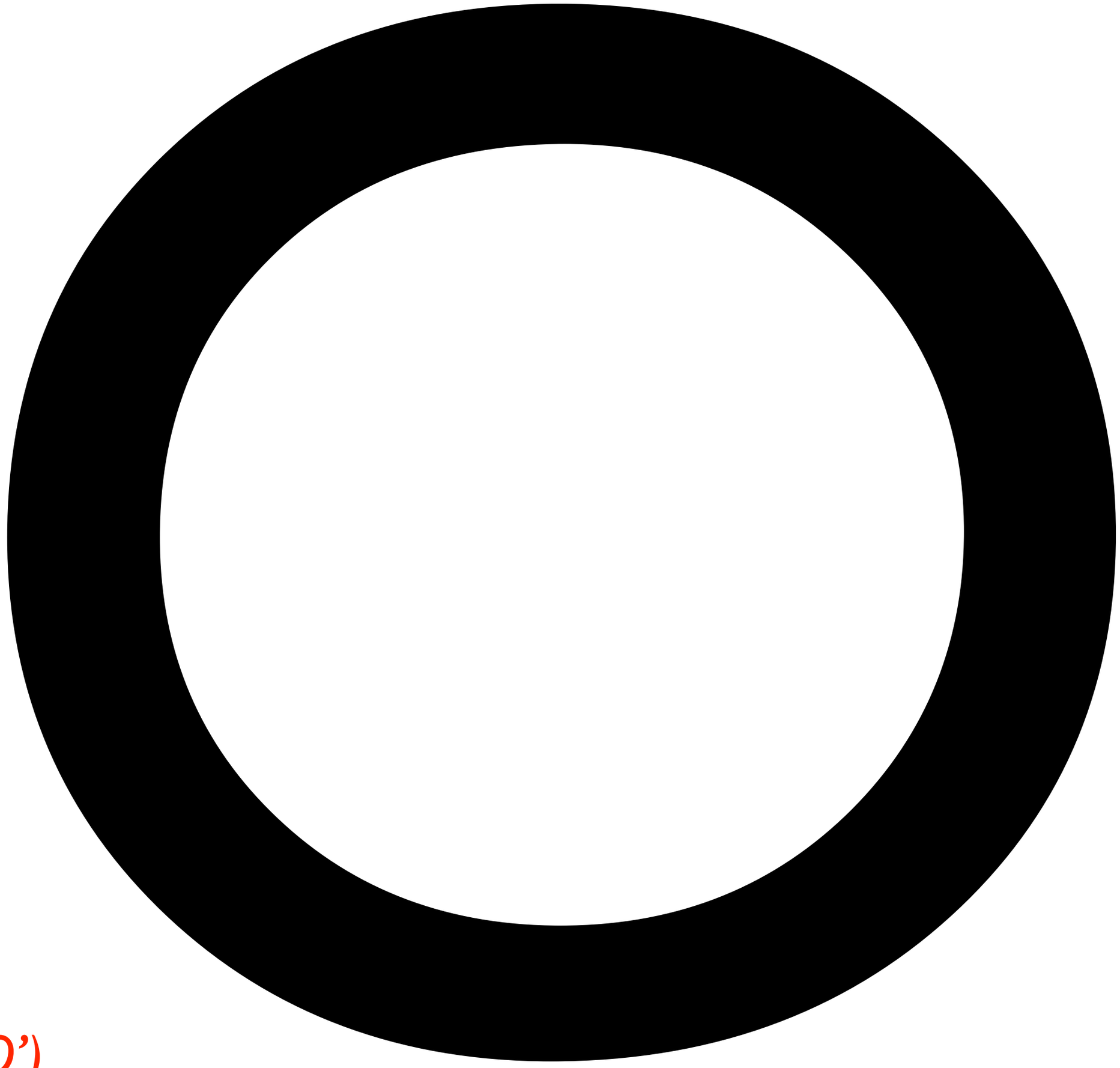


(It's an 'O')

A Digression

A Digression

```
if (false); {
   System.out.println("THIS IS IMPOSSIBLE");
}
```

A Digression

```
if (false); {
   System.out.println("THIS IS IMPOSSIBLE");
}
```

```
THIS IS IMPOSSIBLE
THIS IS IMPOSSIBLE
THIS IS IMPOSSIBLE

(etc.)
```

A Digression

ARGH

```
if (false); {
   System.out.println("THIS IS IMPOSSIBLE");
}
```

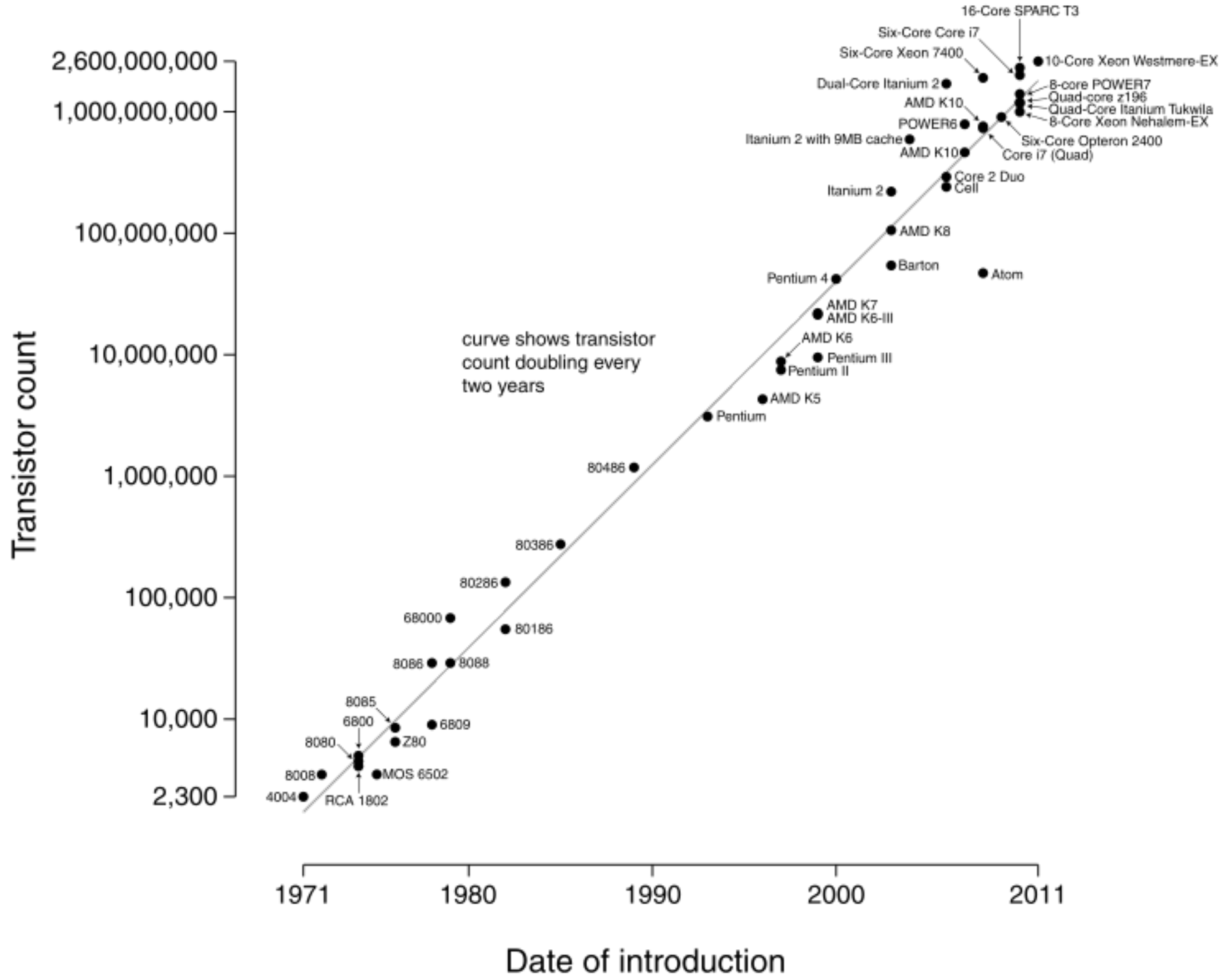
```
THIS IS IMPOSSIBLE
THIS IS IMPOSSIBLE
THIS IS IMPOSSIBLE

(etc.)
```

News!

Jotto extended until Wednesday

What's wrong with milliseconds?

What's wrong with milliseconds?



"Wait 18 months" is a terrible meta-algorithm for faster code!

Lessons from last time

Count *operations*, not ms

How it scales is the important thing

Lessons from last time

Count *operations*, not ms

How it scales is the important thing

```
public void easyThing(String[] strings) {
   System.out.println("Hello!");
}
```

Of length N

Lessons from last time

Count *operations*, not ms

How it scales is the important thing

```
public void easyThing(String[] strings) {
   System.out.println("Hello!");
}
```

Of length N

```
public void harderThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
      System.out.println(strings[i]);
   }
}
```

Lessons from last time

```
public void easyThing(String[] strings) {
  System.out.println("Hello!");
}
```

Suppose printing a string takes a constant *C* units of computation.

```
public void harderThing(String[] strings) {
  for (int i = 0; i < strings.length; ++i) {
    System.out.println(strings[i]);
  }
}
```

Lessons from last time

```
public void easyThing(String[] strings) {
  System.out.println("Hello!");
}
```

Suppose printing a string takes a constant *C* units of computation.

```
public void harderThing(String[] strings) {
  for (int i = 0; i < strings.length; ++i) {
    System.out.println(strings[i]);
  }
}
```

```
public void harderThing2(String[] strings) {
  for (int i = 0; i < strings.length; ++i) {
    for (int j = 0 ; j < strings.length ; ++j) {
      System.out.println(strings[j]);
    }
  }
}
```

What's the cost of each method?

Constants

Many things take *constant time*:

- Any (single) operation on a primitive (+, -, =, etc.)
- Method calls
- Subscripting (i.e. foo[5])
- Conditionals
- Declaring Variables
- Pointer assignment

```
public void harderThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
      System.out.println(strings[i]);
   }
}
```

Of length N

Treat printing like it costs *C*, as well.

Constants

Many things take *constant time*:

- Any (single) operation on a primitive (+, -, =, etc.)
- Method calls
- Subscripting (i.e. foo[5])
- Conditionals
- Declaring Variables
- Pointer assignment

```
public void harderThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
      System.out.println(strings[i]);
   }
}
```

Of length N

Treat printing like it costs *C*, as well.

I variable declaration. I integer assignment. N+1 integer comparisons. N integer increments. N method calls. N subscripts.

Constants

```
public void harderThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
     System.out.println(strings[i]);
   }
}
```

1 variable declaration. 1 integer assignment. N+1 integer comparisons. N integer increments. N method calls. N subscripts.

Constants

```
public void harderThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
     System.out.println(strings[i]);
   }
}
```

1 variable declaration. 1 integer assignment. N+1 integer comparisons. N integer increments. N method calls. N subscripts.

$$c_1 + c_2 + (n+1)c_3 + nc_4 + nc_5 + nc_6$$

Constants

```
public void harderThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
     System.out.println(strings[i]);
   }
}
```

1 variable declaration. 1 integer assignment. N+1 integer comparisons. N integer increments. N method calls. N subscripts.

$$c_1 + c_2 + (n+1)c_3 + nc_4 + nc_5 + nc_6$$

$$c + nc$$

Constants

```
public void harderThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
     System.out.println(strings[i]);
   }
}
```

1 variable declaration. 1 integer assignment. N+1 integer comparisons. N integer increments. N method calls. N subscripts.

$$c_1 + c_2 + (n+1)c_3 + nc_4 + nc_5 + nc_6$$

$$c + nc$$

$$nc$$

$$O(N)$$

An example

Calculate the running time of this method as a function of *N*.

```
public void trickyThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
      for (int j = i+1 ; j < strings.length ; ++j) {
         System.out.println(strings[j]);
      }
   }
}
```

An example

```
public void trickyThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
      for (int j = i+1 ; j < strings.length ; ++j) {
         System.out.println(strings[j]);
      }
   }
}
```

$O(N)$

An example

```
public void trickyThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
      for (int j = i+1 ; j < strings.length ; ++j) {
        System.out.println(strings[j]);
      }
   }
}
```

$O(N)$

$O(N^2)$

Notational note: constants are $O(1)$

What about...

SandwichBar

IsoMorphicWords

BasketWithApples

commonCount
(from Jotto)

My
CirclesCountry
Solution
(on the calendar page)

A hard one

```
public void trickyThing(String[] strings) {
   for (int i = 0; i < strings.length; ++i) {
      for (int j = 0 ; j < strings.length ; j *= 2) {
         System.out.println(strings[j]);
      }
   }
}
```