CSCI 241: Data Structures

Lecture 2

Insertion and Selection Sort:
Runtime analysis

Announcements

Quiz time!

- On review topics.
- Will be graded, but credit is 1/0 for participation.
- 10 minutes

Last Time

- Two sorting algorithms:
    - Insertion sort
        - Push the next unsorted element into its sorted position
    - Selection sort
        - Find the next smallest element and put it into its final position.

Insertion sort: Pseudocode

```
// Sorts A using insertion sort
insertionSort(A):
  i = 0;
  while i < A.length:
    j = i;
    while j > 0 and A[j] < A[j-1]:
      swap(A[j], A[j-1])
      j--
    i++
```

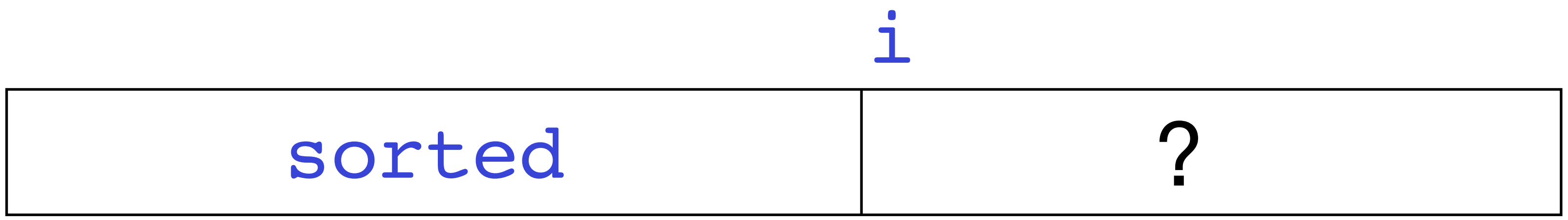
Sort the following array using **insertion** sort:

[1 4 8 2 6]

How many times did you swap two elements?

A. 3
B. 4
C. 6
D. 8

Invariant: A



i

sorted

?

Selection Sort

```
selectionSort(A):
  i = 0;
  while i < A.length:
    // find min of A[i..A.length]
    // swap it with A[i]
    // increment i
```

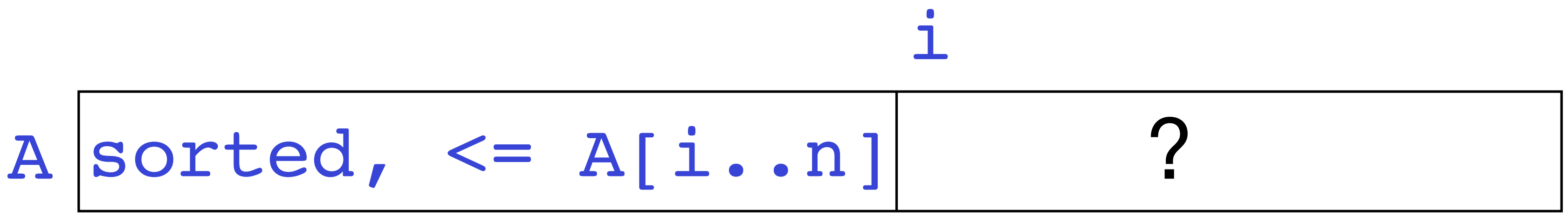
Sort the following array using **selection** sort:

[1 4 8 2 6]

How many times did you swap two distinct elements?

A. 2
B. 3
C. 4
D. 5

Invariant:



i

A | sorted, <= A[i..n] | ?

Practice Problems

1. Write code for Selection Sort

2. Consider the array:

[8 4 6 10 7 1 2]

Write the state of the array at the conclusion of the loop iteration in which i == 4 (don’t forget arrays are 0-indexed!).

InsertionSort:

SelectionSort:

Which sort should we use?

- Which one takes less time?

- Which one takes less memory?

- Other considerations?

How do we measure these things?

- Which one takes less time?

- Which one takes less memory?

- Other considerations?

How should we measure runtime?

How many ways can you think of to describe the runtime of an algorithm?

```
public int findMax(int[] a) {
  int currentMax = a[0];
  for (int i = 1; i < a.length; i++) {
    if (currentMax < a[i]) {
      currentMax = a[i];
    }
  }
  return currentMax;
}
```

How should we measure runtime?

How about metrics that are **invariant** to

- Length of the array a?
- How fast your computer is?

Approach: count the number of “operations” the computer needs to execute.

- Count it *in terms of* the input size
- “operations” may be faster or slower depending on the hardware

“Primitive” Operations

Things the computer can do in a “fixed” amount of time.

“fixed” - doesn’t depend on the input size (n)

A non-exhaustive list:

- Get or set the value of a variable or array location
- Evaluate a simple expression
- Return from a method

Strategies for counting primitive operations

Easiest case:

1. Identify all primitive operations

2. Identify how many time each one happens

3. Add them all up.

```
alg(A, n):
    sum = 0                 1
    for i = 1..n:               |
        sum += A[i]         1   |  n times
```

Strategies for counting primitive operations

Easiest case:

1. Identify all primitive operations
2. Identify the number of iterations each loop performs
3. Multiply primitives by how many times they're looped over
4. A

total: 1 + n

```
alg(A, n):
    sum = 0                 1
    for i = 1..n:
        sum += A[i]         1   | n times
```

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0];

  for (int i = 1; i < a.length; i++) {

    if (currentMax < a[i]) {

      currentMax = a[i];
    }
  }
  return currentMax;
}
```

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0]; get, set

            set     eval, get    eval, set
  for (int i = 1; i < a.length; i++) {

                  eval, get
    if (currentMax < a[i]) {

                set, get
      currentMax = a[i];
    }
  }    return
  return currentMax;
}
```

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0]; get, set

            set      eval, get   eval, set
  for (int i = 1; i < a.length; i++) {
                 eval, get
    if (currentMax < a[i]) {
                set, get
      currentMax = a[i];
    }
  }    return
  return currentMax;
}
```

Let N = `a.length`. How many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0]; get, set   1
                   1         2(N-1)
                  set        eval, get          eval, set 2(N-1)
  for (int i = 1; i < a.length; i++) {
                         eval, get 2(N-1)
    if (currentMax < a[i]) {
                        set, get
      currentMax = a[i];
    }  1
  }   return
  return currentMax;
}
```

Let N = `a.length`. How many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0]; get, set  1
             1        2(N-1)
            set      eval, get         eval, set 2(N-1)
  for (int i = 1; i < a.length; i++) {
                  eval, get 2(N-1)
    if (currentMax < a[i]) {
                 set, get ????
      currentMax = a[i];
    }  1
  }   return
  return currentMax;
}
```

Let N = `a.length`. How many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0]; get, set   1
                1          2(N-1)
              set        eval, get        eval, set 2(N-1)
  for (int i = 1; i < a.length; i++) {

                    eval, get 2(N-1)
    if (currentMax < a[i]) {

                  set, get ????
      currentMax = a[i];
    }  1
  }  return
  return currentMax;
}
```

Let N = `a.length`. AT MOST how many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0]; get, set   1
            1       2(N-1)
           set     eval, get        eval, set 2(N-1)
  for (int i = 1; i < a.length; i++) {
                 eval, get 2(N-1)
    if (currentMax < a[i]) {
                eval, get 2(N-1)
      currentMax = a[i];
    }  1
  }  return
  return currentMax;
}
```

Let N = `a.length`. AT MOST how many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
  int currentMax = a[0]; get, set  1
             1       2(N-1)
            set     eval, get           eval, set 2(N-1)
  for (int i = 1; i < a.length; i++) {
                  eval, get 2(N-1)
    if (currentMax < a[i]) {
                 set, get 2(N-1)
      currentMax = a[i];
    }  1
  }   return                      Total: 8N-5
  return currentMax;
}
```

sillyFindMax

```
public int sillyFindMax(int[] a) {
  for (int i = 0; i < a.length; i++) {
    // check if anything is bigger than a[i]
    boolean isMax = true;
    for (int j = 0; j < a.length; j++) {
      if (a[j] > a[i]) {
        isMax = false;  // found something bigger
      }
    }
    if (isMax) {
      return a[i];
    }
  }
}
```

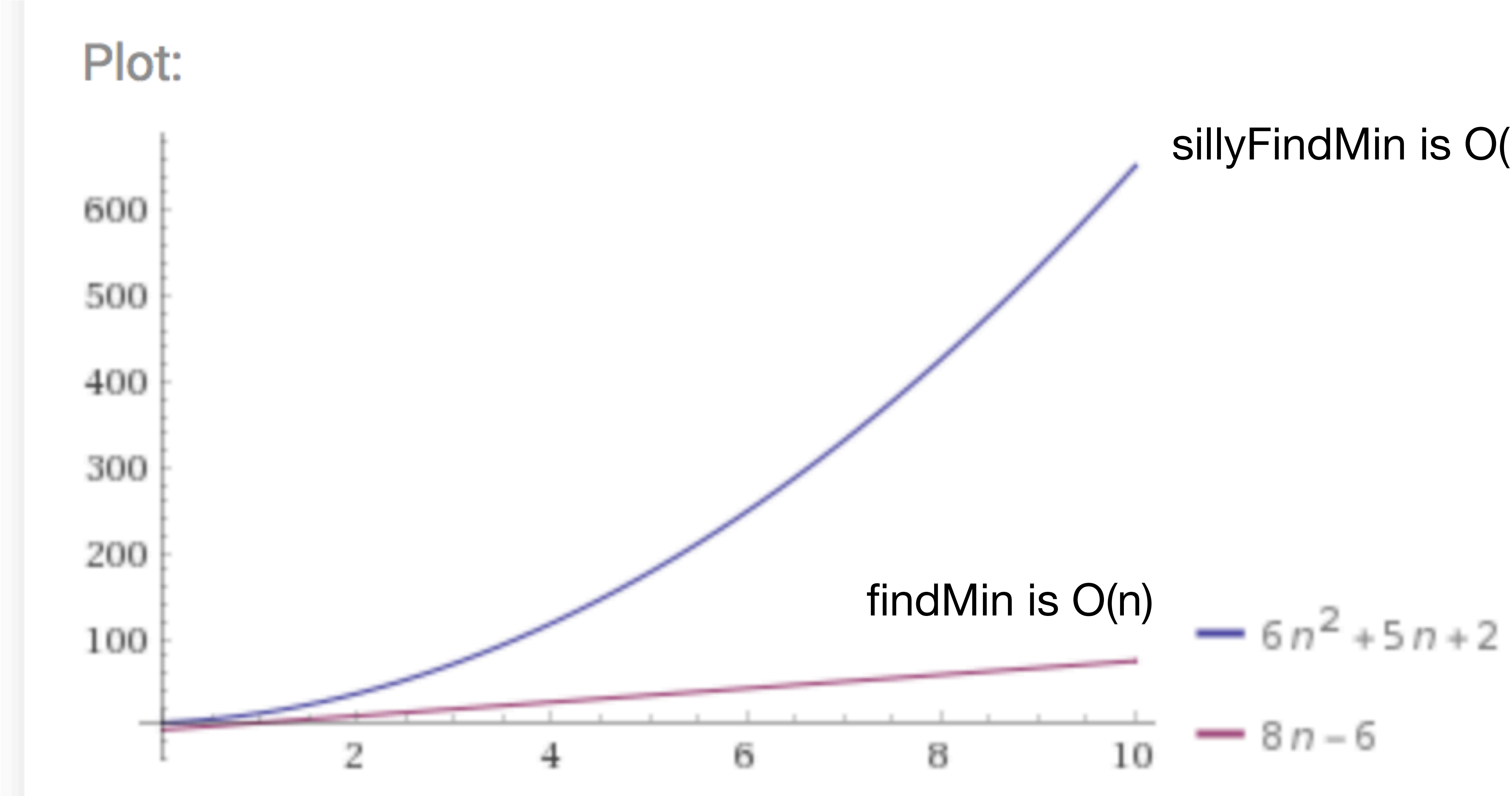
sillyFindMax

```
public int sillyFindMax(int[] a) {
   for (int i = 0; i < a.length; i++) {        1 + N + N
      // check if anything is bigger than a[i]
      boolean isMax = true;                             N
      for (int j = 0; j < a.length; j++) {     N (1+N+N)
         if (a[j] > a[i]) {                        N (3N)
            isMax = false;  // found something bigger   N*N
         }
      }
      if (isMax) {                                      N
         return a[i];                                   1
      }
   }
}
```

$2 + 5N + 6N^2$

Comparing findMaxes

- findMax: 8N - 5
- sillyFindMax: 2 + 5N + 6N^2



sillyFindMin is O(n^2)

findMin is O(n)

Strategies for counting primitive operations

Not as easy case:

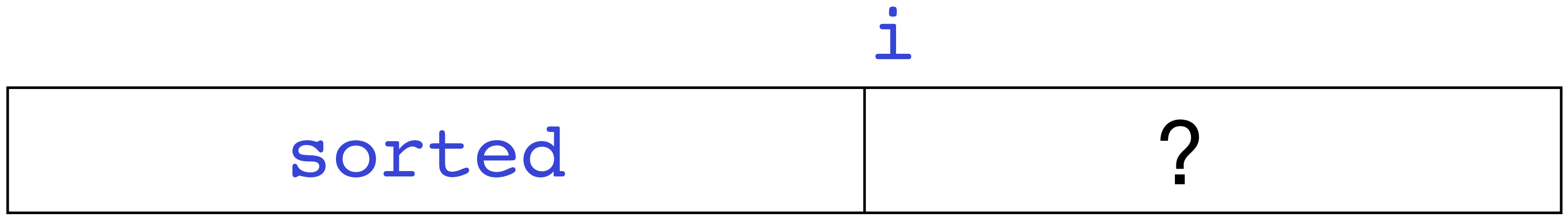
1. Identify all primitive operations

2. Trace through the algorithm, reasoning about the loop bounds in order to count the worst-case number of times each operation happens.

Insertion Sort: Runtime

```
// Sorts A using insertion sort
insertionSort(A):
  i = 0;
  while i < A.length:
    j = i;
    while j > 0 and A[j] < A[j-1]:
      swap(A[j], A[j-1])
      j--
    i++
```

Invariant: A



i

sorted

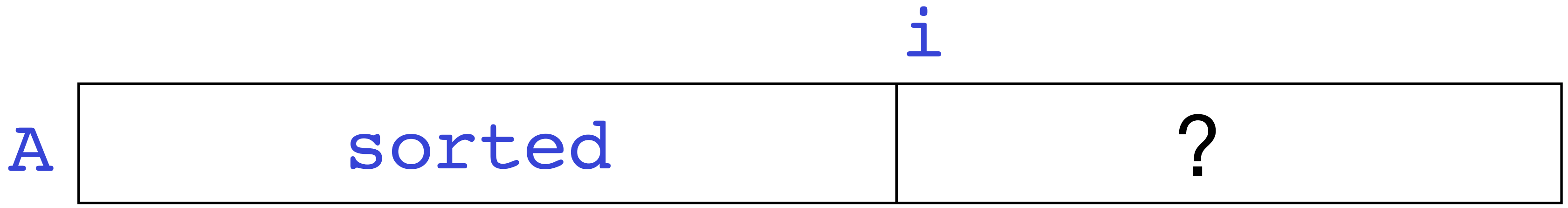
?

AT MOST How many times do we call swap() during iteration i?

Insertion Sort: Runtime

```
// Sorts A using insertion sort
insertionSort(A):
  i = 0;
  while i < A.length:
    j = i;
    while j > 0 and A[j] < A[j-1]:
      swap(A[j], A[j-1])
      j--
    i++
```

Invariant:



AT MOST How many times do we call swap() during iteration i?

j begins at i and could go as far as 1: that’s as many as i swaps at iteration i

Number of swaps: 1 in 1st iteration + 2 in 2nd iteration + … + n in nth iteration

1 + 2 + 3 + … + n-1 + n = (n * (n-1)) / 2 = (n^2 - n) / 2