Chapter 1
Analysis and Proof Motivation
Why analysis is important...

Algorithm Design and Analysis (Fall 2021)

Christian A. Duncan
School of Engineering
Quinnipiac University

Objectives

1. Discuss complexity around a simple algorithm
2. Explain difference between best-case, worst-case, and average-case analysis
3. Explain importance of a proof

Determining the Max

- Sometimes problems look hard but are easy.
- Sometimes they look easy but are hard.

Determining the Max

- Sometimes problems look hard but are easy.
- Sometimes they look easy but are hard.
- Take for instance: `findMax(double[] A)`
- How do we compute the max value in an array of n floats?

Anyone?

Determining the Max

- Sometimes problems look hard but are easy.
- Sometimes they look easy but are hard.
- Take for instance: `findMax(double[] A)`
- How do we compute the max value in an array of *n* floats?
- An incremental alg: if A[i] > max then update max

Example

A	4	2	8	6	9	12	11	1
max	$\underline{4}$	4	$\underline{8}$	8	$\underline{9}$	$\underline{12}$	12	12

Determining the Max

- Sometimes problems look hard but are easy.
- Sometimes they look easy but are hard.
- Take for instance: `findMax(double[] A)`
- How do we compute the max value in an array of *n* floats?
- An incremental alg: if A[i] > max then update max

Breakout Time:

- *Group size: about 4*
- *Time: 5-10 minutes*
- *Ponder: How many times does the maximum value change for an array with n values? (A record high)*
- *Start with* $n = 100, 1000, 1$ *million.*
- *In the best case (fewest number of changes).*
- *In the worst case (most number of changes).*
- *In an average case (consider for 1 million values).*

Determining the Max

- Sometimes problems look hard but are easy.
- Sometimes they look easy but are hard.
- Take for instance: `findMax(double[] A)`
- How do we compute the max value in an array of n floats?
- An incremental alg: if A[i] > max then update max

Share your thoughts

- How many times does the maximum value change for 100 items on the worst-case?
- How many times does the maximum value change for 100 items in the best-case?
- What is it for $n = 1000$? What about in terms of n?
- On *average* how many times does it change if there are a million items?

Determining the Max

- Sometimes problems look hard but are easy.
- Sometimes they look easy but are hard.
- Take for instance: `findMax(double[] A)`
- How do we compute the max value in an array of *n* floats?
- An incremental alg: if A[i] > max then update max

Solution:

- *Best case: 1. Example: n 1 2 3 4 ... $n-1$*
- *Worst case: n. Example: 1 2 3 4 ... n*
- *Average case? Hmm... how would we figure this out?*

Analyzing Record Highs

- Three ways I can think of to do this.

Analyzing Record Highs

- Three ways I can think of to do this.
  1. Guess. (Intuition is not always reliable.)

Analyzing Record Highs

- Three ways I can think of to do this.
  1. Guess. (Intuition is not always reliable.)
  2. Code it up and test it empirically.
     - Can give *some* insight.
     - See `MaxTracker.java` code.

Analyzing Record Highs

- Three ways I can think of to do this.
  1. Guess. (Intuition is not always reliable.)
  2. Code it up and test it empirically.
     - Can give *some* insight.
     - See `MaxTracker.java` code.
  3. Analyze it mathematically.
     - Not always easy...
     - Has own set of possibilities and problems.
     - Need to develop tools to speak this language.

Analyzing Record Highs

- Three ways I can think of to do this.
  1. Guess. (Intuition is not always reliable.)
  2. Code it up and test it empirically.
     - Can give *some* insight.
     - See `MaxTracker.java` code.
  3. Analyze it mathematically.
     - Not always easy...
     - Has own set of possibilities and problems.
     - Need to develop tools to speak this language.
- Let us do it empirically.

MaxTracker.java **Lecture 01**

```java
/************
 * Christian Duncan
 *
 * MaxTracker: Designed for CSC215: Algorithms Design and Analysis
 *
 * This program runs a very rudimentary experiment to determine how often the maximum value changes
 * in a straight-forward scan to find the largest element in an (unsorted) array.
 * For each array size, it reports both the average number of swaps, the
 * best-case number of swaps, and the worst-case number of swaps.
 *   Printing them out in a CSV format (for analysis on a spreadsheet program).
 ************/
import java.util.Random;

public class MaxTracker {
    static Random ran;
    public static final int MIN_SIZE = 10;
    public static final int MAX_SIZE = 2000000;
    public static final int NUM_CASES = 1000;

    /****
     * trackMax:
     *   array: The input array to search
     *   return: The NUMBER of times the max changed.
     *   Given an array of values, computes the maximum value in that array.
     *   But returns the number of times the maximum changes (not the max). - For experimental reasons
     ****/
    public static int trackMax(double[] array) {
        if (array.length < 1)
            return 0;  // Nothing to do

        int count = 1;  // Count that first assignment as one change.
        double max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];  // New maximum value
                count++;         // Increase the count
            }
        }
        return count;   // Note: Doesn't return MAX value, just number of changes
    }

    /****
     * testTracker:
     *   size: Size of array to be testing
     *   testCases: Number of test cases to perform
     *   Prints out best case, worst case, average case for given array size
     *     (Generating a different array for each case of course)
     ****/
    public static void testTracker(int size, int testCases) {
        long totalChanges = 0;
        long minChange = size+1;  // Just more than the maximum every possible
        long maxChange = -1;      //  Less than minimum possible
        double[] array = new double[size];

        for (int i = 0; i < testCases; i++) {
            for (int j = 0; j < size; j++) array[j] = ran.nextDouble();   // Generate a new test case
            int changes = trackMax(array);   // Compute the number of changes for this array

            // Determine if it is a best-case or worst-case situation and tally it
            totalChanges += changes;
            if (changes < minChange) {
                minChange = changes;
            } else if (changes > maxChange) {
                maxChange = changes;
            }
        }
        System.out.println(size + ", " + minChange + ", " + maxChange + ", " +
                           ((double) totalChanges / (double) testCases));
    }

    public static void main(String[] args) {
        ran = new Random(); // Create random number generator
        for (int size = MIN_SIZE; size <= MAX_SIZE; size *= 2)
            testTracker(size, NUM_CASES);
    }
```

Majority Vote

Problem Statement

- Given: An array of numbers.
- Know: there is one number that is in the majority (more than half)
- Determine that number
- Catch: Only use constant space (const. var.)

Majority Vote

Problem Statement

- Given: An array of numbers.
- Know: there is one number that is in the majority (more than half)
- Determine that number
- Catch: Only use constant space (const. var.)

Breakout Time:

- *Group size: about 4*
- *Time: 5 minutes*
- *Ponder*
    - *What approach works if memory was not an issue?*
    - *If we must use constant memory, is it even possible?*
    - *How or how would you prove it isn't possible?*

Majority Vote

Problem Statement

- Given: An array of numbers.
- Know: there is one number that is in the majority (more than half)
- Determine that number
- Catch: Only use constant space (const. var.)

Solution:

```
majority(A):
  tally = 0, max = -1 # Does not matter yet
  for a in A:
    if tally = 0: tally = 1, max = a
    else if max = a: tally = tally + 1
    else: tally = tally - 1
  return max
```

Majority Vote

Hold on

Majority Vote

- Does this really work?

Majority Vote

- Does this really work?
- Yes, trust me.

Majority Vote

- Does this really work?
- Yes, trust me.
- How do we know?

Majority Vote

- Does this really work?
- Yes, trust me.
- How do we know?
- Trust me.

Majority Vote

- Does this really work?
- Yes, trust me.
- How do we know?
- Trust me.
- Do we code it up and try it out a few times?
- This is where proofs come in!

Majority Vote (Proof)

Majority Vote Algorithm

```
majority(A):
  tally = 0, max = -1 # Does not matter yet
  for a in A:
    if tally = 0: tally = 1, max = a
    else if max = a: tally = tally + 1
    else: tally = tally - 1
  return max
```

- Let m be the majority element.
- At any step, let t represent either:
  - `tally` - if max stores m
  - `-tally` - otherwise
- Now what happens if $a = m$?

Majority Vote (Proof)

Majority Vote Algorithm

```
majority(A):
  tally = 0, max = -1 # Does not matter yet
  for a in A:
    if tally = 0: tally = 1, max = a
    else if max = a: tally = tally + 1
    else: tally = tally - 1
  return max
```

- Let m be the majority element.
- At any step, let t represent either:
  - `tally` - if max stores m
  - `-tally` - otherwise
- Now what happens if $a = m$?
  - If max$=m$ then t increases by 1.
  - Otherwise, t increases by 1!
    Because tally decreased by 1.

Majority Vote (Proof)

Majority Vote Algorithm

```
majority(A):
  tally = 0, max = -1 # Does not matter yet
  for a in A:
    if tally = 0: tally = 1, max = a
    else if max = a: tally = tally + 1
    else: tally = tally - 1
  return max
```

- Let m be the majority element.
- At any step, let t represent either:
  - `tally` - if max stores m
  - `-tally` - otherwise
- Now what happens if $a = m$? t increases by 1.

Majority Vote (Proof)

Majority Vote Algorithm

```
majority(A):
  tally = 0, max = -1 # Does not matter yet
  for a in A:
    if tally = 0: tally = 1, max = a
    else if max = a: tally = tally + 1
    else: tally = tally - 1
  return max
```

- Let m be the majority element.
- At any step, let t represent either:
  - `tally` - if max stores m
  - `-tally` - otherwise
- Now what happens if $a = m$? t increases by 1.
- If $a \neq m$ then t increases or decreases by 1.

Majority Vote (Proof)

Majority Vote Algorithm

```
majority(A):
  tally = 0, max = -1 # Does not matter yet
  for a in A:
    if tally = 0: tally = 1, max = a
    else if max = a: tally = tally + 1
    else: tally = tally - 1
  return max
```

- Let m be the majority element.
- At any step, let t represent either:
  - `tally` - if max stores m
  - `-tally` - otherwise
- Now what happens if $a = m$? t increases by 1.
- If $a \neq m$ then t increases or decreases by 1.
- However, since m is the majority, there would be more increases than decreases.
- Therefore, at the end, t is positive and hence max stores m.