Sorting & Induction

Eleanor Birrell

February 2, 2018

"Organizing is what you do before you do something, so that when you do it, it is not all mixed up."

~ A. A. Milne

Why Sorting?

- Sorting is useful
  - Database indexing
  - Scheduling
  - Operations research
  - Compression

Some Sorting Algorithms

There are lots of ways to sort

- Insertion sort
- Selection sort
- Merge sort
- Quick sort
- And more …

There isn't one right answer

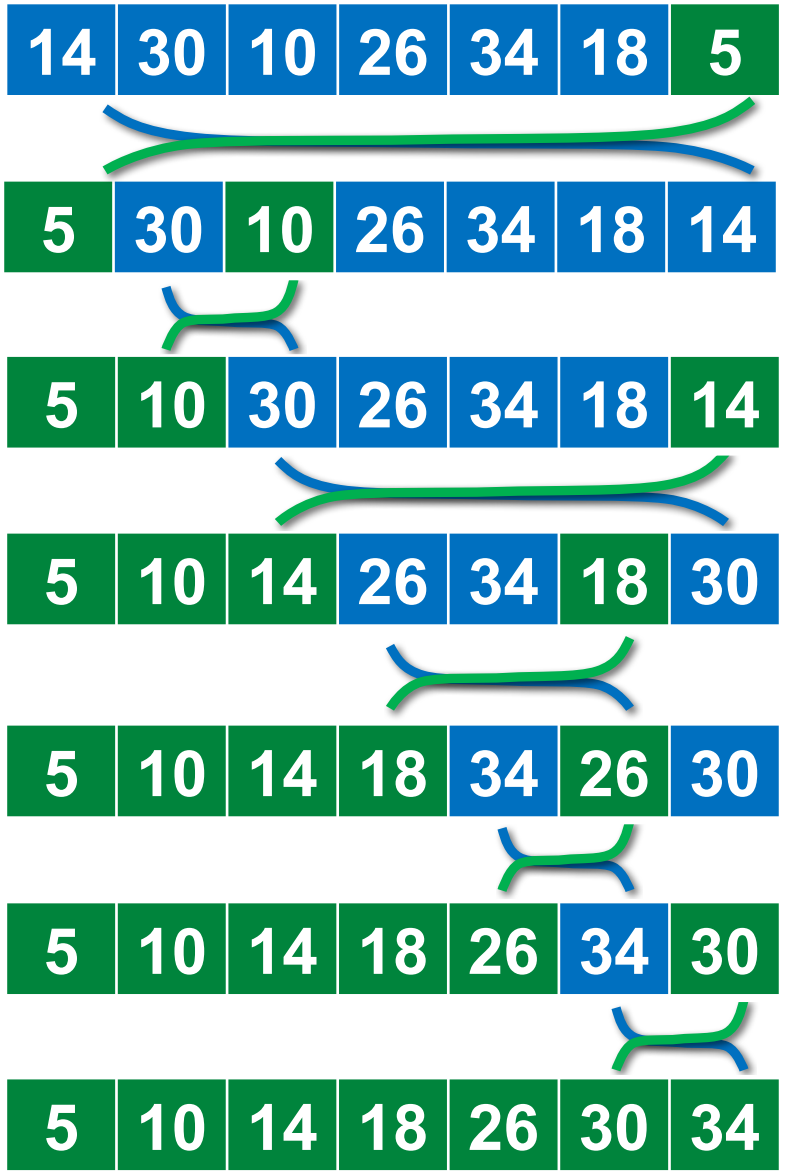
You need to be able to figure out the options and decide which one is right for your application.

Selection Sort

```
for each i:

  int m= index of min of b[i..];

  Swap b[i] and b[m];
```



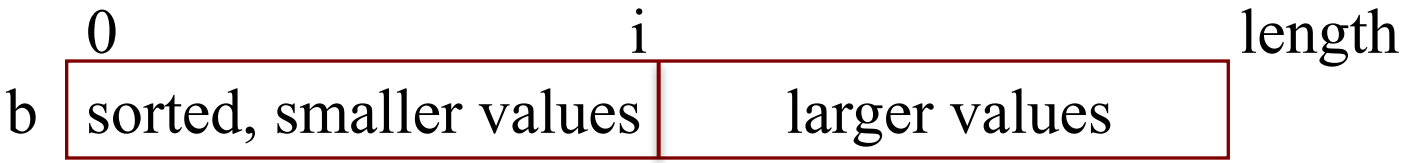
Recursive Selection Sort

```
selectionSort(int[] b, int i){
  if(i > 0){
    //Recursively sort b[0..i-1]
    selectionSort(b, i-1);
  }

  // find index of min b[i..]
  int m= indexOfMin(b,i);

  swap b[i] and b[m]
}
```

14	30	10	26	34	18	5
5	30	10	26	34	18	14
5	10	30	26	34	18	14
5	10	14	26	34	18	30
5	10	14	18	34	26	30
5	10	14	18	26	34	30
5	10	14	18	26	34	30
5	10	14	18	26	30	34



Induction

- Mathematical proof technique
- To prove propositions:

$$\forall n \geq c,\ P(n) \text{ is true}$$

1) Prove Base Case: $i = c$
2) Prove Induction Case $\forall i > c, P(i-1) \Rightarrow P(i)$

Induction Example

- Prove $\forall n \geq 0,\ n < 2^n$

- Prove $\forall n \geq 1, 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$

Correctness of Recursive Selection Sort

```
selectionSort(int[] b, int i){
  if(i > 0){
    //Recursively sort b[0..i-1]
    selectionSort(b, i-1);
  }

  // find index of min b[i..]
  int m= indexOfMin(b,i);

  // swap b[i] and b[m]
  int temp= b[i];
  b[i]= b[m];
  b[m]= temp;
}
```

Complexity of Recursive SelectionSort

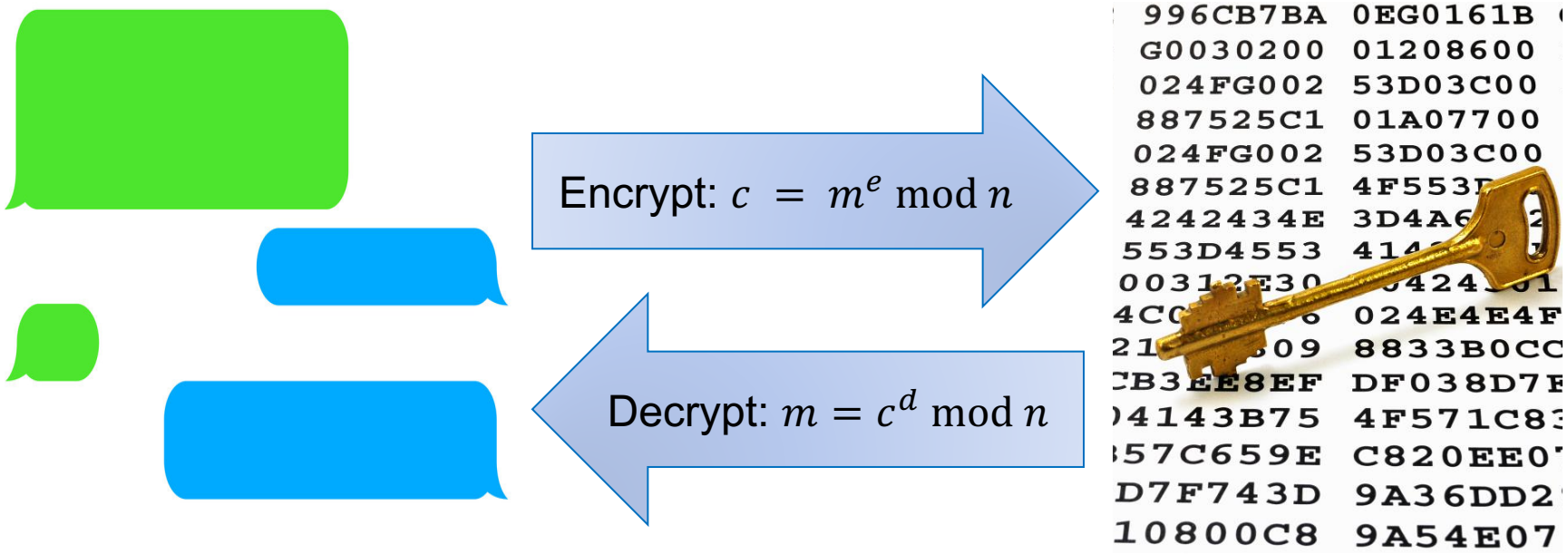
- To compute the running time of this algorithm, we need to count the number of comparisons in each recursive call
- All of the comparison are in `indexOfMin(b,i)`
  - that makes _____ comparisons
- Selection sort takes time _____

Performance

Algorithm	Time	Space	Stable?
Selection Sort			
Insertion Sort			
Merge Sort			
Heap Sort			
Quick Sort			

More fun with induction…

RSA


Encrypt: $c = m^e \bmod n$
Decrypt: $m = c^d \bmod n$
996CB7BA 0EG0161B
G0030200 01208600
024FG002 53D03C00
887525C1 01A07700
024FG002 53D03C00
887525C1 4F553D
4242434E 3D4A6
553D4553
024E4E4F
8833B0CC
DF038D7E
4F571C83
C820EE0
9A36DD2
9A54E07

Fast Power

```
fastPower(int x, int n){
  if(n == 0){ return 1; }

  // handle odd values
  if(n % 2 == 1){
    return x * fastPower(x, n-1);
  }

  // handle even values
  return fastPower(x*x, n/2);
}
```

Strong Induction

Mathematical proof technique

To prove propositions:

$$\forall n \geq c,\ P(n) \text{ is true}$$

1) Prove Base Case:
   $i = c$
2) Prove Induction Case:
   $\forall i > c,\ (\forall c \leq j < i,\ P(j)) \Rightarrow P(i)$

Correctness of Fast Power

- Base Case: i=0

- Induction Case: $\forall i > c,\ (\forall c \leq j < i,\ P(j)) \Rightarrow P(i)$

  - If i is odd:

  - If i is even:

Square-and-Multiply

```
sqrAndMul(int x, int n){
  if(n == 0){ return 1; }

  // handle odd values
  if(n % 2 == 1){
    return x * sqrAndMul(x*x, (n-1)/2);
  }

  // handle even values
  return sqrAndMul(x*x, n/2);
}
```

We can iteratively compute exponentiation with square (and optional multiply) and bit shifting!

Iterative Square-and-Multiply

- Example: compute 5^{13} (13 is 1101 in binary)

1.
2.
3.
4.
5.

Side Channel Attacks