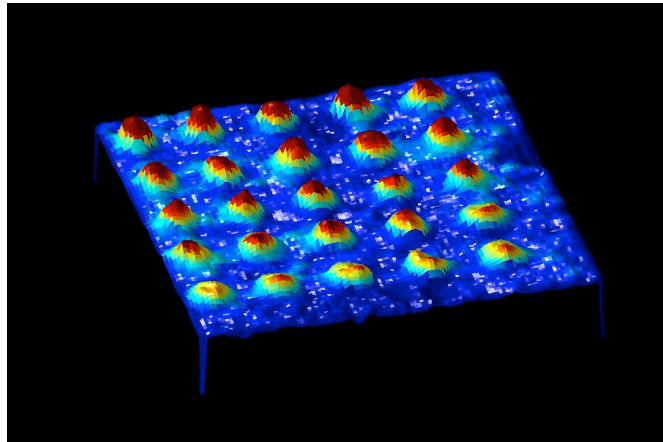


# 1.4 Arrays



Arrays

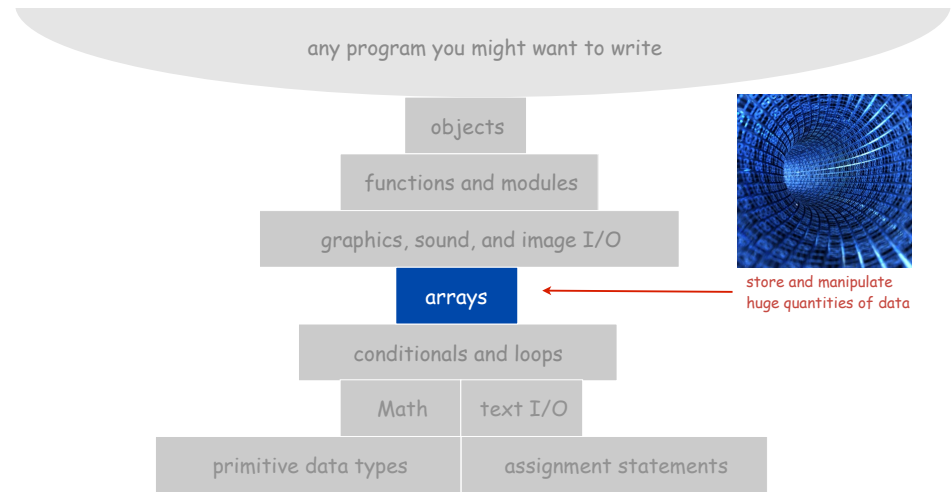
**This lecture.** Store and manipulate huge quantities of data.

**Array.** Indexed sequence of values of the same type.

**Examples.**

- 52 playing cards in a deck.
- 5 thousand undergrads at Princeton.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 50 trillion cells in the human body.
- $6.02 \times 10^{23}$  particles in a mole.

index	value
0	doug
1	cmoretti
2	dgabai
3	maia
4	jcnelson
5	cmmiller
6	tpereira
7	dkthree
8	adrdr
9	soumyade
10	idavey
11	drutskoy
12	mbrowni



Many Variables of the Same Type

**Goal.** 10 variables of the same type.

```
// Tedious and error-prone code.
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
...
a4 = 3.0;
...
a8 = 8.0;
...
double x = a4 + a8;
```

## Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// Easy alternative.
double[] a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```

← declares, creates, and initializes  
[stay tuned for details]

## Many Variables of the Same Type

Goal. 1 million variables of the same type.

```
// Scales to handle large arrays.
double[] a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```

## Arrays in Java

Java has special language support for arrays.

- To make an array: **declare**, **create**, and **initialize** it.
- To access element *i* of array named *a*, use *a[i]*.
- Array indices start at 0.

```
int N = 1000;
double[] a;           // declare the array
a = new double[N];    // create the array
for (int i = 0; i < N; i++) // initialize the array
    a[i] = 0.0;       // all to 0.0
```

## Arrays in Java

Java has special language support for arrays.

- To make an array: **declare**, **create**, and **initialize** it.
- To access element *i* of array named *a*, use *a[i]*.
- Array indices start at 0.

```
int N = 1000;
double[] a;           // declare the array
a = new double[N];    // create the array
for (int i = 0; i < N; i++) // initialize the array
    a[i] = 0.0;       // all to 0.0
```

**Compact alternatives:** Declare, create, and initialize in one statement.

- Default: all entries automatically set to 0.

```
double[] a = new double[1000];
```

- Initialize to literal values

```
double[] x = { 0.3, 0.6, 0.1 };
```

compact version      equivalent code

```
double[] x = new double[3];
x[0] = 0.3; x[1] = 0.6; x[2] = 0.1;
```

## Sample Array Code: Vector Dot Product

**Dot product.** Given two vectors  $x[]$  and  $y[]$  of length  $n$ , their **dot product** is the sum of the products of their corresponding components.

```
double[] x = { 0.3, 0.6, 0.1 };
double[] y = { 0.5, 0.1, 0.4 };

double sum = 0.0;
for (int i = 0; i < N; i++)
    sum += x[i]*y[i];
```

<u>i</u>	<u>x[i]</u>	<u>y[i]</u>	<u>x[i]*y[i]</u>	<u>sum</u>
				0
0	.30	.50	.15	.15
1	.60	.10	.06	.21
2	.10	.40	.04	.25
				.25

## Mumbo-Jumbo Demystification, Part 1

```
public class Gambler {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);
        . . .
    }
}
```

## Array Processing Examples

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

create an array with N random values

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

print the array values, one per line

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

find the maximum of the array values

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

copy to another array

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

compute the average of the array values

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

reverse the elements within the array

## Shuffling a Deck



## Setting Array Values at Compile Time

Ex. Print a random card.

```
String[] rank =
{
    "2", "3", "4", "5", "6", "7", "8", "9",
    "10", "Jack", "Queen", "King", "Ace"
};

String[] suit =
{
    "Clubs", "Diamonds", "Hearts", "Spades"
};

int i = (int) (Math.random() * 13); // between 0 and 12
int j = (int) (Math.random() * 4); // between 0 and 3

System.out.println(rank[i] + " of " + suit[j]);
```

## Array Challenge 2

Swap the **for** statements: **rank** index in inner loop, **suit** index in outer loop.  
Now, in which order are they printed?

```
String[] rank = { "2", "3" ..., "King", "Ace" };
String[] suit =
{ "clubs", "diamonds", "hearts", "spades" };

String[] deck = new String[52];
for (int j = 0; j < 4; j++)
    for (int i = 0; i < 13; i++)
        deck[4*i + j] = rank[i] + " of " + suit[j];

for (int i = 0; i < 52; i++)
    System.out.println(deck[i]);
```

*these lines swapped from previous slide*

- |    |               |    |            |
|----|---------------|----|------------|
| A. | 2 of clubs    | B. | 2 of clubs |
|    | 2 of diamonds |    | 3 of clubs |
|    | 2 of hearts   |    | 4 of clubs |
|    | 2 of spades   |    | 5 of clubs |
|    | 3 of clubs    |    | 6 of clubs |
|    | ...           |    | ...        |

## Array Challenge 1

The following code sets array values to the 52 card values and prints them.  
In which order are they printed?

```
String[] rank = { "2", "3" ..., "King", "Ace" };
String[] suit =
{ "clubs", "diamonds", "hearts", "spades" };

String[] deck = new String[52];
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 4; j++)
        deck[4*i + j] = rank[i] + " of " + suit[j];

for (int i = 0; i < 52; i++)
    System.out.println(deck[i]);
```

← typical array processing code changes values at runtime

- |    |               |    |            |
|----|---------------|----|------------|
| A. | 2 of clubs    | B. | 2 of clubs |
|    | 2 of diamonds |    | 3 of clubs |
|    | 2 of hearts   |    | 4 of clubs |
|    | 2 of spades   |    | 5 of clubs |
|    | 3 of clubs    |    | 6 of clubs |
|    | ...           |    | ...        |

## Array Challenge 3

The following code sets array values to the 52 card values and prints them.  
What change to the code will produce the "B" order?

```
String[] rank = { "2", "3" ..., "King", "Ace" };
String[] suit =
{ "clubs", "diamonds", "hearts", "spades" };

String[] deck = new String[52];
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 4; j++)
        deck[4*i + j] = rank[i] + " of " + suit[j];

for (int i = 0; i < 52; i++)
    System.out.println(deck[i]);
```

- |    |               |    |            |
|----|---------------|----|------------|
| A. | 2 of clubs    | B. | 2 of clubs |
|    | 2 of diamonds |    | 3 of clubs |
|    | 2 of hearts   |    | 4 of clubs |
|    | 2 of spades   |    | 5 of clubs |
|    | 3 of clubs    |    | 6 of clubs |
|    | ...           |    | ...        |

## Shuffling

Goal. Given an array, rearrange its elements in **random** order.

### Shuffling algorithm.

- In iteration  $i$ , pick random card from `deck[i]` through `deck[N-1]`, with each card equally likely.
- Exchange it with `deck[i]`.

```
int N = deck.length;
for (int i = 0; i < N; i++)
{
    int r = i + (int) (Math.random() * (N-i));
    String t = deck[r];
    deck[r] = deck[i];
    deck[i] = t;
}
```

swap idiom between  $i$  and  $N-1$

## Shuffle an Array

### Shuffle a deck of cards.

- In  $i^{\text{th}}$  iteration, put a random element from remainder of deck at index  $i$ .
  - choose random integer  $r$  between  $i$  and  $N-1$
  - swap values in positions  $r$  and  $i$

Array index	0	1	2	3	4	5	6	7	8	9
Value	9♣	3♣	4♣	5♣	6♣	7♣	8♣	2♣	10♣	J♣

random integer = 7

## Shuffling a Deck of Cards

```
public class Deck
{
    public static void main(String[] args)
    {
        String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };
        String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9",
            "10", "Jack", "Queen", "King", "Ace" };
        int SUITS = suit.length;
        int RANKS = rank.length;
        int N = SUITS * RANKS;

        String[] deck = new String[N];
        for (int i = 0; i < RANKS; i++)
            for (int j = 0; j < SUITS; j++)
                deck[SUITS*i + j] = rank[i] + " of " + suit[j];

        for (int i = 0; i < N; i++)
        {
            int r = i + (int) (Math.random() * (N-i));
            String t = deck[r];
            deck[r] = deck[i];
            deck[i] = t;
        }

        for (int i = 0; i < N; i++)
            System.out.println(deck[i]);
    }
}
```

## Shuffling a Deck of Cards

```
% java Deck
5 of Clubs
Jack of Hearts
9 of Spades
10 of Spades
9 of Clubs
7 of Spades
6 of Diamonds
7 of Hearts
7 of Clubs
4 of Spades
Queen of Diamonds
10 of Hearts
5 of Diamonds
Jack of Clubs
Ace of Hearts
...
5 of Spades
```

```
% java Deck
10 of Diamonds
King of Spades
2 of Spades
3 of Clubs
4 of Spades
Queen of Clubs
2 of Hearts
7 of Diamonds
6 of Spades
Queen of Spades
3 of Spades
Jack of Diamonds
6 of Diamonds
8 of Spades
9 of Diamonds
...
10 of Spades
```

# Coupon Collector



## Coupon Collector: Java Implementation

```
public class CouponCollector
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        int cardcnt = 0; // number of cards collected
        int valcnt = 0; // number of distinct cards

        // Do simulation.
        boolean[] found = new boolean[N];
        while (valcnt < N)
        {
            int val = (int) (Math.random() * N);
            cardcnt++;
            if (!found[val])
            {
                valcnt++;
                found[val] = true;
            }
        }

        // all N distinct cards found
        System.out.println(cardcnt);
    }
}
```

## Coupon Collector Problem

**Coupon collector problem.** Given  $N$  different card types, how many do you have to collect before you have (at least) one of each type?



assuming each possibility is equally likely for each card that you collect

**Simulation algorithm.** Repeatedly choose an integer  $i$  between 0 and  $N-1$ . Stop when we have at least one card of every type.

**Q.** How to check if we've seen a card of type  $i$ ?

**A.** Maintain a boolean array so that `found[i]` is true if we've already collected a card of type  $i$ .

## Coupon Collector: Debugging

**Debugging.** Add code to print contents of **all** variables.

val	found						valcnt	cardcnt
	0	1	2	3	4	5		
	F	F	F	F	F	F	0	0
2	F	F	T	F	F	F	1	1
0	T	F	T	F	F	F	2	2
4	T	F	T	F	T	F	3	3
0	T	F	T	F	T	F	3	4
1	T	T	T	F	T	F	4	5
2	T	T	T	F	T	F	4	6
5	T	T	T	F	T	T	5	7
0	T	T	T	F	T	T	5	8
1	T	T	T	F	T	T	5	9
3	T	T	T	T	T	T	6	10

**Challenge.** Debugging with arrays requires tracing many variables.

## Coupon Collector: Mathematical Context

**Coupon collector problem.** Given  $N$  different possible cards, how many do you have to collect before you have (at least) one of each type?

**Fact.** About  $N (1 + 1/2 + 1/3 + \dots + 1/N) \sim N \ln N$

see ORF 245 or COS 341

**Ex.**  $N = 30$  baseball teams. Expect to wait  $\approx 120$  years before all teams win a World Series.

under idealized assumptions

## Coupon Collector: Scientific Context

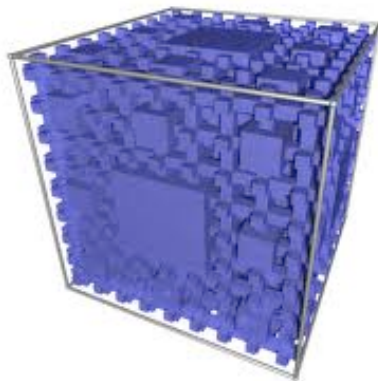
**Q.** Given a sequence from nature, does it have same characteristics as a random sequence?

**A.** No easy answer - many tests have been developed.

**Coupon collector test.** Compare number of elements that need to be examined before all values are found against the corresponding answer for a random sequence.



## Multidimensional Arrays



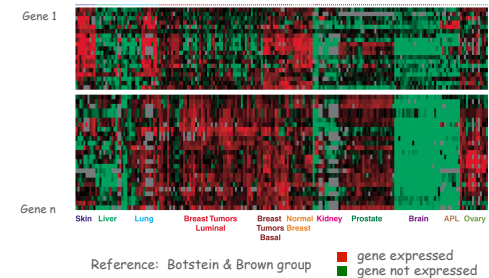
## Two Dimensional Arrays

**Two dimensional arrays.**

- Table of data for each experiment and outcome.
- Table of grades for each student and assignments.
- Table of grayscale values for each pixel in a 2D image.

**Mathematical abstraction.** Matrix.

**Java abstraction.** 2D array.



## Two Dimensional Arrays in Java

Declare, create, initialize. Like 1D, but add another pair of brackets.

```
int M = 10;
int N = 3;
double[][] a = new double[M][N];
```

Array access.

Use `a[i][j]` to access entry in row `i` and column `j`.  
Indices start at 0.

Initialize.

This code is implicit (sets all entries to 0).

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
        a[i][j] = 0.0;
```

`a[][]` →

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]
a[3][0]	a[3][1]	a[3][2]
a[4][0]	a[4][1]	a[4][2]
a[5][0]	a[5][1]	a[5][2]
a[6][0]	a[6][1]	a[6][2]
a[7][0]	a[7][1]	a[7][2]
a[8][0]	a[8][1]	a[8][2]
a[9][0]	a[9][1]	a[9][2]

A 10-by-3 array

**Warning.** This implicit code might slow down your program for big arrays.

## Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

```
double[][] p =
{
    { .02, .92, .02, .02, .02 },
    { .02, .02, .32, .32, .32 },
    { .02, .02, .02, .92, .02 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 }
};
```

`a[1][3]` →

	.92	.02	.02	.02	.02
row 1 →	.02	.92	.32	.32	.32
	.02	.02	.02	.92	.02
	.92	.02	.02	.02	.02
	.47	.02	.47	.02	.02

↑  
column 3

## Matrix Addition

**Matrix addition.** Given two `N`-by-`N` matrices `a` and `b`, define `c` to be the `N`-by-`N` matrix where `c[i][j]` is the sum `a[i][j] + b[i][j]`.

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

`a[][]`

.70	.20	.10
.30	.60	.10
.50	.10	.40

← `a[1][2]`

`b[][]`

.80	.30	.50
.10	.40	.10
.10	.30	.40

← `b[1][2]`

`c[][]`

1.5	.50	.60
.40	1.0	.20
.60	.40	.80

← `c[1][2]`

## Matrix Multiplication

**Matrix multiplication.** Given two `N`-by-`N` matrices `a` and `b`, define `c` to be the `N`-by-`N` matrix where `c[i][j]` is the dot product of the `i`<sup>th</sup> row of `a` and the `j`<sup>th</sup> row of `b`.

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for (int k = 0; k < N; k++)
            c[i][j] += a[i][k] * b[k][j];
```

all values initialized to 0

`a[][]`

.70	.20	.10
.30	.60	.10
.50	.10	.40

← row 1

`b[][]`

.80	.30	.50
.10	.40	.10
.10	.30	.40

↓  
column 2

`c[][]`

.59	.32	.41
.31	.36	.25
.45	.31	.42

`c[1][2] = .3 * .5`  
+ `.6 * .1`  
+ `.1 * .4`  
= `.25`



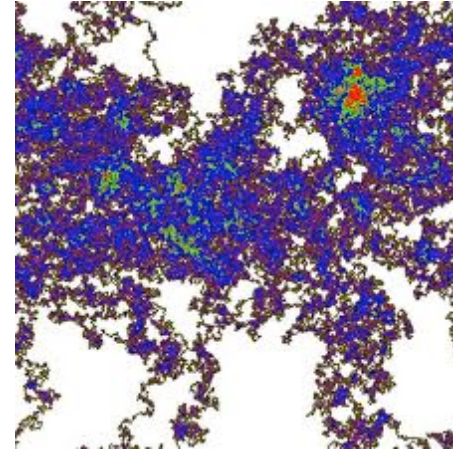
## Array Challenge 4

How many multiplications to multiply two N-by-N matrices?

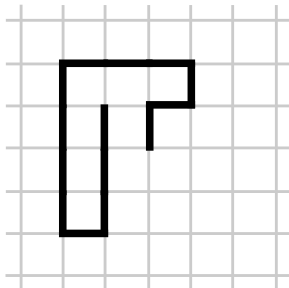
```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      c[i][j] += a[i][k] * b[k][j];
```

- A. N
- B.  $N^2$
- C.  $N^3$
- D.  $N^4$

## Application: 2D Random Walks



### Application: Self-Avoiding Walks



### Self-Avoiding Walk

#### Model.

- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previously visited intersections.
- Two possible outcomes: **escape** and **dead end**



**Applications.** Polymers, statistical mechanics, etc.

- Q. What fraction of time will you escape in an 5-by-5 lattice?
- Q. In an N-by-N lattice?
- Q. In an N-by-N-by-N lattice?

## Self-Avoiding Walk: Implementation

```
public class SelfAvoidingWalk
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]); // lattice size
        int T = Integer.parseInt(args[1]); // number of trials
        int deadEnds = 0; // trials ending at dead end

        for ( each trial )
        {
            boolean[][] a = new boolean[N][N]; // intersections visited
            int x = N/2, y = N/2; // current position

            while ( you're still inside the lattice )
            {
                if ( you're at a dead end )
                { deadEnds++; break; }

                a[x][y] = true; // mark as visited

                double r = Math.random();
                if (r < 0.25) { if (!a[x+1][y]) x++; }
                else if (r < 0.50) { if (!a[x-1][y]) x--; }
                else if (r < 0.75) { if (!a[x][y+1]) y++; }
                else if (r < 1.00) { if (!a[x][y-1]) y--; }
            }
            System.out.println(100*deadEnds/T + "% dead ends");
        }
    }
}
```

dead end

take a random  
step to a new  
intersection

## Self-Avoiding Walk: Implementation

```
public class SelfAvoidingWalk
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]); // lattice size
        int T = Integer.parseInt(args[1]); // number of trials
        int deadEnds = 0; // trials ending at dead end

        for (int t = 0; t < T; t++)
        {
            boolean[][] a = new boolean[N][N]; // intersections visited
            int x = N/2, y = N/2; // current position

            while (x > 0 && x < N-1 && y > 0 && y < N-1)
            {
                if (a[x-1][y] && a[x+1][y] && a[x][y-1] && a[x][y+1])
                { deadEnds++; break; }

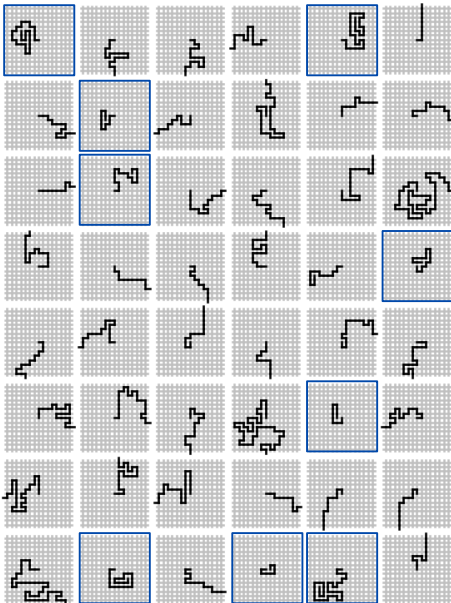
                a[x][y] = true; // mark as visited

                double r = Math.random();
                if (r < 0.25) { if (!a[x+1][y]) x++; }
                else if (r < 0.50) { if (!a[x-1][y]) x--; }
                else if (r < 0.75) { if (!a[x][y+1]) y++; }
                else if (r < 1.00) { if (!a[x][y-1]) y--; }
            }
            System.out.println(100*deadEnds/T + "% dead ends");
        }
    }
}
```

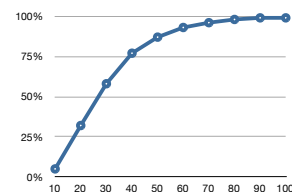
dead end

take a random  
step to a new  
intersection

## Self-Avoiding Walks



```
% java SelfAvoidingWalk 10 100000
5% dead ends
% java SelfAvoidingWalk 20 100000
32% dead ends
% java SelfAvoidingWalk 30 100000
58% dead ends
% java SelfAvoidingWalk 40 100000
77% dead ends
% java SelfAvoidingWalk 50 100000
87% dead ends
% java SelfAvoidingWalk 60 100000
93% dead ends
% java SelfAvoidingWalk 70 100000
96% dead ends
% java SelfAvoidingWalk 80 100000
98% dead ends
% java SelfAvoidingWalk 90 100000
99% dead ends
% java SelfAvoidingWalk 100 100000
99% dead ends
```



## Summary

### Arrays.

- Organized way to store huge quantities of data.
- Almost as easy to use as primitive types.
- Can directly access an element given its index.

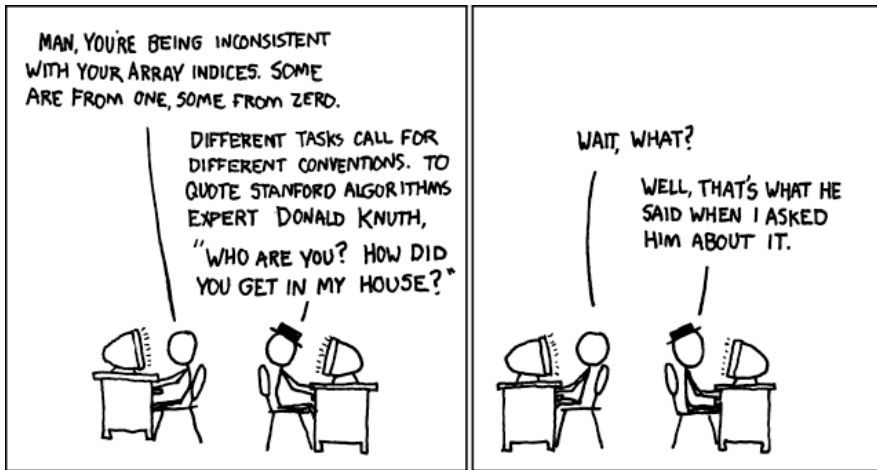
### Caveats:

- Need to fix size of array ahead of time.
- Don't forget to allocate memory with `new`.
- Indices start at 0 not 1.
- Out-of-bounds to access `a[-1]` or `a[N]` of N element array.
  - in Java: `ArrayIndexOutOfBoundsException`
  - in C: "ghastly error"

**Ahead.** Reading in large quantities of data from a file into an array.

## Off by One

"You're always off by 1 in this business." - J. Morris



[http://imgs.xkcd.com/comics/donald\\_knuth.png](http://imgs.xkcd.com/comics/donald_knuth.png)