsed time " + time);
}
```

RUN PROGRAM ON DIFFERENT INPUT SIZES

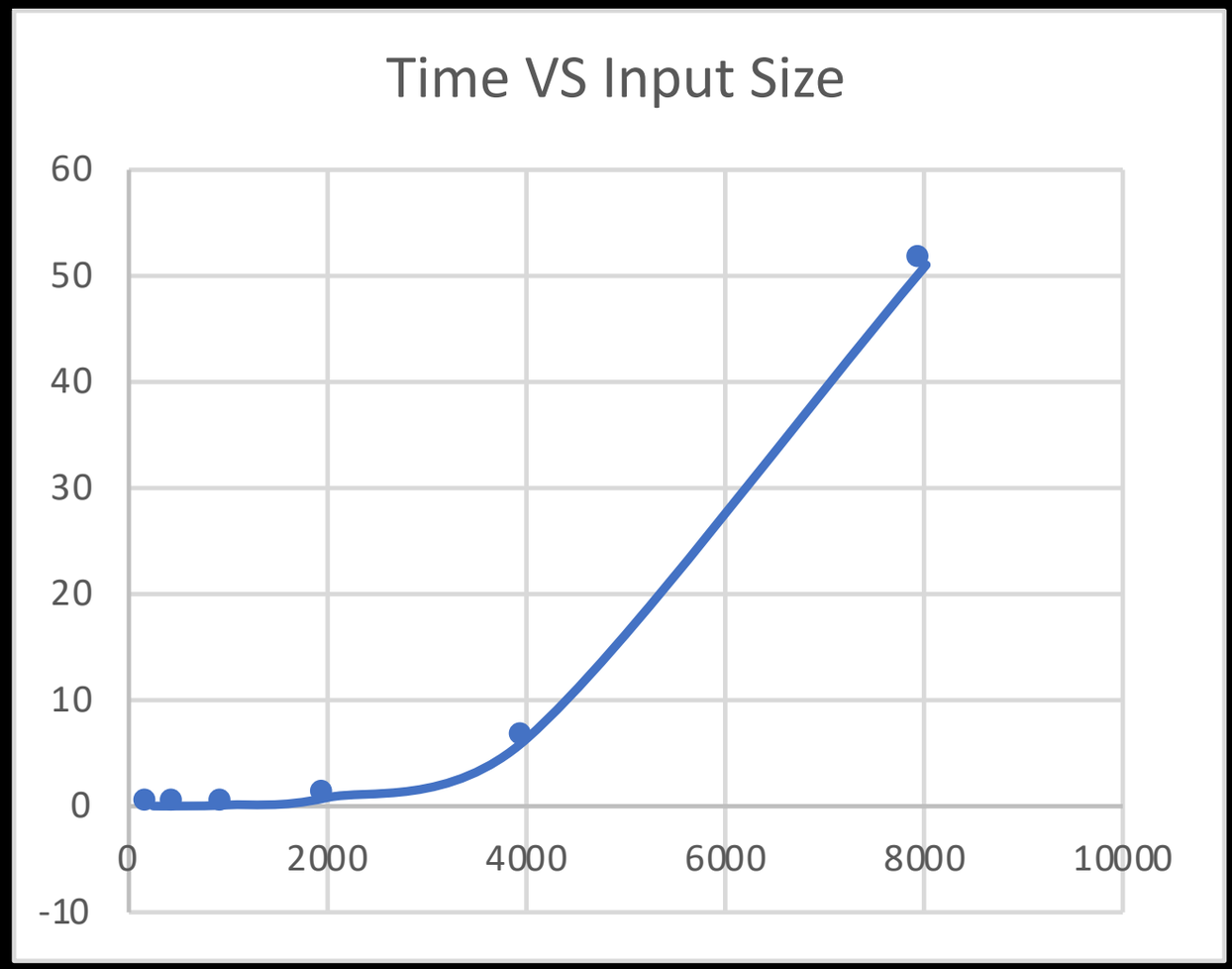


MEASURE THE TIME

N	time (seconds) †
250	0
500	0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

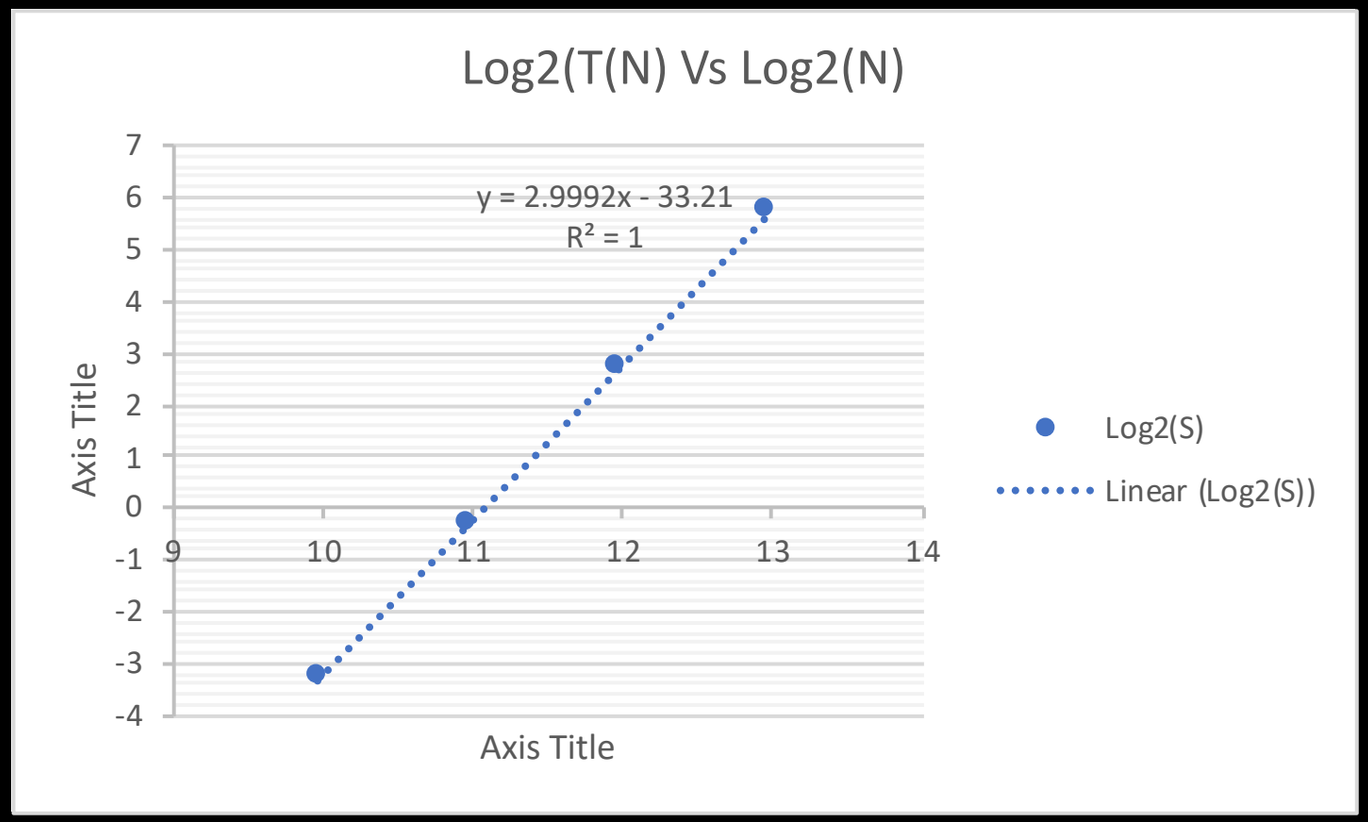
GRAPHING THE PERFROMANCE

N	time (seconds) †
250	0
500	0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?



CREATING A MODEL

N	T(N)		LOG2(N)	LOG2(T(N))
1,000	0.1		9.96578428	-3.3219281
2,000	0.8		10.9657843	-0.3219281
4,000	6.4		11.9657843	2.67807191
8,000	51.1		12.9657843	5.67525139



$\lg(T(N)) = b \lg N + c$

$b = 2.999$

$c = -33.2103$

Raise both sides to the power of 2

$T(N) = a N^b$, *where* $a = 2^c$

Hypothesis. The running time is about $1.006 \times 10^{-10} \times N^{2.999}$ seconds.

$2^{-33.2102} = 1.006 \times 10^{-10}$

IS THE A MORE EFFICIENT WAY

TO GET A & B CONSTANTS

WHAT ABOUT PROGRAMS THAT DON'T FIT THE POWER LAW

How well does the model do?

Hypothesis. The running time is about $1.006 \times 10^{-10} \times N^{2.999}$ seconds.

"order of growth" of running time is about N^3 [stay tuned]

Predictions.

- 51.0 seconds for $N = 8{,}000$.
- 408.1 seconds for $N = 16{,}000$.

Observations

N	time (seconds) †
8,000	51.1
8,000	51
8,000	51.1
16,000	410.8

MULTIPLE RUNS

validates hypothesis!

QUICK WAY TO ESTIMATE B

Doubling hypothesis. Quick way to estimate b in a power-law relationship.

Run program, doubling the size of the input.

N	time (seconds) †	ratio	lg ratio
250	0		–
500	0	4.8	2.3
1,000	0.1	6.9	2.8
2,000	0.8	7.7	2.9
4,000	6.4	8	3
8,000	51.1	8	3

$$\frac{T(2N)}{T(N)} = \frac{a(2N)^b}{aN^b} = 2^b$$

lg (6.4 / 0.8) = 3.0

seems to converge to a constant b ≈ 3

Hypothesis. Running time is about $a\,N^b$ with b = lg ratio.

QUICK WAY OF ESTIMATING

Doubling hypothesis. Quick way to estimate b in a power-law relationship.

Q. How to estimate a (assuming we know b) ?

A. Run the program (for a sufficient large value of N) and solve for a.

N	time (seconds) †
8,000	51.1
8,000	51
8,000	51.1

$$51.1 = a \times 8000^3$$

$$\Rightarrow \quad a = 0.998 \times 10^{-10}$$

Hypothesis. Running time is about $0.998 \times 10^{-10} \times N^3$ seconds.

EXPERIMENTAL APPROACH

System independent effects.

- Algorithm.
- Input data.

} determines exponent in power law

System dependent effects.

- Hardware: CPU, memory, cache, ...
- Software: compiler, interpreter, garbage collector, ...
- System: operating system

} determines constant in power law

CAN WE ESTIMATE THE PERFORMANCE & BUILD MODELS WITHOUT RUNNING THE PROGRAM

YES WE SHOULD BE ABLE TO BUILD MODELS FROM THE CODE ITSELF

Total running time: sum of cost × frequency for all operations.

- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.

In principle accurate mathematical models are available.

Donald Knuth
1974 Turing Award

HOW DO WE ESTIMATE COST

CHECK PROCESSOR SPECIFICATIONS FOR EACH OPERATION

operation	example	nanoseconds †
integer add	a + b	2.1
integer multiply	a * b	2.4
integer divide	a / b	5.4
floating-point add	a + b	4.6
floating-point multiply	a * b	4.2
floating-point divide	a / b	13.5
sine	Math.sin(theta)	91.3
arctangent	Math.atan2(y, x)	129
...	...	...

† Running OS X on Macbook Pro 2.2GHz with 2GB RAM

MOST PRIMITIVES TAKE CONSTANT TIME

operation	example	nanoseconds †
variable declaration	`int a`	c_1
assignment statement	`a = b`	c_2
integer compare	`a < b`	c_3
array element access	`a[i]`	c_4
array length	`a.length`	c_5
1D array allocation	`new int[N]`	$c_6\ N$
2D array allocation	`new int[N][N]`	$c_7\ N^2$

Other operations take constant time
scaled the input to the operation

SOME OPERATIONS DON'T CONSTANT TIME
NOVICE MISTAKE: ABUSIVE STRING CONCATENATION

IT IS POSSIBLE BUILD MODELS DIRECTLY FROM CODE

Q. How many instructions as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   if (a[i] == 0)
      count++;
```

N array accesses

operation	frequency
variable declaration	2
assignment statement	2
less than compare	$N + 1$
equal to compare	N
array access	N
increment	N to $2\,N$

5 N + 5

4 N + 5

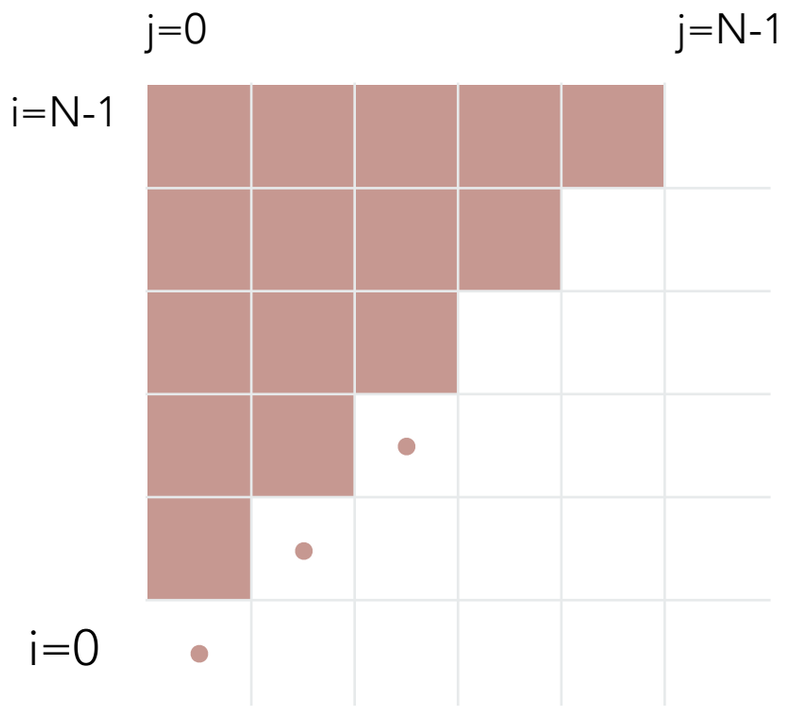
HOWEVER, BUILDING THESE MODELS CAN BE TEDIOUS

Specially as algorithms become
More complicated

CONSIDER THE 2-SUM PROBLEM

Q. How many instructions as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```



$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2}N^2 - \frac{1}{2}N$$

half of — half of

$$\begin{aligned} 0 + 1 + 2 + \ldots + (N-1) &= \frac{1}{2}N(N-1) \\ &= \binom{N}{2} \end{aligned}$$

CONSIDER THE 2-SUM PROBLEM

Q. How many instructions as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

operation	frequency
variable declaration	$N + 2$
assignment statement	$N + 2$
equal to compare	$\frac{1}{2} N (N - 1)$
array access	$N (N - 1)$
.....	.....

tedious to count

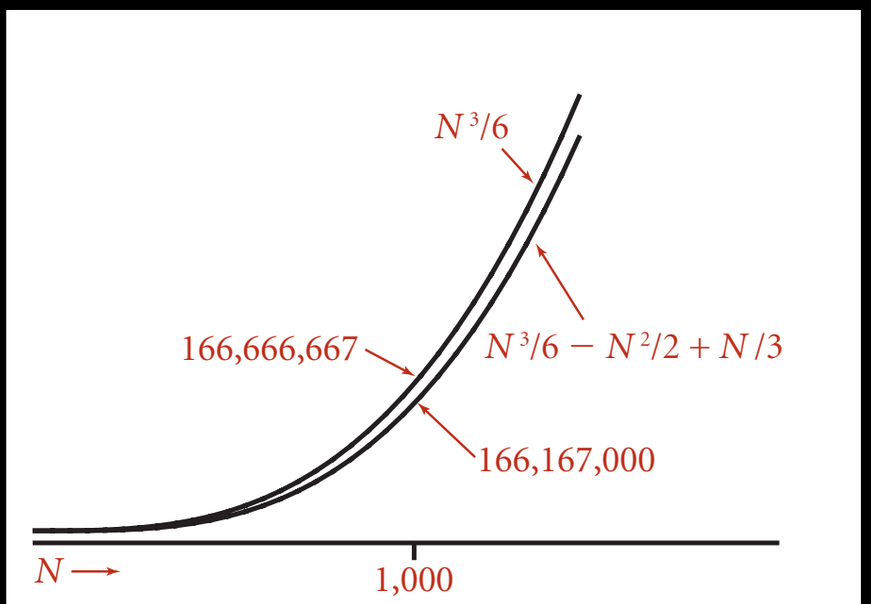
$$\frac{3}{2} N^2 - \frac{1}{2} N + 4$$

SIMPLIFYING LONGER EXPRESSIONS

- when N is large, terms are negligible
- when N is small, we don't care

Ex. $\frac{1}{6} N^3 - \frac{1}{2} N^2 + \frac{1}{3} N \sim \frac{1}{6} N^3$

discard lower-order terms
(e.g., N = 1000: 166.67 million vs. 166.17 million)



Leading-term approximation

~ SIMPLIFICATION

operation	frequency	tilde notation
variable declaration	$N + 2$	$\sim N$
assignment statement	$N + 2$	$\sim N$
equal to compare	$\frac{1}{2} N (N - 1)$	$\sim \frac{1}{2} N^2$
array access	$N (N - 1)$	$\sim N^2$

THREE SUM EXAMPLE

Q. Approximately how many array accesses as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

"inner loop"

A. $\sim \frac{1}{2} N^3$ array accesses.

$$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!} \sim \frac{1}{6} N^3$$

Bottom line. Use cost model and tilde notation to simplify counts.

WE CAN USE AN EVEN HIGHER LEVEL OF CLASSIFICATION

ORDER OF GROWTH CLASSIFICATION

ORDER OF GROWTH CLASSIFICATIONS

Definition. If $f(N) \sim c\, g(N)$ for some constant $c > 0$, then the order of growth of $f(N)$ is $g(N)$.

- Ignores leading coefficient.
- Ignores lower-order terms.

Ex. The order of growth of the running time of this code is N^2.

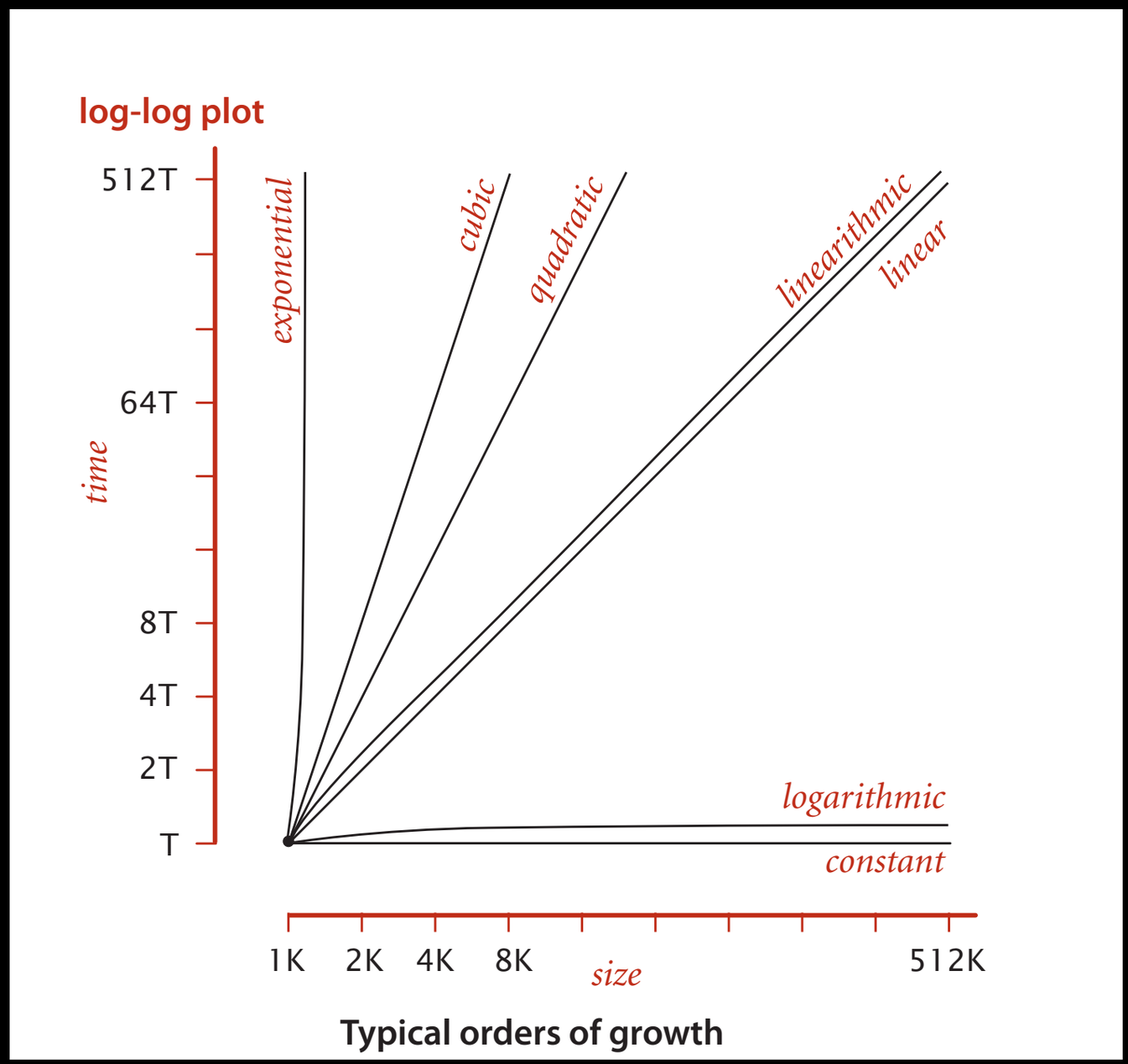
```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
        if (a[i] + a[j] == 0)
            count++;
```

ORDER OF GROWTH FUNCTION

Good news. The set of functions

$$1,\ \log N,\ N,\ N \log N,\ N^2,\ N^3, \text{ and } 2^N$$

suffices to describe the order of growth of most common algorithms.



Typical orders of growth

CLASSIFICATIONS OF ODORS OF GROWTH

order of growth	name	typical code framework	description	example	$T(2N)$ / $T(N)$
1	**constant**	`a = b + c;`	statement	add two numbers	1
$\log N$	**logarithmic**	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search	~ 1
N	**linear**	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum	2
$N \log N$	**linearithmic**	[we discuss more later: mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	**quadratic**	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }`	double loop	check all pairs	4
N^3	**cubic**	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }`	triple loop	check all triples	8
2^N	**exponential**	[combinatorial search]	exhaustive search	check all subsets	$T(N)$

QUIZ

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

Q. What is order of growth classification

Q. Can we do better?

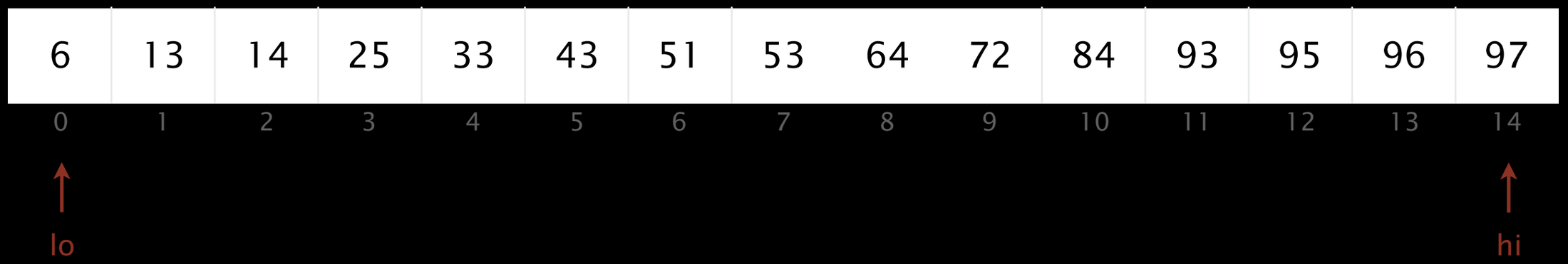
BINARY SEARCH DEMO

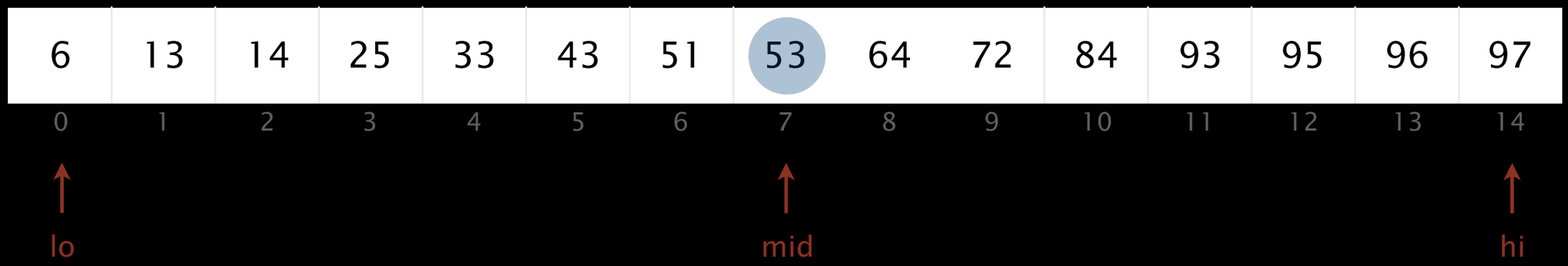
Goal. Given a sorted array and a key, find index of the key in the array?

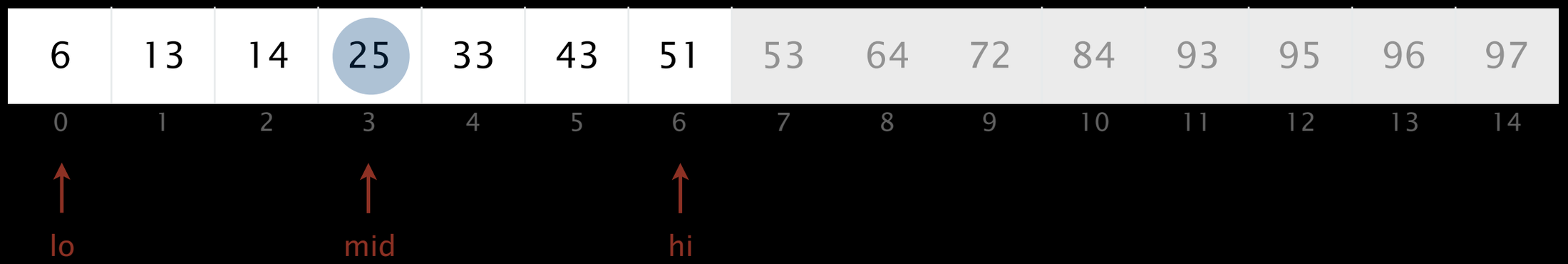
Binary search. Compare key against middle entry.

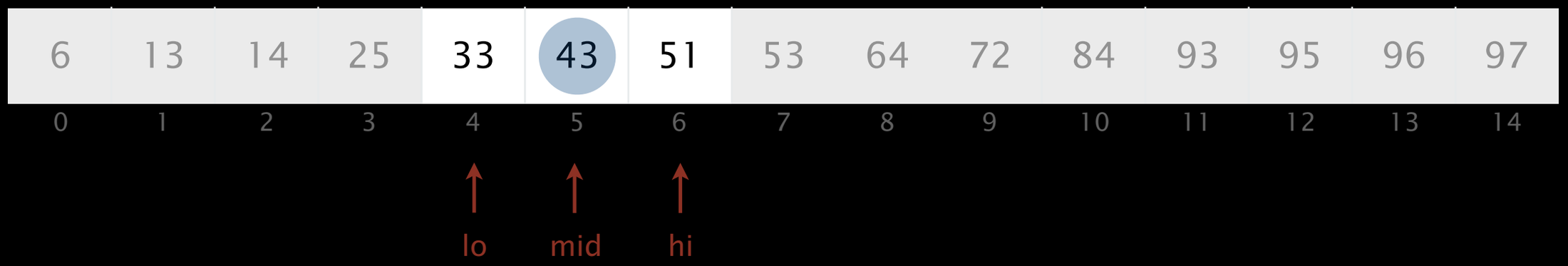
- Too small, go left.
- Too big, go right.
- Equal, found.

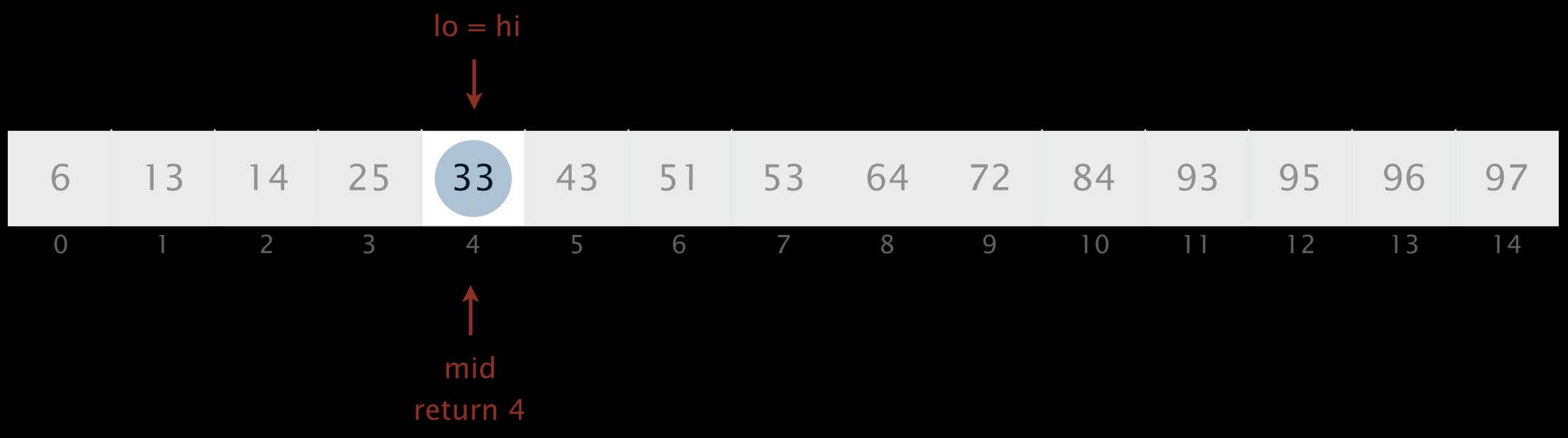
SUCCESSFUL SEARCH FOR 33



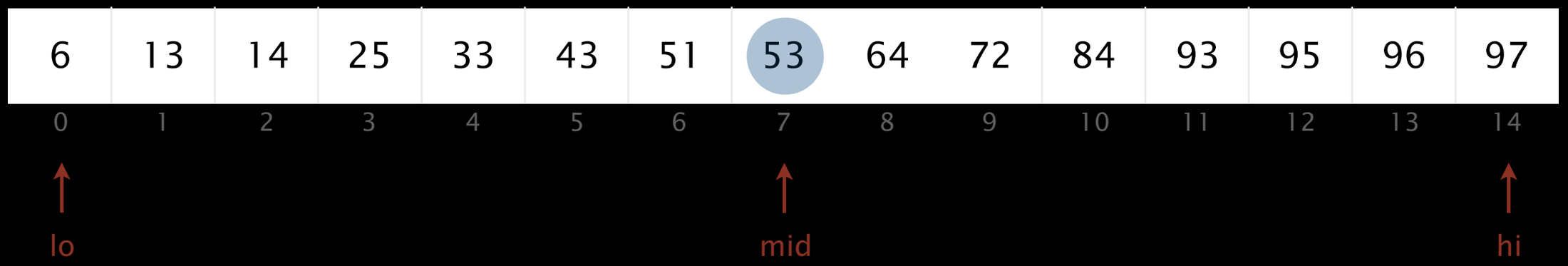

6
13
14
25
33
43
51
53
64
72
84
93
95
96
97
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
lo
mid
hi

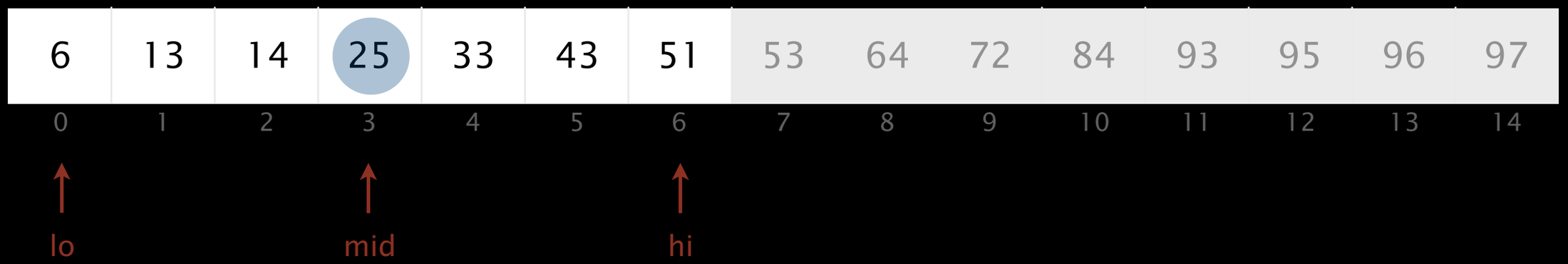

6
13
14
25
33
43
51
53
64
72
84
93
95
96
97
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
lo
mid
hi

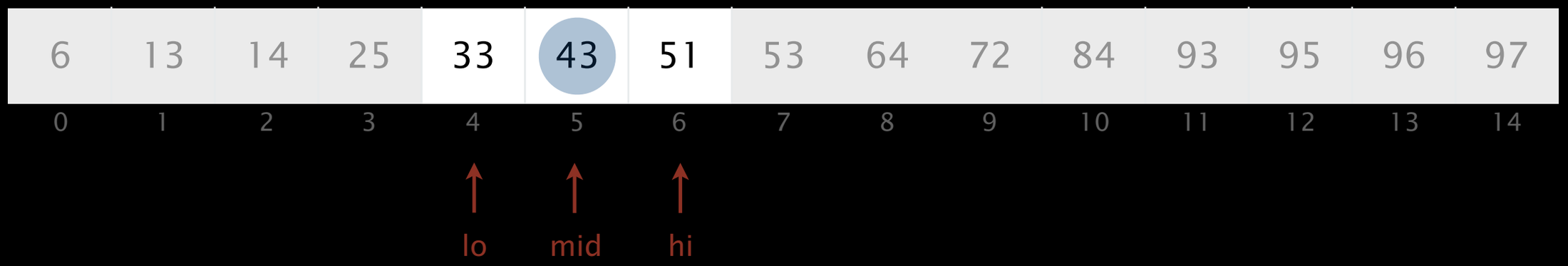

6
13
14
25
33
43
51
53
64
72
84
93
95
96
97
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
lo
mid
hi

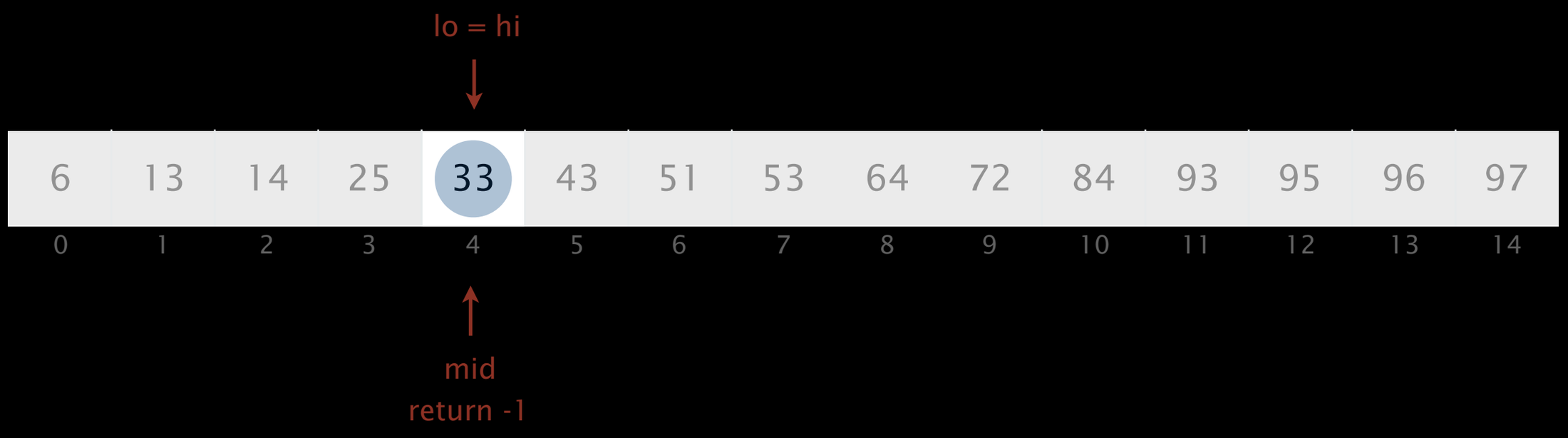

lo = hi
6
13
14
25
33
43
51
53
64
72
84
93
95
96
97
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
mid
return 4

UNSUCCESSFUL SEARCH FOR 34




6
13
14
25
33
43
51
53
64
72
84
93
95
96
97
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
lo
mid
hi


6
13
14
25
33
43
51
53
64
72
84
93
95
96
97
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
lo
mid
hi


lo = hi
6
13
14
25
33
43
51
53
64
72
84
93
95
96
97
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
mid
return -1

TRIVAL TO IMPLEMENT?

- First binary search published in 1946.
- First bug-free one in 1962.

  Bug in Java's `Arrays.binarySearch()` discovered in 2

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length-1;
   while (lo <= hi)
   {
       int mid = (lo + hi) / 2;
       if      (key < a[mid]) hi = mid - 1;
       else if (key > a[mid]) lo = mid + 1;
       else return mid;
   }
   return -1;
}
```

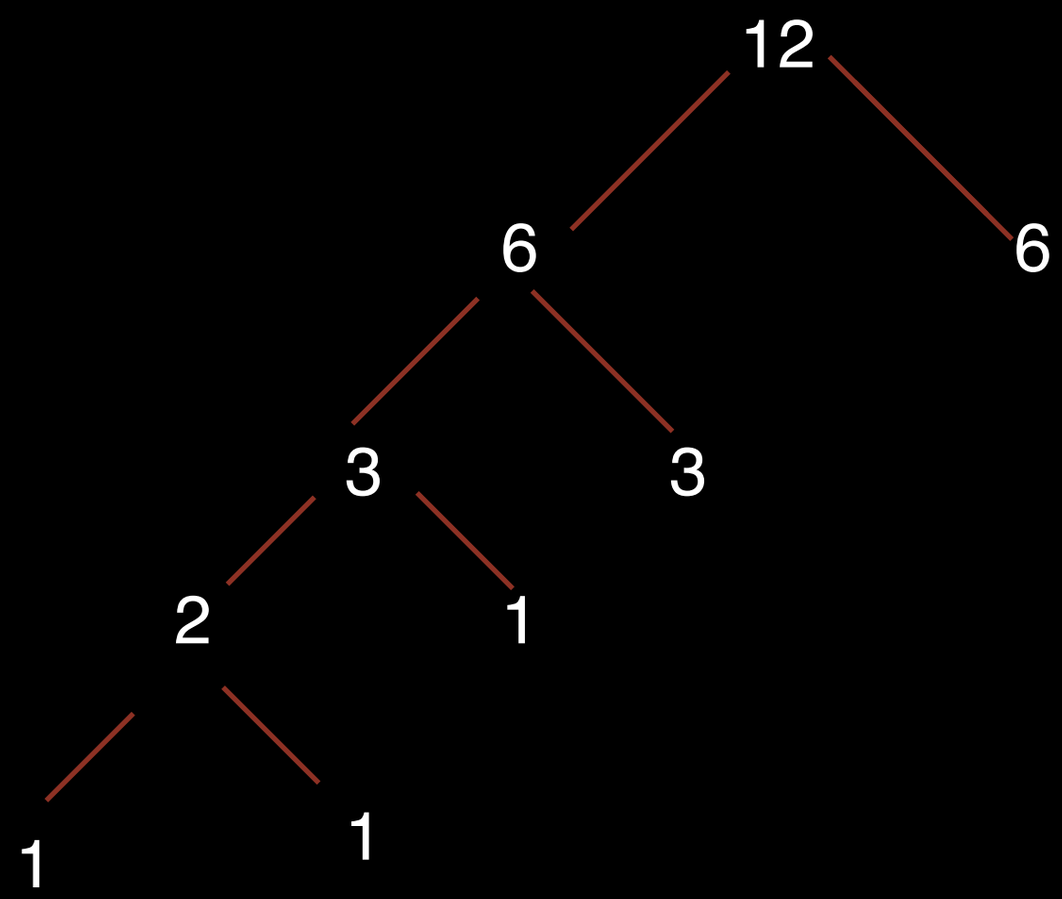
one "3-way compare"

CORRECT IMPLEMENTATION

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length-1;
   while (lo <= hi)
   {
       int mid = lo + (hi - lo) / 2;
       if      (key < a[mid]) hi = mid - 1;
       else if (key > a[mid]) lo = mid + 1;
       else return mid;
   }
   return -1;
}
```

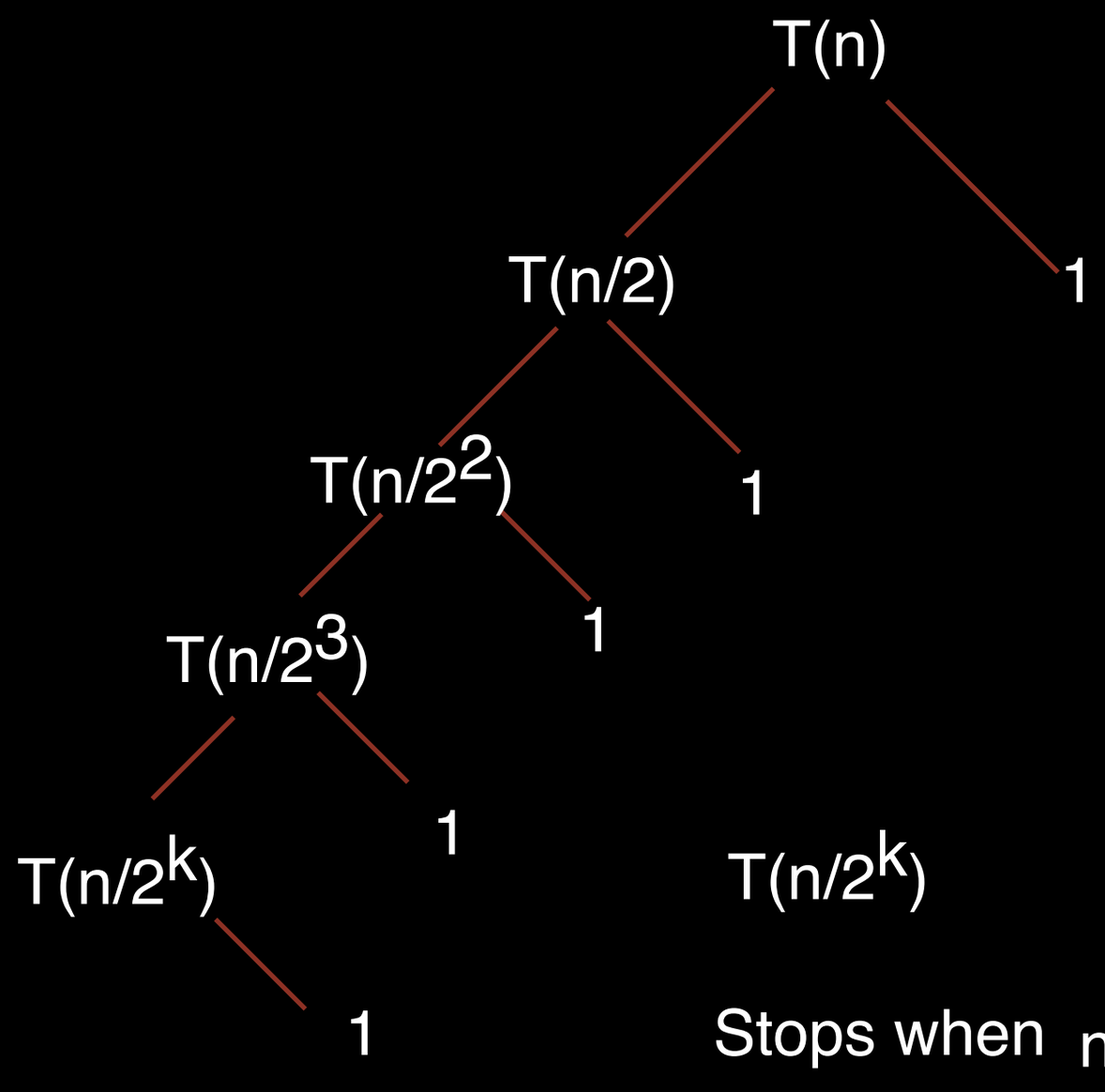
one "3-way compare"

HOW DO WE KNOW THAT BINARY SEARCH IS FASTER?



floor(log(12)/log(2)) + 1 = 4

RECURRENCE TREES



BASE CASE

$T(1) = 1.$

$T(n/2^k)$

Stops when $n/2^k=1$

$n=2^k$

$k = \log_2(n)$

TOTAL NUMBER OF COMPARES

$\log_2(n) + 1$

Proposition. Binary search uses at most $1 + \lg N$ key compares to search in a sorted array of size N.

Def. $T(N)$ = # key compares to binary search a sorted subarray of size $\leq N$.

Binary search recurrence. $T(N) \leq T(N/2) + 1$ for $N > 1$, with $T(1) = 1$.

left or right half

(floored division)

Pf sketch. [assume N is a power of 2]

$$
\begin{aligned}
T(N) &\leq T(N/2) + 1 && \text{[given]} \\
&\leq (T((N/2)/2) + 1) + 1 && \text{[apply recurrence to first term]} \\
&\leq T(N/8) + 1 + 1 + 1 && \text{[apply recurrence to first term]} \\
&\vdots \\
&\leq T(N/N) + 1 + 1 + \ldots + 1 && \text{[stop applying, T(1) = 1]} \\
&= 1 + \lg N
\end{aligned}
$$

An $N^2 \log N$ algorithm for 3-SUM

Algorithm.

- Step 1: Sort the N (distinct) numbers.
- Step 2: For each pair of numbers `a[i]` and `a[j]`, binary search for `-(a[i] + a[j])`.

Analysis.

Order of growth is $N^2 \log N$.

- Step 1: N^2 with insertion sort.
- Step 2: $N^2 \log N$ with binary search.

input

```
 30 -40 -20 -10  40   0  10   5
```

sort

```
-40 -20 -10   0   5  10  30  40
```

binary search

```
(-40, -20)    60
(-40, -10)    50
(-40,   0)    40
(-40,   5)    35
(-40,  10)    30
   ⋮           ⋮
(-20, -10)    30
   ⋮           ⋮
(-10,   0)    10
   ⋮           ⋮
( 10,  30)   -40
( 10,  40)   -50
( 30,  40)   -70
```

Does my program run faster than yours

Hypothesis. The sorting-based $N^2 \log N$ algorithm for 3-SUM is significantly faster in practice than the brute-force N^3 algorithm.

N	time (seconds)
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1

N	time (seconds)
1,000	0.14
2,000	0.18
4,000	0.34
8,000	0.96
16,000	3.67
32,000	14.88
64,000	59.16

Guiding principle. Typically, better order of growth $\Rightarrow$ faster in practice.

WE WANT TO BE ABLE TO REASON MORE GENERALLY ABOUT THE PROBLEM

Goals.

- Classify and compare different classes of algorithms

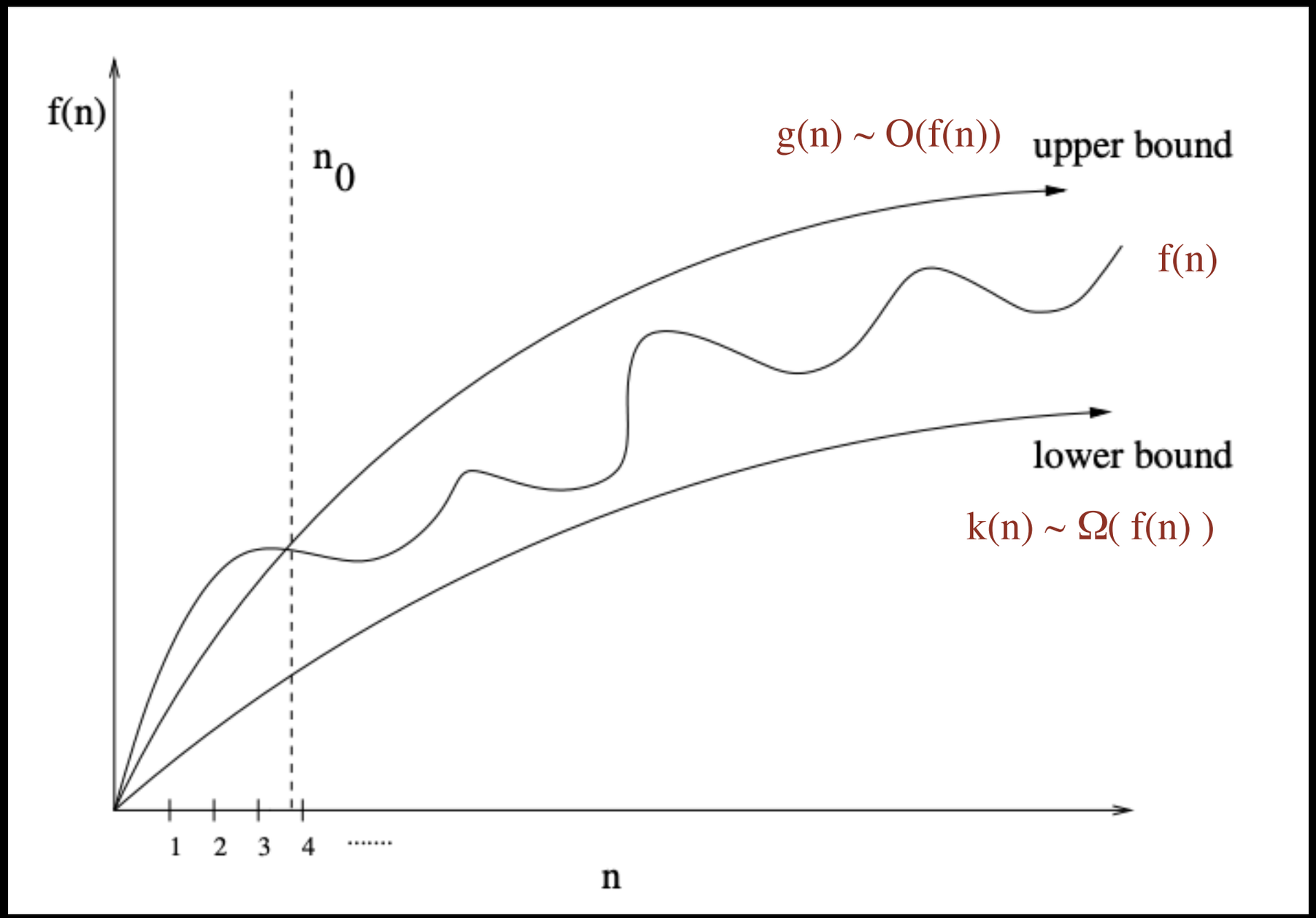
Approach.

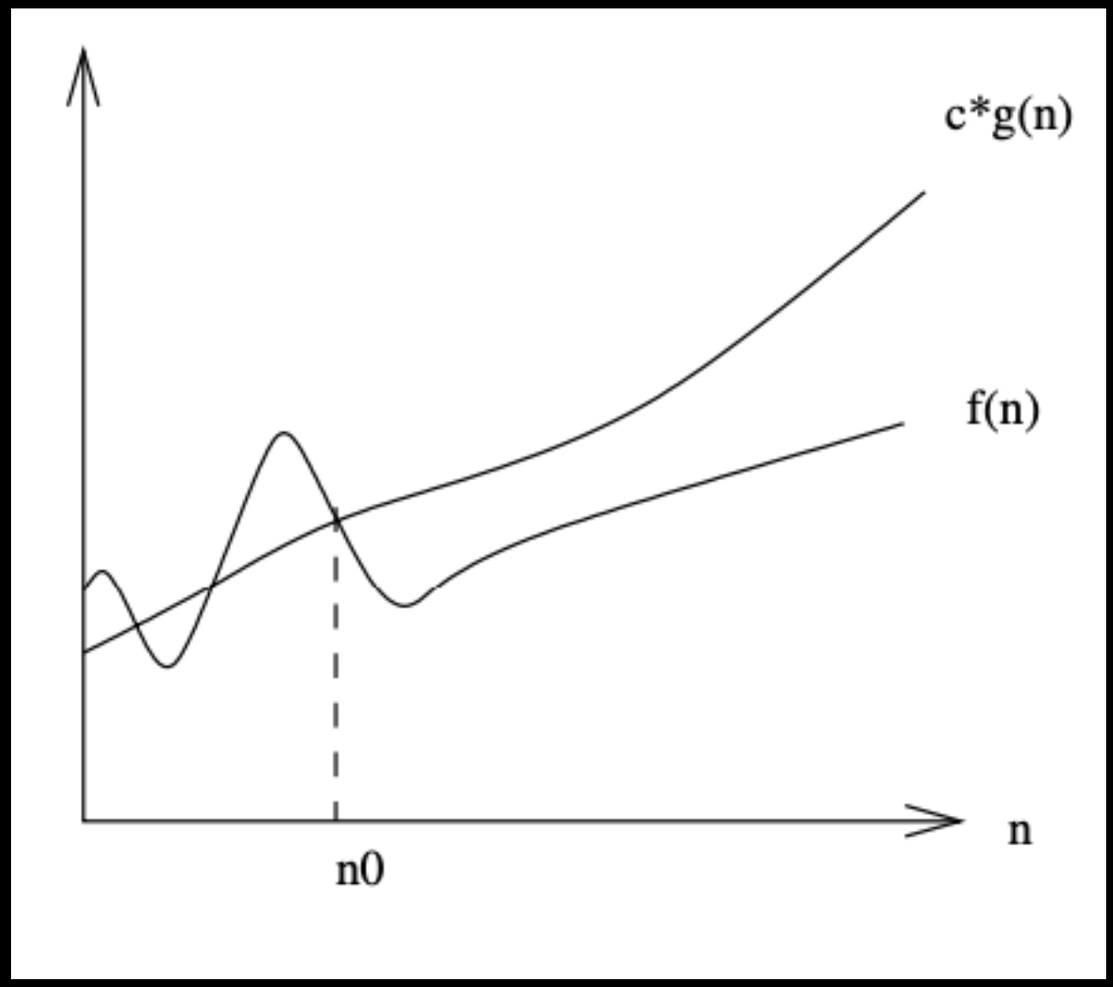
- Upper bound. Performance guarantee of algorithm for any input.
- Lower bound. The algorithm performance in the best case.
- Optimal algorithm. (“Tight Bound”) Lower bound = upper bound (to within a cons
  factor).

Commonly-used notations in the theory of algorithms

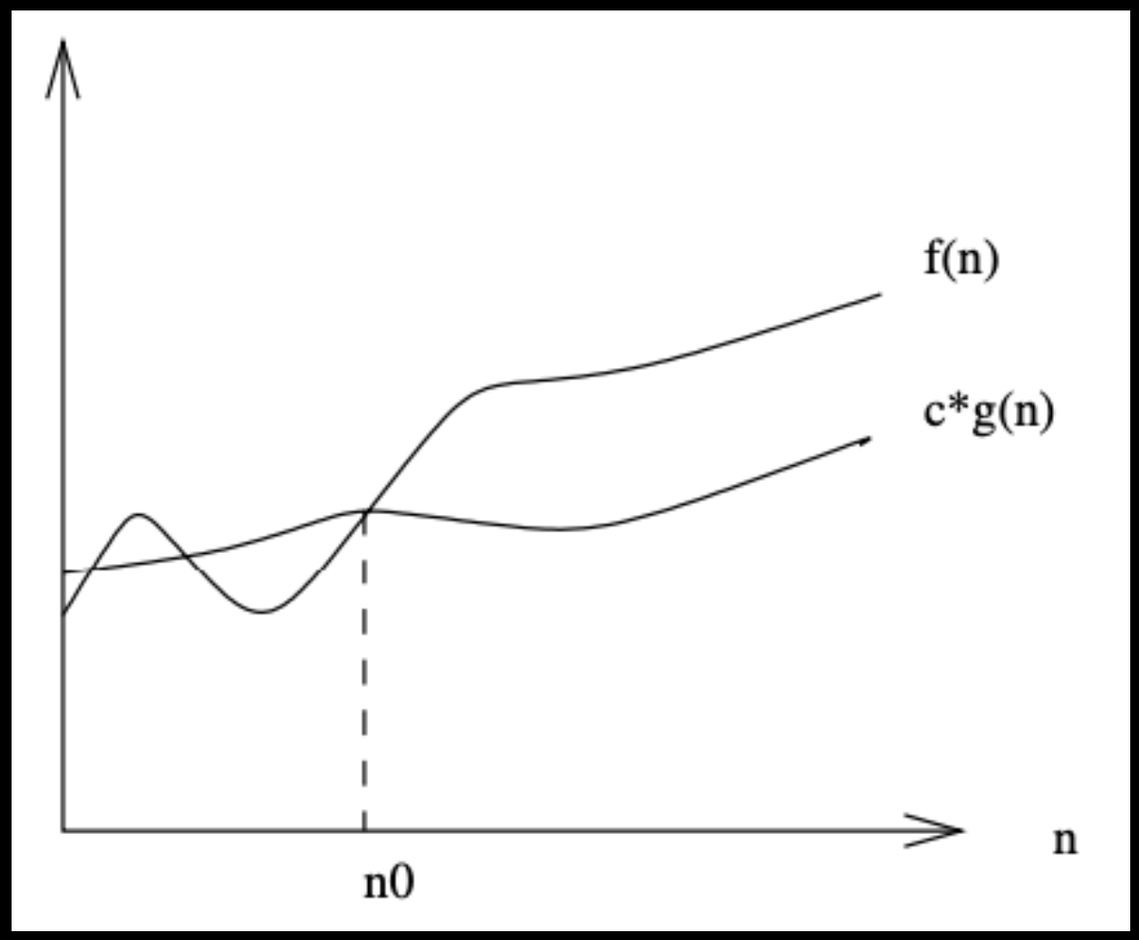
notation	example	shorthand for	used to
Big Oh	$O(N^2)$	$10\,N^2$ $100\,N$ $22\,N \log N + 3\,N$ ⋮	develop upper bounds
Big Omega	$\Omega(N^2)$	$\frac{1}{2}\,N^2$ N^5 $N^3 + 22\,N \log N + 3\,N$ ⋮	develop lower bounds

Commonly-used notations in the theory of algorithms

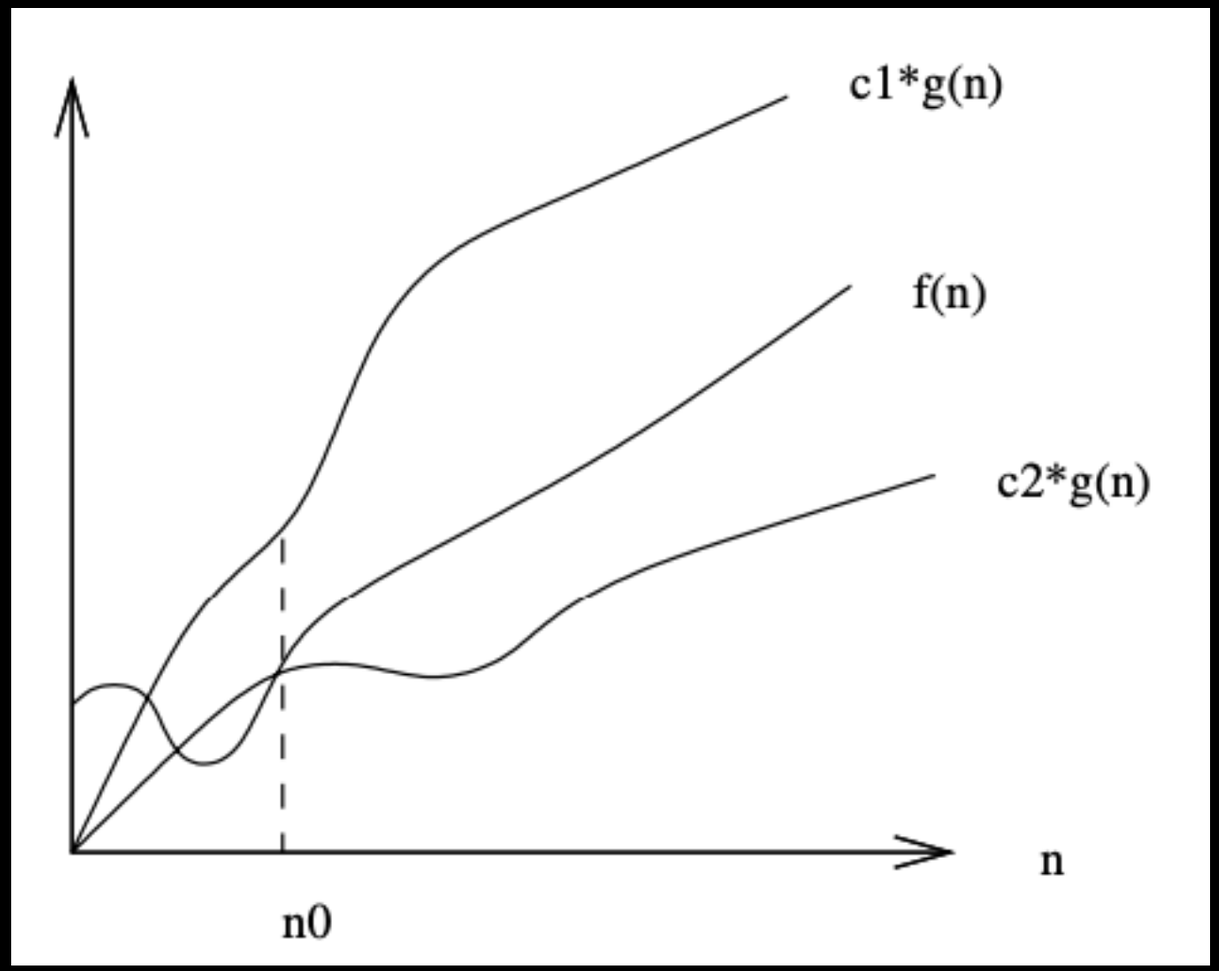




f(n) = O(g(n)) means c*g(n) is an upper bound on f(n). Thus there exists some constant c such that f(n) is always ≤ c * g(n), for large enough n (i.e, n ≥ n.0 for some constant n0)



f(n) = Ω (g(n)) means c * g(n) is an upper bound on f(n). Thus there exists some constant c such that f(n) is always ≥c*g(n), for all n ≥ n0



f(n) = Θ(g(n)) means c1*g(n) is an upper bound on f(n) and c2*g(n) is a lower bound on f(n), for all n ≥n0. Thus there exist constants c1 and c2 such that f(n) ≥ c1*g(n) and f(n) ≥c2*g(n). This means that g(n) provides nice, tight bound on f(n)

Stop and Think: Hip to the Squares?

Is $(x+y)^2 = O(x^2 + y^2)$?

$(x+y)^2 \le c(x^2 + y^2)$

$x^2 + 2xy + y^2 \le c(x^2 + y^2)$

If the 2xy wasn't there it would clearly hold

We need to relate $2xy$ and $x^2 + y^2$

$x^2 \le y^2$

$x \le y$

$x \ge y$

$y^2 \le x^2$

$2xy \le 2y^2 \le 2(x^2 + y^2)$

$2xy \le 2x^2 \le 2(x^2 + y^2)$

Either way, we now can bound this middle term by two times the right-side function.

$(x+y)^2 \le c(x^2 + y^2)$ for $c \ge 2$

ORDER OF GROWTH MEMORY

Typical memory usage for primitive types and arrays

type	bytes
`boolean`	1
`byte`	1
`char`	2
`int`	4
`float`	4
`long`	8
`double`	8

primitive types

type	bytes
`char[]`	$2N + 24$
`int[]`	$4N + 24$
`double[]`	$8N + 24$

one-dimensional arrays

type	bytes
`char[][]`	$\sim 2MN$
`int[][]`	$\sim 4MN$
`double[][]`	$\sim 8MN$

two-dimensional arrays

Typical memory usage for objects in Java

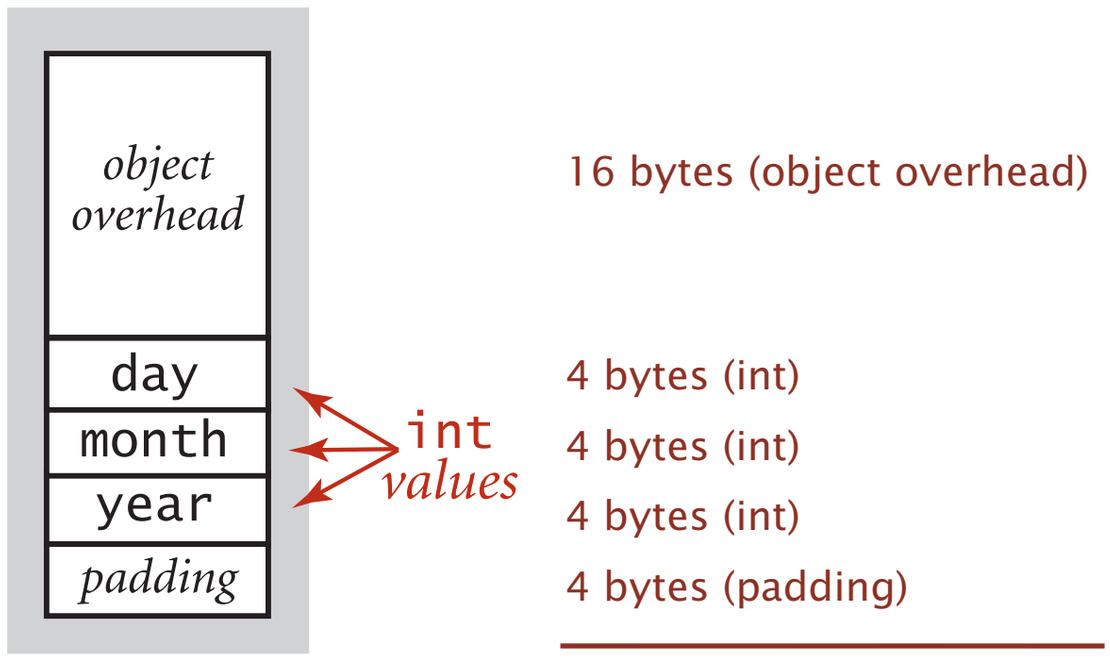
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Each object uses a multiple of 8 bytes.

Ex 1. A `Date` object uses 32 bytes of memory.

```
public class Date
{
   private int day;
   private int month;
   private int year;
...
}
```



Memory profiler

Classmexer library. Measure memory usage by querying JVM.
http://www.javamex.com/classmexer

```
import com.javamex.classmexer.MemoryUtil;

public class Memory {
   public static void main(String[] args) {
      Date date = new Date(1, 14, 2019);
      StdOut.println(MemoryUtil.memoryUsageOf(date));
      String s = "Hello, World”;
   }
}
```

```
% javac -cp .:classmexer.jar Memory.java
% java  -cp .:classmexer.jar -javaagent:classmexer.jar Memory
32
```

Reference:
Slides based on content from:
Kevin Wayne
Robert Shedgewich
Steven Skiena
Nathan Burnelle