

CSC 212: Data Structures and Abstractions

Basic Sorting Algorithms

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2020



Announcements

- Programming #1
 - ✓ 1 week left
- Programming #2
 - ✓ will be posted next week (maze generation)
 - ✓ focus on classes and dynamic arrays (`std::vector`)
- If programming is still a **significant** issue ...
 - ✓ consider taking this class next semester and focus on addressing the issue

Looking for internships / jobs?

<https://careers.google.com/how-we-hire/interview/>

https://www.amazon.jobs/en/landing_pages/software-development-topics

<https://www.facebook.com/careers/life/preparing-for-your-software-engineering-interview-at-facebook/>

Worst-case, Average-case,
Best-case

Warming up: Analyze this code

```
unsigned int argmin(const std::vector<int> &values) {  
    unsigned int length = values.size();  
    assert(length > 0);  
    unsigned int idx = 0;  
    int current = values[0];  
    for (unsigned int i = 1 ; i < length ; i ++) {  
        if (values[i] < current) {  
            current = values[i];  
            idx = i;  
        }  
    }  
    return idx;  
}
```

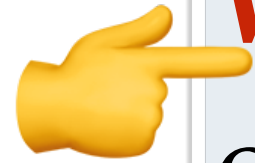
$T(n) = ?$
based on number of comparisons

Warming up: Analyze this code

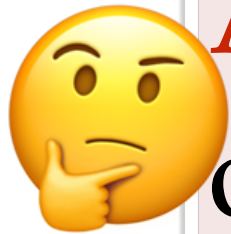
```
bool argk(const std::vector<int> &values, int k, unsigned int &idx) {  
    unsigned int length = values.size();  
    for (unsigned int i = 0 ; i < length ; i++) {  
        if (values[i] == k) {  
            idx = i;  
            return true;  
        }  
    }  
    return false;  
}
```

$T(n) = ?$
based on number of comparisons

Different types of analysis



Worst-case: maximum time of algorithm on any input



Average-case: expected time of algorithm over all inputs



Best-case: minimum time of algorithm on some (optimal) input

Different types of analysis

- While **asymptotic analysis** describes $T(n)$ as n approaches infinity ...
 - ✓ asymptotic notation: big O, big Omega, big Theta
- **Case analysis** looks into the different input types
 - ✓ best-case, worst-case, average-case

Both analysis types are orthogonal
to each other

Worst-case, Average-case, Best-case

- Ex: factorial of a number (iterative algorithm)
- Ex: sequential search (return first occurrence)
- Ex: sequential search (return last occurrence)

Basic Sorting Algorithms

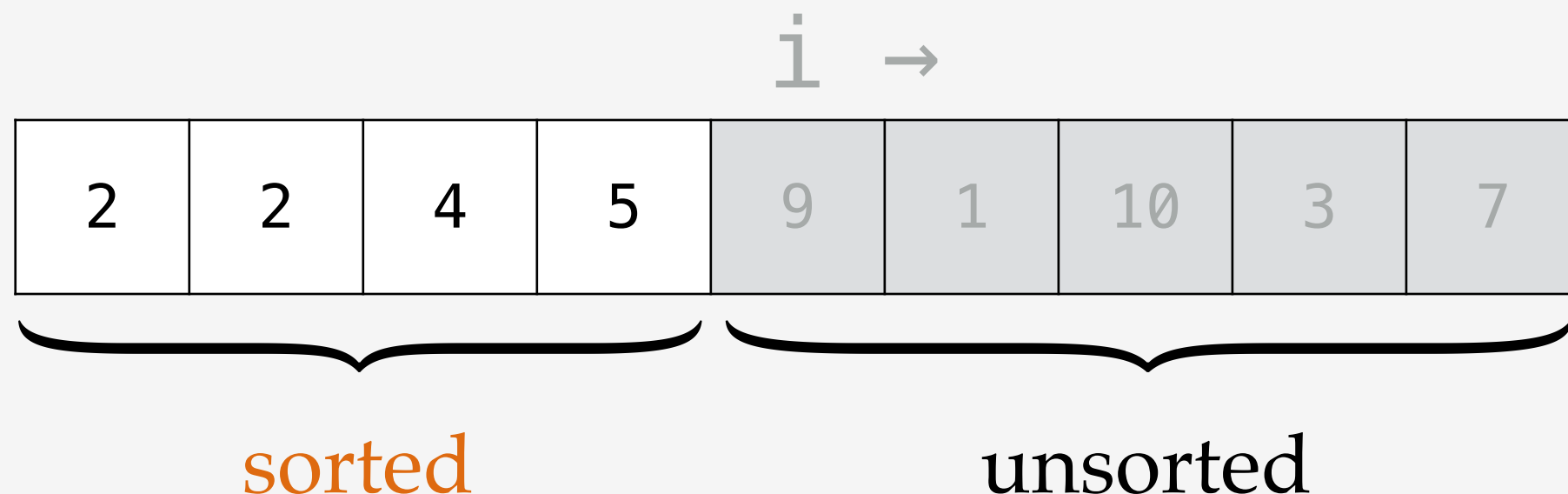
Sorting

- Given **n** elements that can be compared according to a **total order** relation
 - ✓ we want to rearrange them in non-increasing / non-decreasing order
 - ✓ for example (non-decreasing):
 - **input:** sequence of items $A = [k_0, k_1, \dots, k_{n-1}]$
 - **output:** permutation of A $B \mid B[0] \leq B[1] \leq \dots B[n-1]$

Central problem in computer science

Insertion Sort

- Array is divided into **sorted** and **unsorted** parts
 - ✓ algorithm scans array from **left to right**
- Invariants
 - ✓ elements in **sorted** are in ascending order
 - ✓ elements in **unsorted** have not been seen



Insertion Sort Demo

```

void insertionsort(int *A, unsigned int n) {
    int temp;
    unsigned int i, j;
    // grows the left part (sorted)
    for (i = 0 ; i < n ; i++) {
        // inserts A[j] in sorted part
        for (j = i ; j > 0 ; j--) {
            if (A[j] < A[j-1]) {
                temp = A[j];
                A[j] = A[j-1];
                A[j-1] = temp;
            }
            else
                break;
        }
    }
}

```

Number of comparisons? Number of exchanges?

Analysis — Insertion Sort (comparisons)

- Running time depends on the input
- Worst-case?
 - ✓ input reverse sorted
- Best-case?
 - ✓ input already sorted
- Average-case?
 - ✓ expect every element to move $O(n/2)$ times

Partially sorted arrays

- An **inversion** is a pair of keys that are out of order

1	3	4	5	2	6	10	15	7
---	---	---	---	---	---	----	----	---

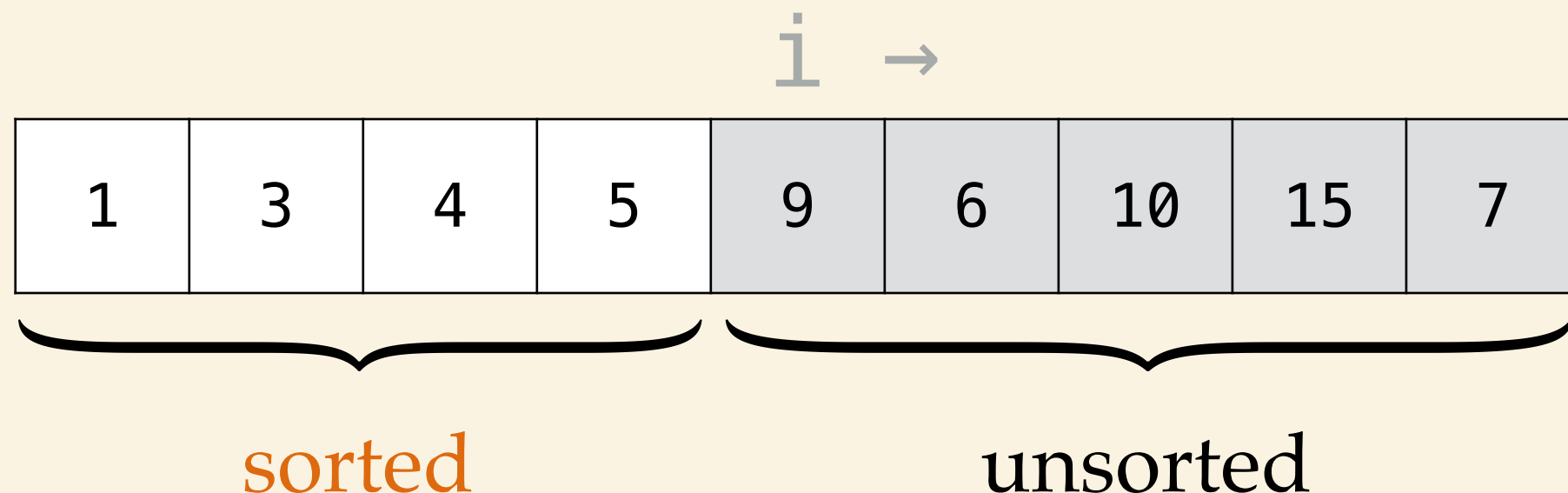
“array is **partially sorted** if the number of pairs that are out-of-order is $O(n)$ ”

For partially-sorted arrays, insertion sort runs in **linear time**.

$$\Theta(n)$$

Selection Sort

- Array is divided into **sorted** and **unsorted** parts
 - ✓ algorithm scans array from **left to right**
- Invariants
 - ✓ elements in **sorted** are **fixed** and in ascending order
 - ✓ no element in **unsorted** is smaller than any element in **sorted**



Selection Sort Demo

```

void selectionsort(int *A, unsigned int n) {
    int temp;
    unsigned int i, j, min;
    // grows the left part (sorted)
    for (i = 0 ; i < n ; i++) {
        min = i;
        // find min in unsorted part
        for (j = i+1 ; j < n ; j++) {
            if (A[j] < A[min]) {
                min = j;
            }
        }
        // swap A[i] and A[min]
        temp = A[i];
        A[i] = A[min];
        A[min] = temp;
    }
}

```

Number of comparisons? Number of exchanges?

Summary

	Best-Case	Average-Case	Worst-Case
Selection Sort	$\theta(n^2)$	$\theta(n^2)$	$\theta(n^2)$
Insertion Sort	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$