INTRODUCTION TO DATA STRUCTURES and ALGORITHMS

Rutgers University

1A. Greatest Hits of 111

- **Memory, objects, Arrays**
- **Program stack and heap**
- **Images as 2D arrays**
- **Introduction to program analysis**
- **Running time (experimental analysis)**
- **Running time (mathematical models)**
- **Memory usage**
- **Garbage Collection in Java**

https://ds.cs.rutgers.edu

Last updated on 1/8/21

1A. Greatest Hits of 111

- *Memory, objects, Arrays*
- *Program stack and heap*
- *Images as 2D arrays*
- *Introduction to program analysis*
- *Running time (experimental analysis)*
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

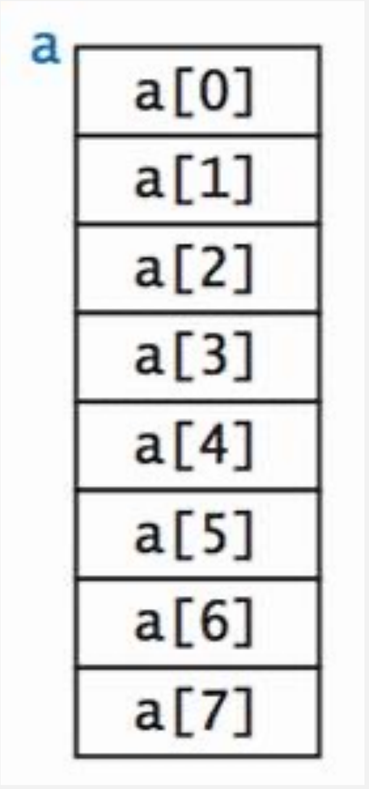
Some slides Adopted and modified from Sedgewick and Wayne

Memory, objects and arrays

LO 1.1

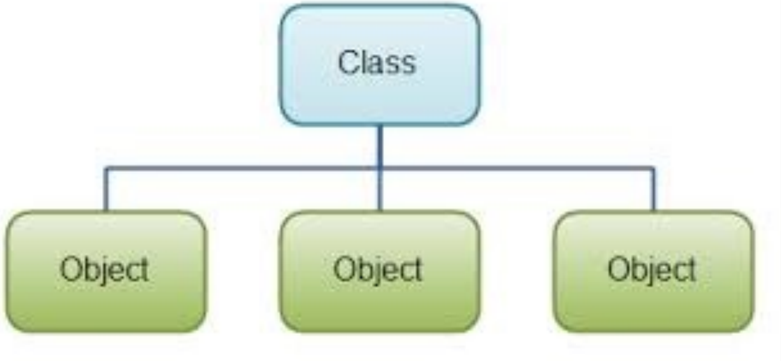
A view of physical memory storage

```
int[] a = new int[8];
```



An Array is a contiguous block of memory

```
String str1 = new String(“Welcome ”);
String str2 = new String(“to ”);
String str3 = new String(“class”);
```



Object is an instance of a class

Memory representation of an array

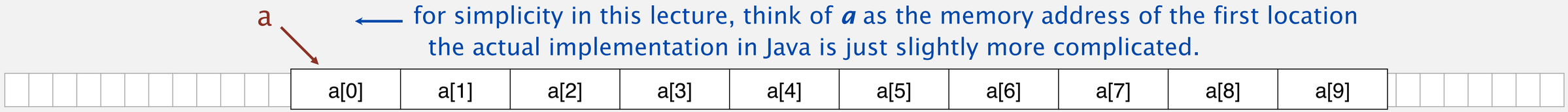
LO 1.1

An array is an indexed sequence of values of the same type.

A computer's memory is *also* an indexed sequence of memory locations.

- Each primitive type value occupies a fixed number of locations.
- *Array values are stored in contiguous locations.*

← stay tuned for many details



← for simplicity in this lecture, think of ***a*** as the memory address of the first location
the actual implementation in Java is just slightly more complicated.

Critical concepts

- Indices start at 0.
- Given i, the operation of accessing the value a[i] is extremely efficient.
- The assignment b = a makes the names b and a refer to the same array.

it does *not* copy the array,
as with primitive types
(stay tuned for details)

Java language support for arrays

Basic support

operation	*typical code*
Declare an array reference	double[] a;
Create an array of a given length	a = new double[1000];
Refer to an array entry by index	a[i] = b[j] + c[k];
Refer to the length of an array	a.length;

Initialization options

operation	*typical code*
Default initialization to 0 for numeric types	a = new double[1000];
Declare, create and initialize in one statement	double[] a = new double[1000];
Initialize to literal values	double[] x = { 0.3, 0.6, 0.1 };

no need to use a loop like

```
for (int i = 0; i < 1000; i++)
  a[i] = 0.0;
```

BUT cost of creating an array is proportional to its length.

Programming with arrays: typical examples

create an array with random values

```
double[] a = new double[n];
for (int i = 0; i < n; i++)
   a[i] = Math.random();
```

write to the array location with index i

print the array values, one per line

```
for (int i = 0; i < n; i++)
   System.out.println(a[i]);
```

read the element in index i

find the maximum of the array values

```
double max = Double.NEGATIVE_INFINITY;
for (int i = 0; i < n; i++)
   if (a[i] > max) max = a[i];
```

read from array index i and assign the value to max

compute the average of the array values

```
double sum = 0.0;
for (int i = 0; i < n; i++)
   sum += a[i];
double average = sum / n;
```

reverse the values within an array

```
for (int i = 0; i < n/2; i++)
{
   double temp = a[i];
   a[i] = a[n-1-i];
   a[n-i-1] = temp;
}
```

executed after each iteration

executed before each iteration

executed before the first iteration

copy sequence of values to another array

```
double[] b = new double[n];
for (int i = 0; i < n; i++)
   b[i] = a[i];
```

2D Arrays

LO 1.1, 1.2

```
double[][] a;
a = new double[m][n];
for (int i = 0; i < m; i++)
   for (int j = 0; j < n; j++)
      a[i][j] = 0;
```

2D Array Allocation

```
double[][] a = {
    { 99.0, 85.0, 98.0, 0.0 },
    { 98.0, 57.0, 79.0, 0.0 },
    { 92.0, 77.0, 74.0, 0.0 },
    { 94.0, 62.0, 81.0, 0.0 },
    { 99.0, 94.0, 92.0, 0.0 },
    { 80.0, 76.5, 67.0, 0.0 },
    { 76.0, 58.5, 90.5, 0.0 },
    { 92.0, 66.0, 91.0, 0.0 },
    { 97.0, 70.5, 66.5, 0.0 },
    { 89.0, 89.5, 81.0, 0.0 },
    {  0.0,  0.0,  0.0, 0.0 }
};
```

Initializing 2D array at compile time

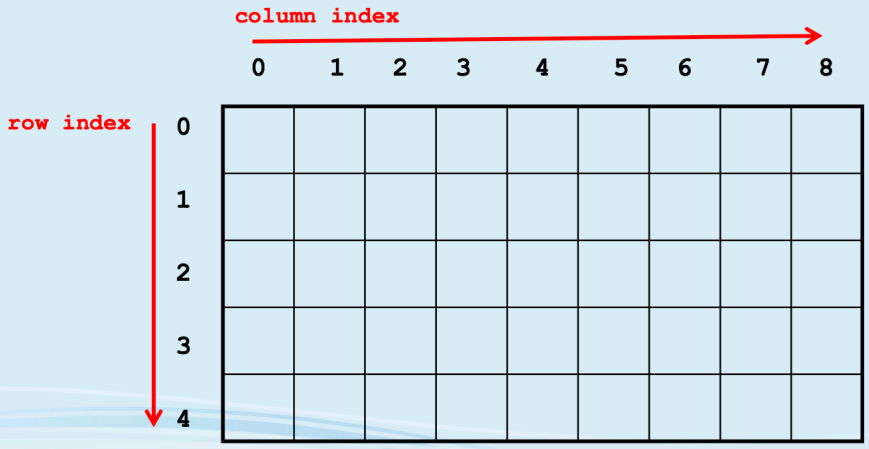
Memory representation. Java represents a two-dimensional array as an array of arrays. A matrix with `m` rows and `n` columns is actually an array of length `m`, each entry of which is an array of length `n`. In a two-dimensional Java array, we can use the code `a[i]` to refer to the ith row (which is a one-dimensional array). Enables ragged arrays.

`a[i].length = number of columns`

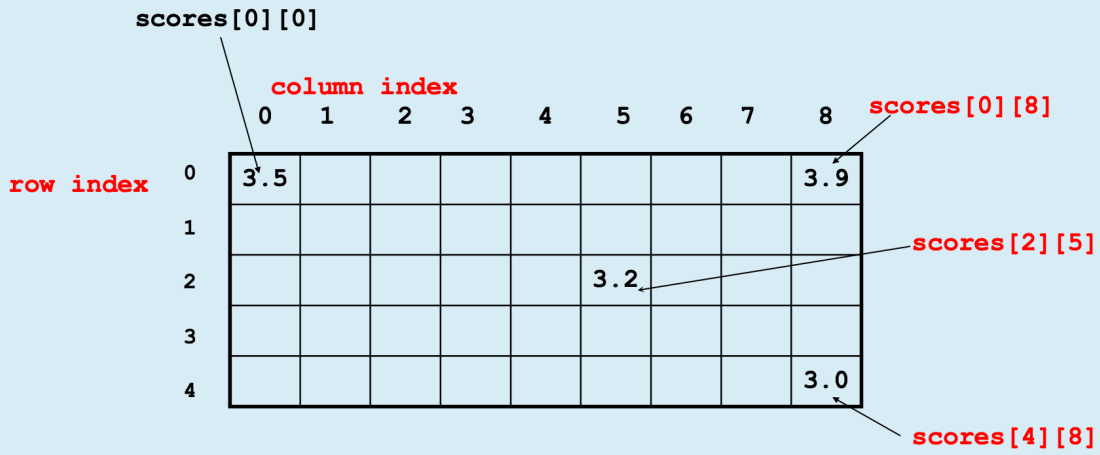
01A

- **2D Arrays**

```
double [][] scores = new double[5][9];
```




```
double [][] scores = new double[5][9]; // store 45 elements
```



1A. Greatest Hits of 111

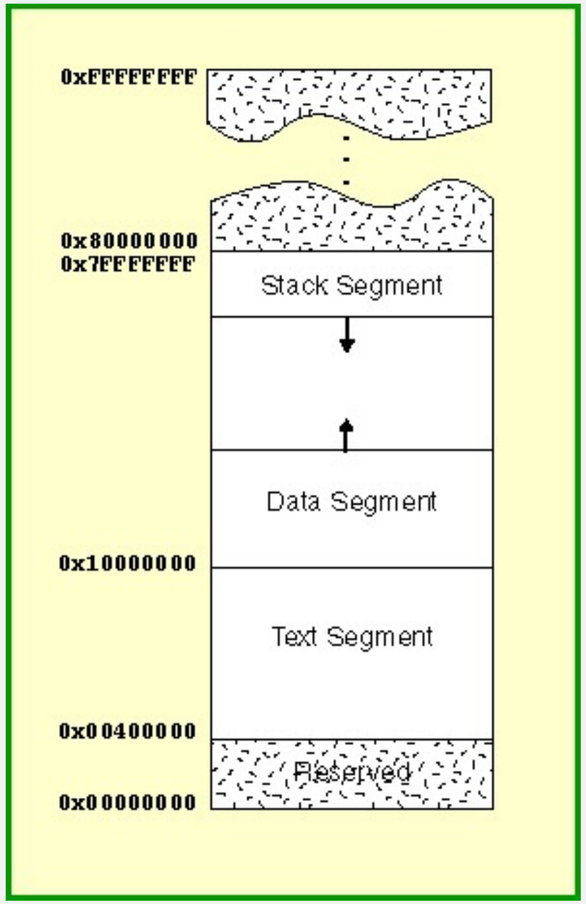
- *Memory, objects, Arrays*
- *Program stack and heap*
- *Images as 2D arrays*
- *Introduction to program analysis*
- *Running time (experimental analysis)*
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

Some slides Adopted and modified from Sedgewick and Wayne

Program stack and heap memory

LO 1.1

- Runtime Stack - a location in the memory where **static memory** is kept



-Each running thread has a private stack,it holds local variables and partial results
-Each entry in the runtime stack is known as activation record or stack frame
-At the bottom of the stack is the main(), which is the first method being called when a thread is running
-After completing all the methods, the stack will be empty and that runtime stack will be destroyed by the JVM before terminating the thread

- Runtime Heap – a location in the memory where **dynamic memory** is kept

Heap memory can be fragmented as they are allocated (new) and deallocated (garbage collection) as needed

-Shared among all JVM threads
-The run-time data area from which memory for all class instances (objects) and arrays is allocated
-Heap storage for objects is reclaimed by an automatic storage management system (known as a garbage collector)

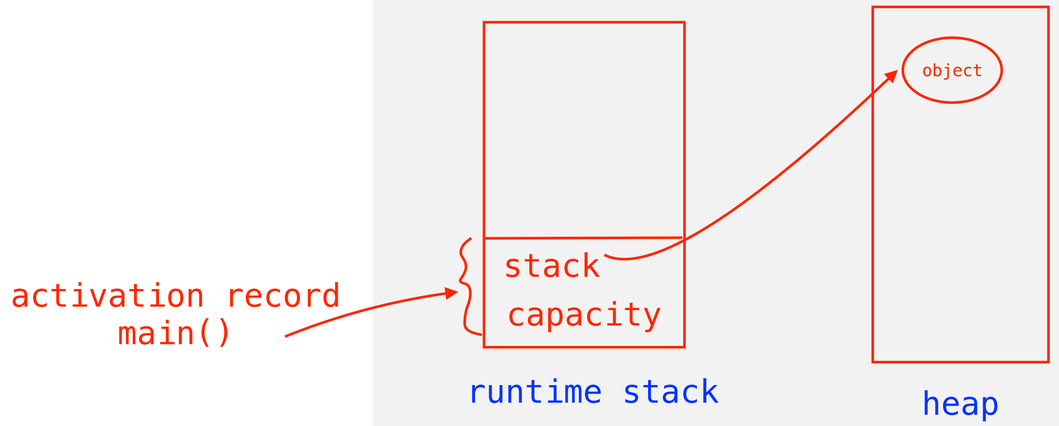
Source: https://chortle.ccsu.edu/

Quiz

LO 1.1

Identify the locations of this code where static memory (compile time) and dynamic memory (run time) are allocated?

```
public static void main(String[] args) {
    int capacity = Integer.parseInt(args[0]);
    ArrayStackOfStrings stack = new ArrayStackOfStrings(capacity);
    while (!StdIn.isEmpty()) {
        String item = StdIn.readString();
        if (!item.equals("-")) {
            stack.push(item);
        }
        else {
            StdOut.print(stack.pop() + " ");
        }
    }
    StdOut.println();
}
```



Quiz

LO 1.1

Identify the locations of this code where static memory (compile time) and dynamic memory (run time) are allocated?

```
public static void main(String[] args) {
    int capacity = Integer.parseInt(args[0]);
    ArrayStackOfStrings stack = new ArrayStackOfStrings(capacity);
    while (!StdIn.isEmpty()) {
        String item = StdIn.readString();
        if (!item.equals("-")) {
            stack.push(item);
        }
        else {
            StdOut.print(stack.pop() + " ");
        }
    }
    StdOut.println();
}
```

Answer:

Stack (static memory, size known at compile time): `int capacity`
`ArrayStackOfStrings stack`

Heap (dynamic memory, size known at run time) : `ArrayStackOfStrings(capacity)`

an object created with the "new" keyword

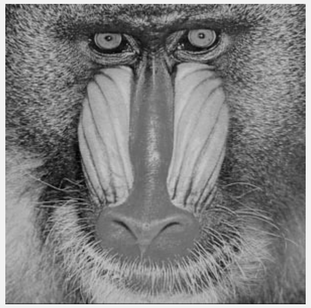
1. Greatest Hits of 111

- *Memory, objects, Arrays*
- *Program stack and heap*
- *Images as 2D arrays*
- *Introduction to analysis*
- *Running time (experimental analysis)*
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

Some slides Adopted and modified from Sedgewick and Wayne

Images as 2D Arrays

Color image



grayscale image

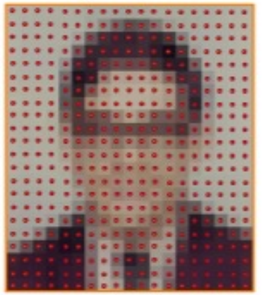
Red

Green

Blue

- An image is a 2D rectilinear array of pixels

Continuous image



Digital image

A pixel is a sample, not a little square!

(255, 255, 0)	(255, 255, 0	(255, 255, 0	(255, 255, 0			

Each entry has (R,G,B) color values
8-bit Integer between 0-255

Processing Images

Constructor and Description
`Picture(File file)` Creates a picture by reading the image from a PNG, GIF, or JPEG file.
`Picture(int width, int height)` Creates a `width`-by-`height` picture, with `width` columns and `height` rows, where each pixel is black.
`Picture(Picture picture)` Creates a new picture that is a deep copy of the argument picture.
`Picture(String name)` Creates a picture by reading an image from a file or URL.

```
int r = (rgb >> 16) & 0xFF;
int g = (rgb >>  8) & 0xFF;
int b = (rgb >>  0) & 0xFF;
```

Extracting RGB values from a 32-bit color pixel

r g b

00000000 01010101 00001111 10101010 >> 16 → 00000000 00000000 00000000 01010101

00000000 00000000 11111111 11111111

F F

```
int rgb = (r << 16) + (g << 8) + (b << 0);
```

Given RGB values (0-255) pack them into a 32-bit pixel

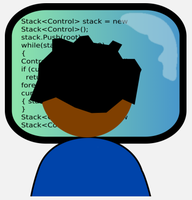
See: https://algs4.cs.princeton.edu/code/javadoc/edu/princeton/cs/algs4/Picture.html

1A. Greatest Hits of 111

- *Stack and heap*
- *Memory, objects, Arrays*
- *Images as 2D arrays*
- *Introduction to program analysis*
- *Running time (experimental analysis)*
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

Some slides Adopted and modified from Sedgewick and Wayne

Cast of characters



programmer needs to
develop a working solution

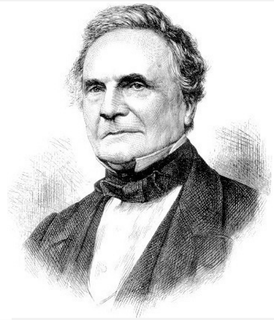
client wants to solve
problem efficiently

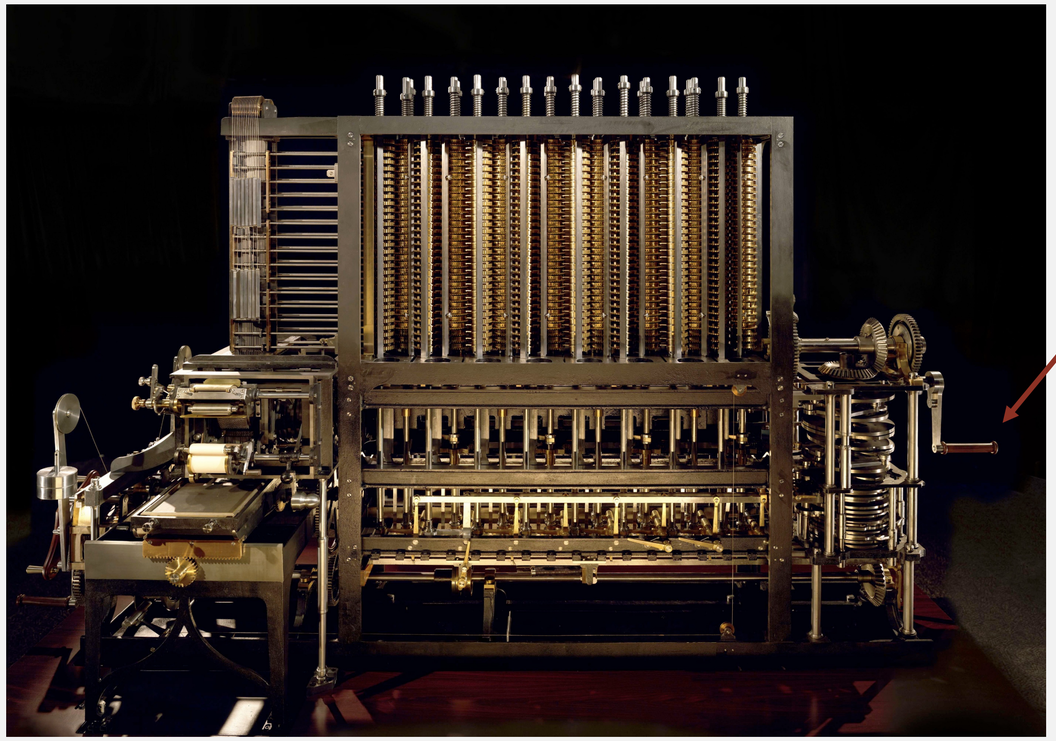
theoretician seeks
to understand

student (you)
might play all of
these roles someday

Running time

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

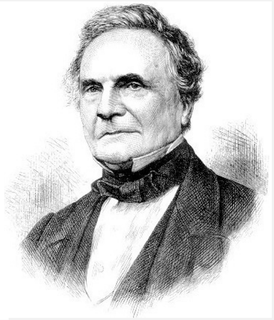




how many times do you have to turn the crank?

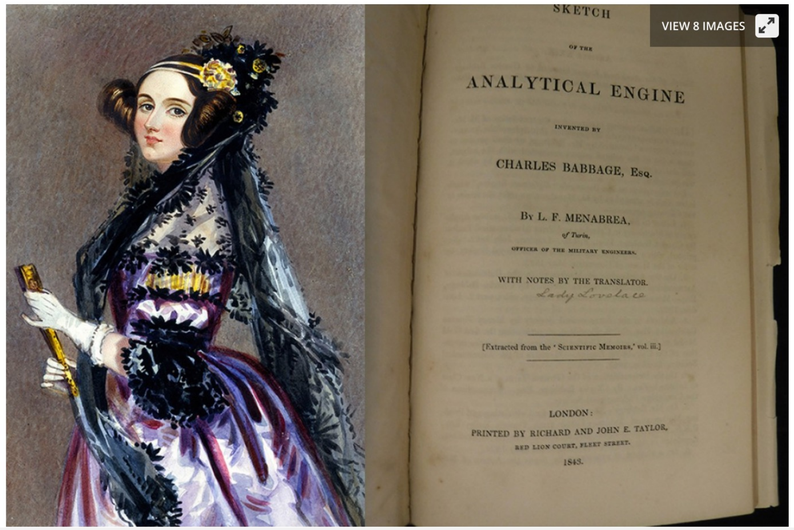
Running time

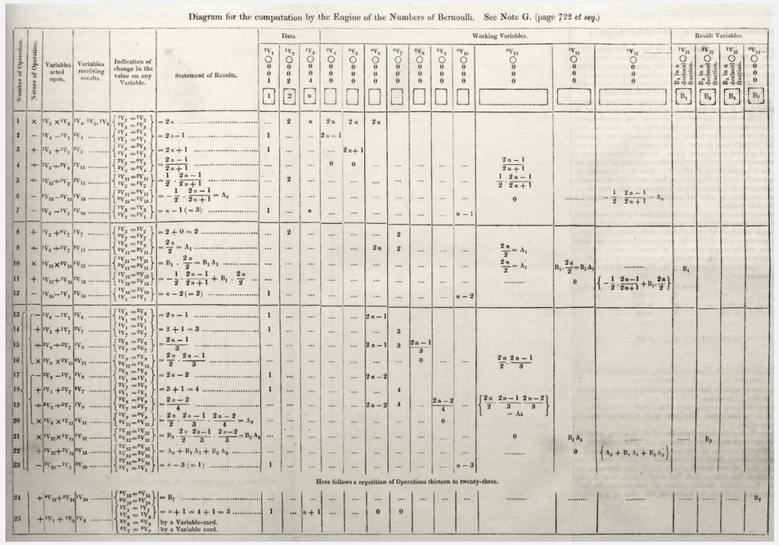
> *“ As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise—By what course of calculation can these results be arrived at by the machine in the shortest time? ”* — *Charles Babbage (1864)*



Rare book containing the world’s first computer algorithm earns $125,000 at auction

By Matt Kennedy
July 25, 2018



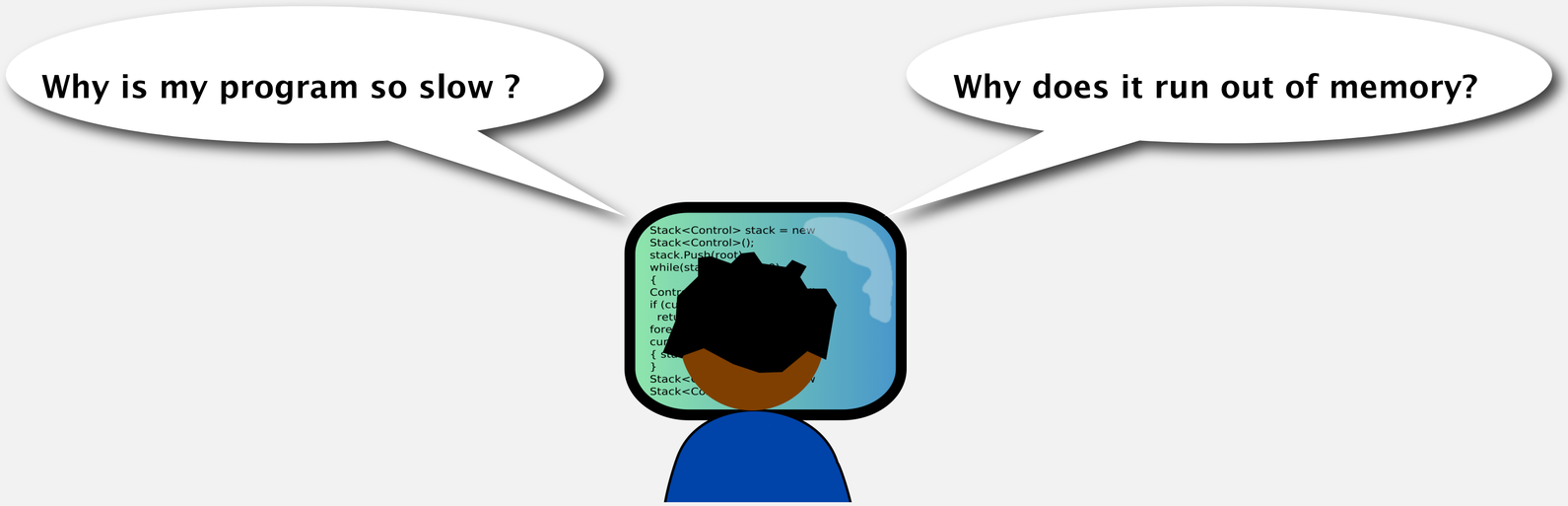


Ada Lovelace’s algorithm to compute Bernoulli numbers on Analytic Engine (1843)

Ada Lovelace’s program to compute Bernoulli numbers is the first published algorithm written for a computer

The challenge

Q. Will my program be able to solve a large practical input?



Our approach. Combination of experiments and mathematical modeling.

1A. Greatest Hits of 111

- *Stack and heap*
- *Memory, objects, Arrays*
- *Images as 2D arrays*
- *Introduction to program analysis*
- ***Running time (experimental analysis)***
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

Some slides Adopted and modified from Sedgewick and Wayne

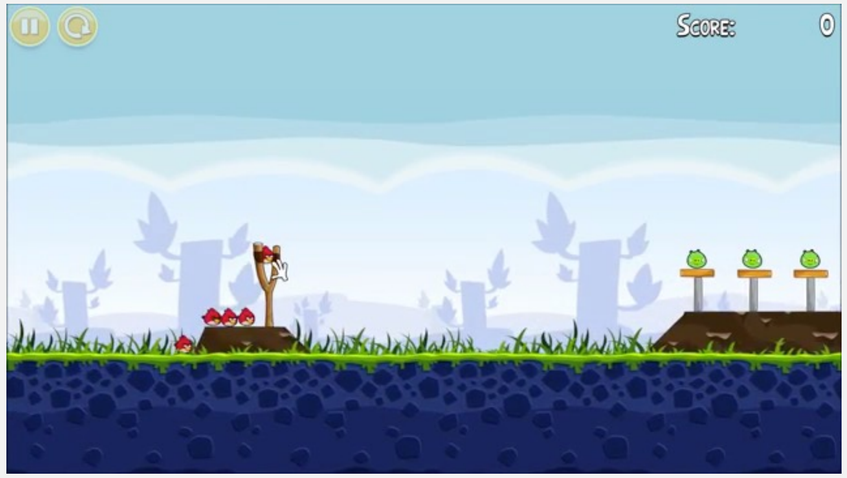
Example: 3-SUM

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

```
~/Desktop/3sum> more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

~/Desktop/3sum> 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	✓

Context. Related to problems in computational geometry.



3-SUM: brute-force algorithm

```
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 distinct triples

for simplicity,

ignore integer overflow

Measuring the running time

Q. How to time a program?

A. Manual.

`% java ThreeSum 1Kints.txt`

tick tick tick

`70`

`% java ThreeSum 2Kints.txt`

tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick

`528`

`% java ThreeSum 4Kints.txt`

tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick

`4039`

Measuring the running time

Q. How to time a program?

A. Automatic.

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.Stopwatch;

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);
}
```

Empirical analysis

Run the program for various input sizes and measure running time.

```
%
```

Empirical analysis

Run the program for various input sizes and measure running time.

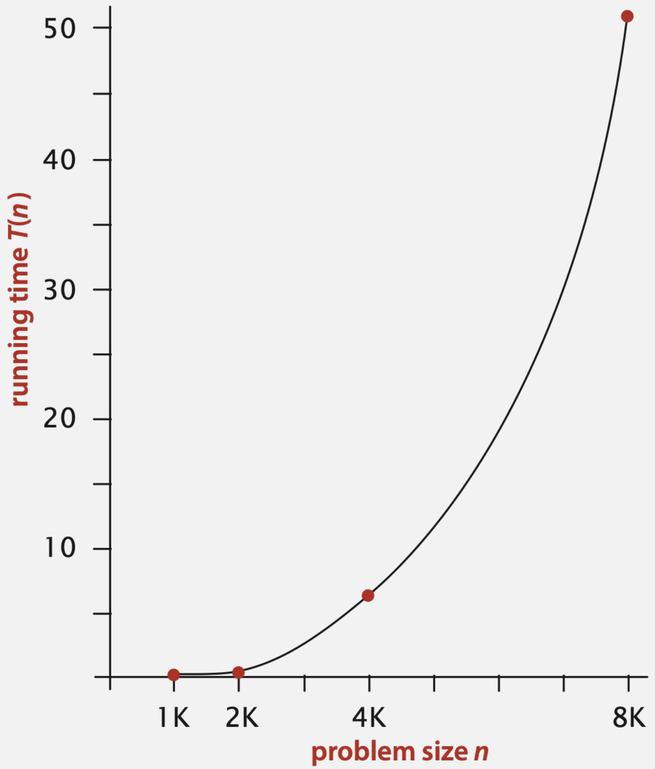
n	time (seconds) †
250	0
500	0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

How long will my program take, as a function of the input size?
To help answer this question let's plot the data.

† on a 2.8GHz Intel PU-226 with 64GB DDR E3 memory and 32MB L3 cache; running Oracle Java 1.7.0_45-b18 on Springdale Linux v. 6.5

Data analysis

Standard plot. Plot running time $T(n)$ vs. input size n.

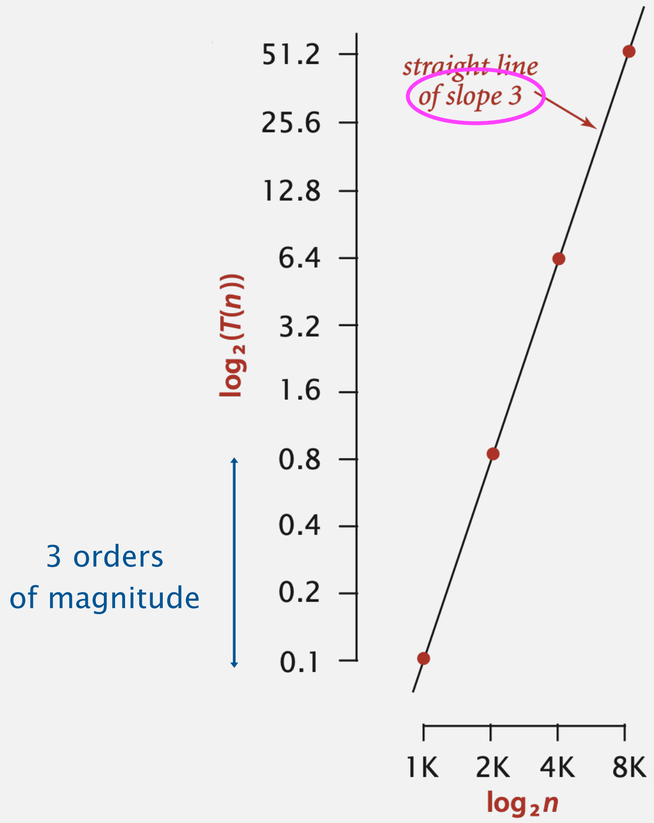


Hypothesis (power law). $T(n) = a\, n^b$

Questions. How to validate hypothesis? How to estimate a and b ?

Data analysis

Log–log plot. Plot running time $T(n)$ vs. input size n using log–log scale.



-uses logarithmic scales on both the horizontal and vertical axes.
-relationships of the form y = an^b appears as straight lines in a log–log graph; the power term corresponding to the slope, and the constant term corresponding to the intercept of the line.

slope

$$\log_2(T(n)) = 2.999 \log_2 n + (-33.21)$$

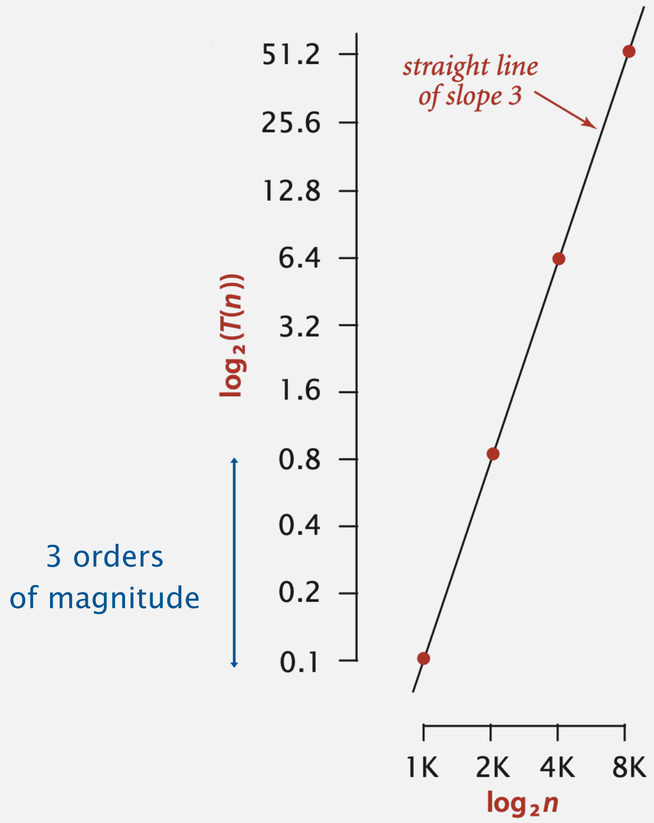
$$T(n) = 2^{-33.21} \times n^{2.999}$$
$$= 1.006 \times 10^{-10} \times n^{2.999}$$

Regression. Fit straight line through data points.

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

Data analysis (additional slide)

Log-log plot. Plot running time $T(n)$ vs. input size n using log-log scale.



Taking the logarithm of the equation $T(n) = a\ n^b$ yields:

slope

$$log_2\ (T(n)) = b\ log_2\ n + log_2\ a$$

which is the equation of the straight line of slope 3 on the log-log plot

$log_2\ (T(n)) = 3\ log_2\ n + log_2\ a$ (where a is a constant) is equivalent to:

$$T(n) = a\ n^3$$

The running time, as a function of the input size, as desired.

We can use one of our data points to solve for *a*. (slide 36)

for example, T(8000) = 51.1, thus a = 9.98 X 10^-11

we can use T(n) = 9.98 X 10^-11 X n^3 to estimate the running time for large n

Regression. Fit straight line through data points.

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

Prediction and validation, using regression

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	51.0
8,000	51.1
16,000	410.8

validates hypothesis!

Very accurate predictions.
2x input size -> 8x increase in running time for cubic algorithm

Doubling hypothesis

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

Run program, doubling the size of the input.

n	time (seconds) †	ratio	$\log_2$ 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	3.0
8,000	51.1	8	3.0

$$\frac{T(n)}{T(n/2)} = \frac{an^b}{a(n/2)^b} = 2^b$$

$$\Longrightarrow \quad b = \log_2 \frac{T(n)}{T(n/2)}$$

$\log_2 (6.4 / 0.8) = 3.0$

seems to converge to a constant $b \approx 3$

Hypothesis. Running time is about $a\, n^b$ with $b = \log_2$ ratio.

Caveat. Cannot identify logarithmic factors with doubling hypothesis.

Doubling hypothesis

Doubling hypothesis. Quick way to estimate exponent 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	51.0
8,000	51.1

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

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

Hypothesis. Running time is about $T(n) = 0.998 \times 10^{-10} \times n^3$ seconds.

almost identical hypothesis
to one obtained via regression
(but less work)

Analysis of algorithms: quiz 1

Estimate the running time to solve a problem of size $n = 96{,}000$.

A. 39 *seconds*

B. 52 *seconds*

C. 117 *seconds*

D. 350 *seconds*

n	time (seconds)
1,000	0.02
2,000	0.05
4,000	0.20
8,000	0.81
16,000	3.25
32,000	13.01

```
lg(0.81/02.) = 2.017921908
lg(3.25/0.81 = 2.004445905
lg(13.01/3.25) = 2.001109339
```

```
Order of growth is n^2.
n = 96,000, which is 3x the size of 32,000, we need 9x the time.
13 * 9 = 117.
```

Analysis of algorithms: quiz 1

Estimate the running time to solve a problem of size $n = 96{,}000$.

A. 39 *seconds*

B. 52 *seconds*

C. 117 *seconds*

D. 350 *seconds*

n	time (seconds)	ratio	$\log_2$ (ratio)
1,000	0.02	–	–
2,000	0.05	2.5	1.3
4,000	0.20	4.0	2.0
8,000	0.81	4.1	2.0
16,000	3.25	4.0	2.0
32,000	13.01	4.0	2.0

$T(n) = a\,n^2$ seconds

$T(32{,}000) = 13$ seconds

$\Rightarrow \quad a = 1.2695 * 10^{-8}$

$\Rightarrow \quad T(96{,}000) = 117$ seconds

$T(3n) = a\,(3n)^2$

$= 9\,a\,n^2$

$= 9\,T(n)$ seconds

$\Rightarrow \quad T(3 * 32{,}000) = 9 * 13 = 117$ seconds

Experimental algorithmics

System independent effects.

- Algorithm.
- Input data.

} determines exponent b in power law $a\,n^b$

System dependent effects.

- Hardware: CPU, memory, cache, …
- Software: compiler, interpreter, garbage collector, …
- System: operating system, network, other apps, …

} determines constant a in power law $a\,n^b$

Bad news. Sometimes difficult to get accurate measurements.

Doesn't say much about the algorithms

Context: the scientific method

Experimental algorithmics is an example of the scientific method.

Chemistry
(1 experiment)

Biology
(1 experiment)

Scientific method.
–Observe some feature of the natural world.
–Hypothesize a model that is consistent with the observations.
–Predict events using the hypothesis.
–Verify the predictions by making further observations.
–Validate by repeating until the hypothesis and observations agree.

Computer Science
(1 million experiments)

Physics
(1 experiment)

Good news. Experiments are easier and cheaper than other sciences.

1A. Greatest Hits of 111

- *Memory, objects, Arrays*
- *Program stack and heap*
- *Images as 2D arrays*
- *Introduction to program analysis*
- *Running time (experimental analysis)*
- ***Running time (mathematical models)***
- *Memory usage*
- *Garbage collection in Java*

Some slides Adopted and modified from Sedgewick and Wayne

Mathematical models for running time

LO 1.3

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

- Need to analyze program to determine set of operations.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.

The New York Times

Donald Knuth, master of algorithms, reflects on 50 years of his opus-in-progress, "The Art of Computer Programming."

Warning. No general-purpose method (e.g., halting problem).

Example: 1-SUM

Q. How many operations as a function of input size n ?

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

exactly n array accesses

operation	cost (ns) †	frequency
variable declaration	2/5	2
assignment statement	1/5	2
less than compare	1/5	$n + 1$
equal to compare	1/10	n
array access `a[i]`	1/10	n
increment `i` `count`	1/10	n to $2\,n$

in practice, depends on caching, bounds checking, … (see COS 217)

```
if all elements are not zero, count will not be incremented -> n
if all elements are zero, count will be incremented -> 2n
```

† representative estimates (with some poetic license)

Analysis of algorithms: quiz 2

How many array accesses as a function of 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++;
```

2 array accesses in the if statement
what is the frequency of executing the if statement?

A. $\frac{1}{2}\, n\,(n-1)$

B. $n\,(n-1)$

C. $2\, n^2$

D. $2\, n\,(n-1)$

Analysis of algorithms: quiz 2

How many array accesses as a function of 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++;
```

2 array accesses each iteration

1st iteration the outer loop with counter i, the number of times if statement is executed

$$(n-1) + (n-2) + \ldots + 1 + 0$$

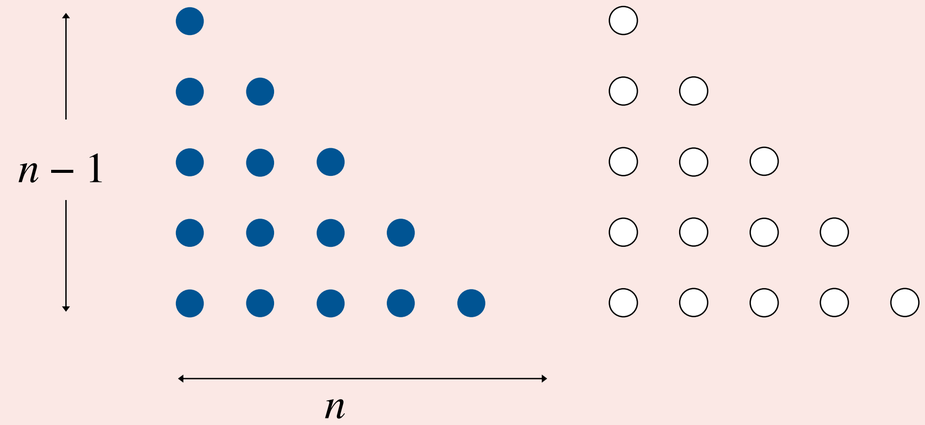
$$\tfrac{1}{2}\, n\, (n-1)$$ → total number of times if statement is executed

A. $\tfrac{1}{2}\, n\, (n-1)$

B. $n\, (n-1)$

C. $2\, n^2$

D. $2\, n\, (n-1)$



2 array accesses

$$2 * (1 + 2 + 3 + \ldots + n - 1) = n\,(n-1)$$

$$\Longrightarrow (1 + 2 + 3 + \ldots + n - 1) = \tfrac{1}{2}\, n\,(n-1)$$

Example: 2-SUM

Q. How many operations 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++;
```

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

$$= \binom{n}{2}$$ = n!/2!(n-2)!

operation	cost (ns)	frequency
variable declaration	2/5	$n + 2$
assignment statement	1/5	$n + 2$
less than compare	1/5	$\frac{1}{2} (n + 1) (n + 2)$
equal to compare	1/10	$\frac{1}{2} n (n - 1)$
array access	1/10	$n (n - 1)$
increment	1/10	$\frac{1}{2} n (n + 1)$ to n^2

approximation –
binomial coefficients of the x^k term
in the polynomial expansion of the
binomial power (1 + x)^n

$1/4\ n^2 + 13/20\ n + 13/10$ ns

to

$3/10\ n^2 + 3/5\ n + 13/10$ ns

(tedious to count exactly)

Simplification 1: cost model

Cost model. Use some elementary operation as a proxy for running time.

- Use the operation that cost and frequency are the highest

Frequency analyses are challenging and can lead to complicated and lengthy mathematical expressions

```
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	cost (ns)	frequency
variable declaration	2/5	$n + 2$
assignment statement	1/5	$n + 2$
less than compare	1/5	$\frac{1}{2}(n + 1)(n + 2)$
equal to compare	1/10	$\frac{1}{2} n (n - 1)$
array access	1/10	$n (n - 1)$
increment	1/10	$\frac{1}{2} n (n + 1)$ to n^2

cost model = array accesses

(we're assuming compiler/JVM does not optimize any array accesses away!)

Simplification 2: asymptotic notations

LO 1.4

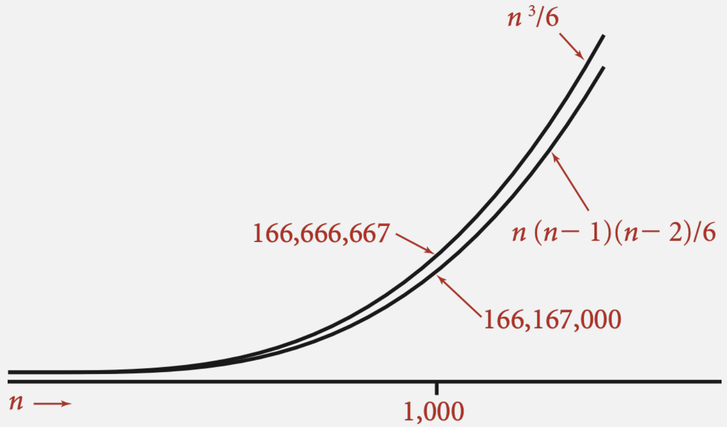
Tilde notation. Discard lower-order terms.

Big O notation. Also discard leading coefficient.

formal definitions involve limits

function	tilde	big O
$4n^5 + 20n + 16$	$\sim 4n^5$	$O(n^5)$
$7n^2 + 100n^{4/3} + 56$	$\sim 7n^2$	$O(n^2)$
$\frac{1}{6}n^3 - \frac{1}{2}n^2 + \frac{1}{3}n$	$\sim \frac{1}{6}n^3$	$O(n^3)$

discard lower-order terms
(e.g., n = 1,000: 166.67 million vs. 166.17 million)



Leading-term approximation

Rationale.

- When n is large, lower-order terms are negligible.
- When n is small, we don't care.

Simplification 2: tilde and big Theta notations

Tilde notation. Discard lower-order terms.

Big Theta notation. Also discard leading coefficient.

Use tilde notation to:

- Count core operations (e.g., compares or array accesses).
- Make predictions about running time (in seconds).
- Measure memory usage (in bytes).

Use big Theta notation to:

- Analyze performance independent of machine, compiler, JVM, ...
- Understand why an algorithm doesn't scale.

```
provides the information that you need to understand
limitations on the size of the problems a program can
solve, but not useful for predicting performance
```

```
big-Oh (upper bound, worst case),
big-Omega (lower bound, cannot be worse than this),
big-Theta (optimal, cannot do better than this)
```

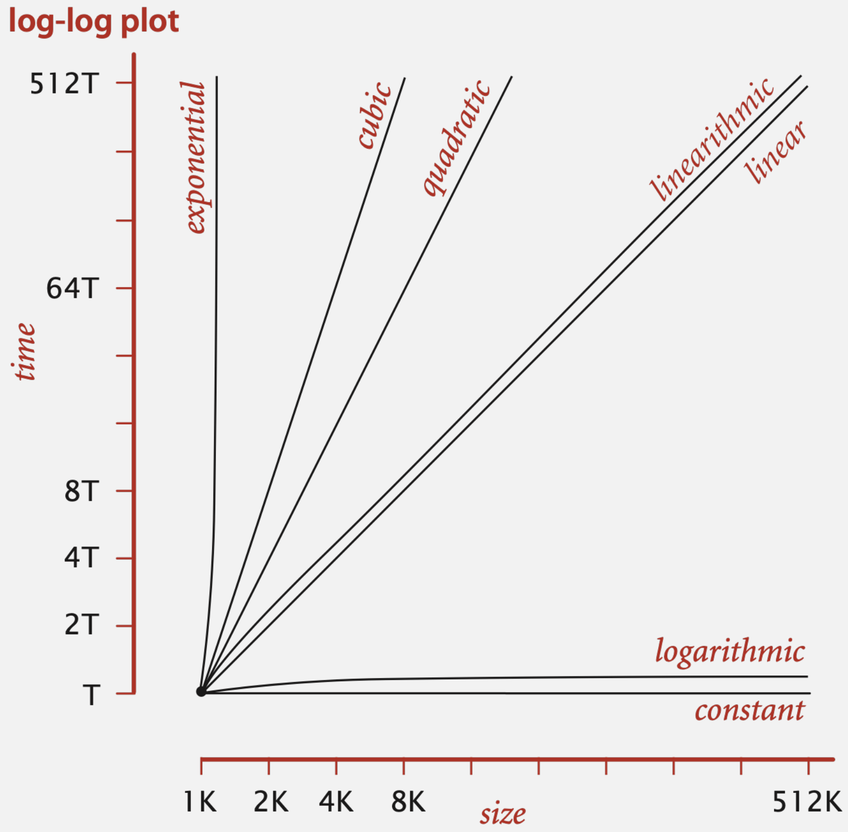
Common order-of-growth classifications

LO 1.5

Good news. The set of functions

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

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



Typical orders of growth

Common order-of-growth classifications

LO 1.4

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

Example: 2-SUM

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++)
      if (a[i] + a[j] == 0)
         count++;
```

"inner loop"

$$0 + 1 + 2 + \ldots + (n-1) = \frac{1}{2} n(n-1)$$
$$= \binom{n}{2}$$

A. $\sim n^2$ array accesses.

Example: 3-SUM

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.

3 array accesses in the if statement

3 x 1/6 x n^3 = 1/2 x n^3

n!/3!(n-3)!

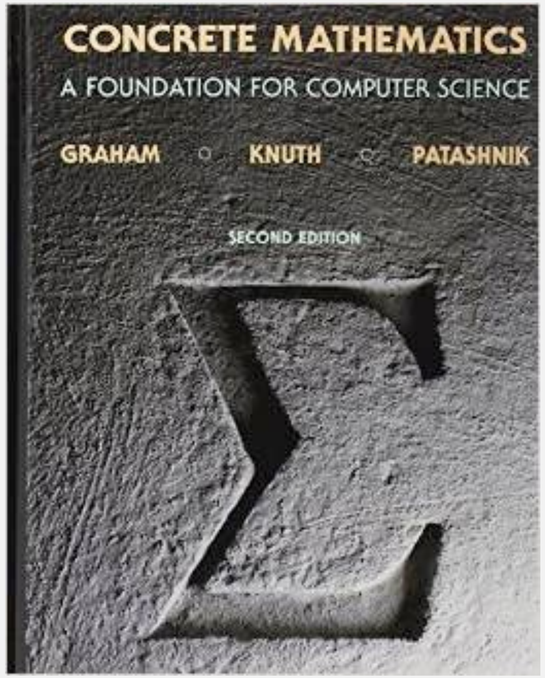
$$\binom{n}{3} = \frac{n(n-1)(n-2)}{3!} \sim \frac{1}{6} n^3$$

Bottom line. Use cost model and asymptotic notation to simplify analysis.

Estimating a discrete sum

Q. How to estimate a discrete sum?

A1. Take a discrete mathematics course (CS 205).



Analysis of algorithms: quiz 3

How many array accesses as a function of n ?

```
int count = 0;
for (int i = 0; i < n; i++)
   for (int j = i+1; j < n; j++)
      for (int k = 1; k <= n; k = k*2)
         if (a[i] + a[j] >= a[k])
            count++;
```

1st iteration of outer loop with counter i, inner loop with counter j executed (n–1) times, in which every time the most inner loop executed lgN times

(n-1) + (n-2) + … + 1 + 0 ==> ~ 1/2 n (n-1)

lg n

3 array accesses

A. $\sim n^2 \log_2 n$

B. $\sim 3/2 \ n^2 \log_2 n$

C. $\sim 1/2 \ n^3$

D. $\sim 3/2 \ n^3$

Analysis of algorithms: quiz 4

What is order of growth of running time as a function of n ?

```
int count = 0;
for (int i = n; i >= 1; i = i/2)
   for (int j = 1; j <= i; j++)
      count++;
```

"inner loop"

A. $O(n)$

B. $O(n \log n)$

C. $O(n^2)$

D. $O(2^n)$

frequency
n + n/2 + n/4 + … + 1
= n(1 + 1/2 + 1/4 + 1/8 + …. 1/n)
= 2n

1A. Greatest Hits of 111

- *Memory, objects, Arrays*
- *Program stack and heap*
- *Images as 2D arrays*
- *Introduction to program analysis*
- *Running time (experimental analysis)*
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

Some slides Adopted and modified from Sedgewick and Wayne

Basics

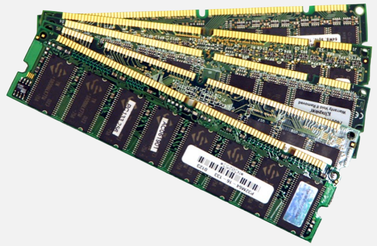
Bit. 0 or 1.

Byte. 8 bits.

NIST most computer scientists

Megabyte (MB). 1 million or 2^{20} bytes.

Gigabyte (GB). 1 billion or 2^{30} bytes.



64-bit machine. We assume a 64-bit machine with 8-byte pointers.

references/memory addresses

some JVMs "compress" ordinary object pointers to 4 bytes to avoid this cost

Typical memory usage for primitive types and arrays

LO 1.3

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

primitive types

type	bytes
boolean[]	$1n + 24$
int[]	$4n + 24$
double[]	$8n + 24$

wasteful
(but $\sim 36n$ in Python 3)

array overhead = 24 byte

one-dimensional arrays (length n)

type	bytes
boolean[][]	$\sim 1\ n^2$
int[][]	$\sim 4\ n^2$
double[][]	$\sim 8\ n^2$

tilde notation

two-dimensional arrays (n-by-n)

Typical memory usage for objects in Java

LO 1.3

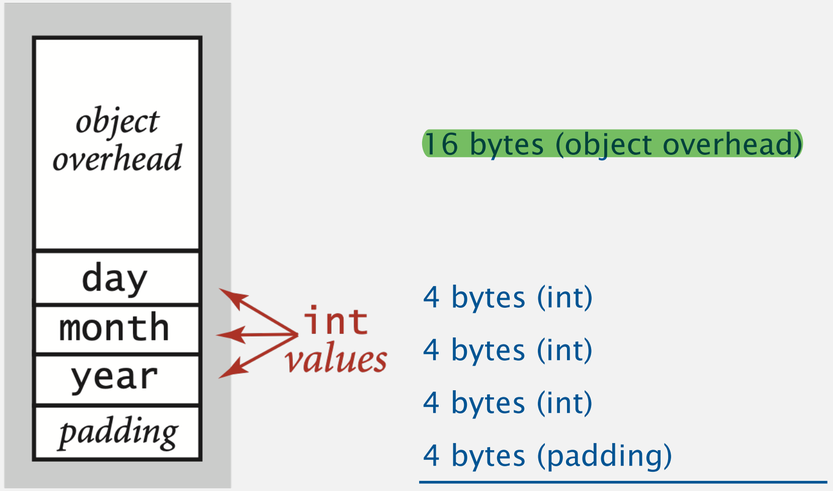
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Memory of each object rounded up to use 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;
...
}
```



Typical memory usage summary

LO 1.3

Total memory usage for a data type value in Java:

- Primitive type: 4 bytes for `int`, 8 bytes for `double`, ...
- Object reference: 8 bytes.
- Array: 24 bytes + memory for each array entry.
- Object: 16 bytes + memory for each instance variable.
- Padding: round up to multiple of 8 bytes.

Note. Depending on application, we often count the memory for any referenced objects (recursively).

"deep memory"

Analysis of algorithms: quiz 5

How much memory does a `WeightedQuickUnionUF` use as a function of n ?

A. $\sim 4\,n$ *bytes*

B. $\sim 8\,n$ *bytes*

C. $\sim 4\,n^2$ *bytes*

D. $\sim 8\,n^2$ *bytes*

```
public class WeightedQuickUnionUF
{
   private int[] parent;
   private int[] size;
   private int count;

   public WeightedQuickUnionUF(int n)
   {
      parent = new int[n];
      size   = new int[n];

      count = 0;
      for (int i = 0; i < n; i++)
         parent[i] = i;
      for (int i = 0; i < n; i++)
         size[i] = 1;
   }
   ...
}
```

constructor to create 2 arrays with capacity of n

Analysis of algorithms: quiz 5

How much memory does a `WeightedQuickUnionUF` use as a function of n ?

16 bytes (object overhead)

$8 + (4n + 24)$ bytes each (reference + `int[]` array)

4 bytes (`int`)

4 bytes (padding)

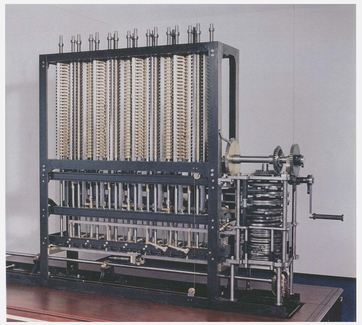
$8n + 88 \sim 8n$ bytes

```
public class WeightedQuickUnionUF
{
   private int[] parent;
   private int[] size;
   private int count;

   public WeightedQuickUnionUF(int n)
   {
      parent = new int[n];
      size   = new int[n];

      count = 0;
      for (int i = 0; i < n; i++)
         parent[i] = i;
      for (int i = 0; i < n; i++)
         size[i] = 1;
   }
   ...
}
```

Turning the crank: summary

Empirical analysis.

- Execute program to perform experiments.
- Assume power law.
- Formulate a hypothesis for running time.
- Model enables us to make predictions.



Mathematical analysis.

- Analyze algorithm to count frequency of operations.
- Use tilde and big-Theta notations to simplify analysis.
- Model enables us to explain behavior.

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \lceil n/2^{h+1} \rceil \; h \; \sim \; n$$

This course. Learn to use both.

1A. Greatest Hits of 111

- *Memory, objects, Arrays*
- *Program stack and heap*
- *Images as 2D arrays*
- *Introduction to program analysis*
- *Running time (experimental analysis)*
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

Some slides Adopted and modified from Sedgewick and Wayne

What is Garbage Collection?

LO 1.6

- Every modern programming language allows programmers to allocate new storage dynamically
- Every modern language needs facilities for reclaiming and recycling the storage used by programs
- It's usually the most complex aspect of the run-time system for any modern language like Java

Definition · A value is garbage if it will not be used in any subsequent computation by the program

Question. Is it easy to determine which objects are garbage?

Answer. No. This is an *undecidable* problem. (details later)

Technically any memory block with no active use by the program is considered garbage. But it is hard to determine which ones are not being used.

Garbage Collection of Unused Objects

- Language Support for Garbage Collection
  - Some languages like C, requires you to manage memory by using constructs like malloc() and free()
  - However, Java compiler provides automated processes to clean unused memory blocks

- Keeping Track of Memory
  - Writing code to manage memory is tedious and often error-prone.
  - If we can keep track of the number of references to each object, when the count goes to zero, we can remove the object.

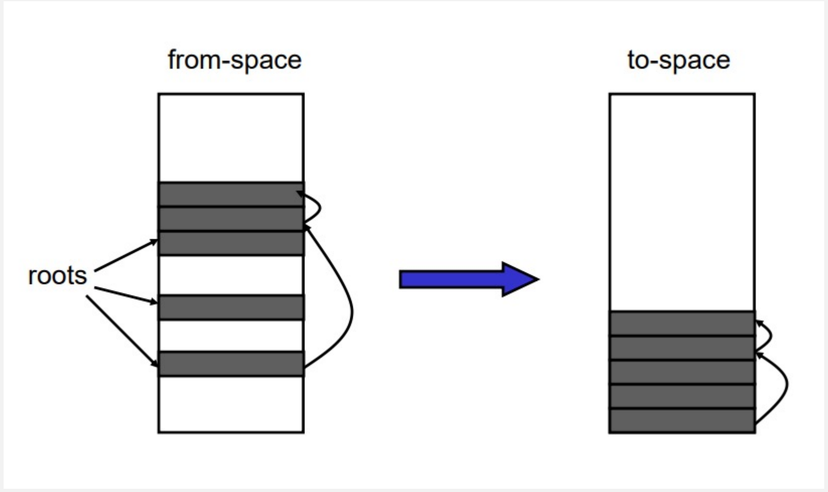
- Java Garbage Collection
  - Java saves you from having to manually clean memory by allowing programmer to create as many objects as they want (limited of course to whatever your system can handle) but never having to destroy them.
  - The Java runtime environment deletes objects when it determines that they are no longer being used. This process is known as **garbage collection**.

The Process of Freeing Memory

- The Java runtime environment supports a garbage collector that periodically frees the memory used by objects that are no longer needed.

- The Java garbage collector is a *mark-sweep* garbage collector that scans Java's dynamic memory areas for objects, marking those that are referenced.

- After all possible paths to objects are investigated, those objects that are not marked (that is, not referenced) are known to be garbage and are collected.



One representation of garbage collection/ memory cleaning

Roots keep track of usable data blocks used by program

Merged into a more contiguous block of memory

When Garbage is Collected

- **synchronously or asynchronously** . The garbage collector runs in a low priority both synchronously and asynchronously depending on the situation and the system on which Java is running.

- **Out of Memory**. The garbage collector runs synchronously when the system runs out of memory, or in response to a request from a Java Program.

- **Manual Cleaning**. We can manually ask a Java program to run the garbage collector at any time by calling System.gc() method. (not used in this course)

- **Time to Clean**. Garbage collector requires about 20 milliseconds to complete its tasks so, your program should run the garbage collector if the performance of the program is not impacted by it.

- **Guarantee**. Asking Garbage collector to run may not guarantee that your objects will be garbage collected

- The Java Garbage collector runs asynchronously when the system is idle.
- **Later.** More on these memory management topics will be covered in systems courses and compiler courses

INTRODUCTION TO DATA STRUCTURES and ALGORITHMS

Rutgers University

1. Greatest Hits of 111

- *Memory, objects, Arrays*
- *Program stack and heap*
- *Images as 2D arrays*
- *Introduction to program analysis*
- *Running time (experimental analysis)*
- *Running time (mathematical models)*
- *Memory usage*
- *Garbage collection in Java*

https://ds.cs.rutgers.edu

Last updated on 1/8/21