Testing and Debugging

Data Structures and Algorithms

Warm Up

From Last Lecture:

- What are the expected operations for a "map" ADT?
- How would you implement a Map…
  - To optimize for insertion?
  - To optimize for retrieval?
- What is an "Iterator" and what are its two key operations?

Socrative:

www.socrative.com

Room Name: CSE373

Please enter your name as: Last, First

Administrivia

Fill out class survey – posted today

Find a partner by Thursday!

Testing

Computers don't make mistakes- people do!

"I'm almost done, I just need to make sure it works"
– Naive 14Xers

Software Test: a separate piece of code that exercises the code you are assessing by providing input to your code and finishes with an assertion of what the result should be.

1. Isolate

break your code into small modules

2. Build in increments

Make a plan from simplest to most complex cases

3. Test as you go

As your code grows, so should your tests

Types of Tests

Black Box

- Behavior only – ADT requirements
- From an outside point of view
- Does your code uphold its contracts with its users?
- Performance/efficiency

White Box

- Includes an understanding of the implementation
- Written by the author as they develop their code
- Break apart requirements into smaller steps
- "unit tests" break implementation into single assertions

What to test?

Expected behavior

- The main use case scenario
- Does your code do what it should given friendly conditions?

Forbidden Input

- What are all the ways the user can mess up?

Empty/Null

- Protect yourself!
- How do things get started?

Boundary/Edge Cases

- First
- last

Scale

- Is there a difference between 10, 100, 1000, 10000 items?

Tips for testing

You cannot test every possible input, parameter value, etc.

- Think of a limited set of tests likely to expose bugs.

Think about boundary cases

- Positive; zero; negative numbers
- Right at the edge of an array or collection's size

Think about empty cases and error cases

- 0, -1, null; an empty list or array

test behavior in combination

- Maybe `add` usually works, but fails after you call `remove`
- Make multiple calls; maybe `size` fails the second time only

Thought Experiment

Discuss with your neighbors: Imagine you are writing an implementation of the List interface that stores integers in an Array. What are some ways you can assess your program's correctness in the following cases:

Expected Behavior

- Create a new list
- Add some amount of items to it
- Remove a couple of them

Forbidden Input

- Add a negative number
- Add duplicates
- Add extra large numbers

Empty/Null

- Call remove on an empty list
- Add to a null list
- Call size on an null list

Boundary/Edge Cases

- Add 1 item to an empty list
- Set an item at the front of the list
- Set an item at the back of the list

Scale

- Add 1000 items to the list
- Remove 100 items in a row
- Set the value of the same item 50 times

JUnit

JUnit: a testing framework that works with IDEs to give you a special GUI experience when testing your code

```
@Test
public void myTest() {
    Map<String, Integer> basicMap = new LinkedListDict<String, Integer>();
    basicMap.put("Kasey", 42);
    assertEquals(42, basicMap.get("Kasey"));
}
```

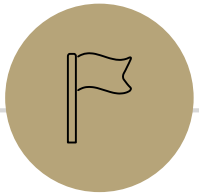
Assertions:

- assertEquals(item1, item2)
- assertTrue(Boolean expression)
- assertFalse(bollean expression)
- assertNotNull(item)

More: https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html

Write Tests for our Dictionary

Debugger



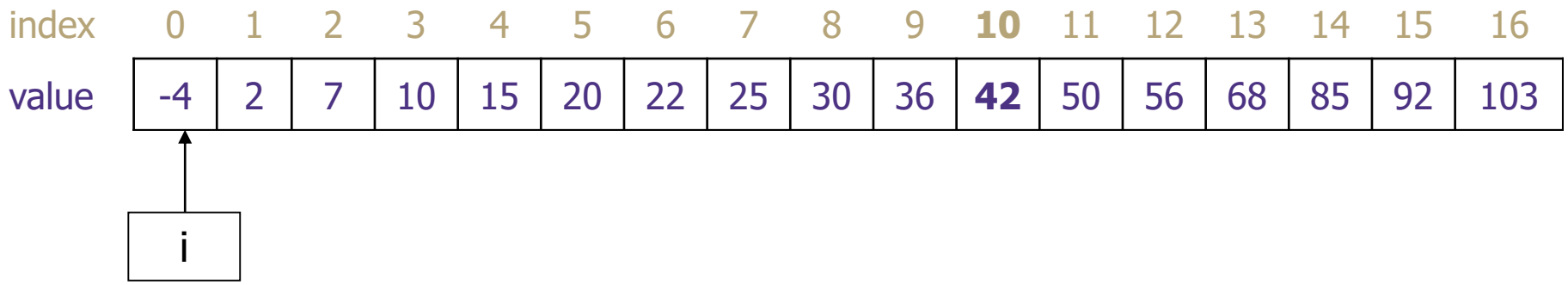
Algorithm Analysis

Review: Sequential Search

sequential search: Locates a target value in an array / list by examining each element from start to finish.

- How many elements will it need to examine?

- Example: Searching the array below for the value **42**:

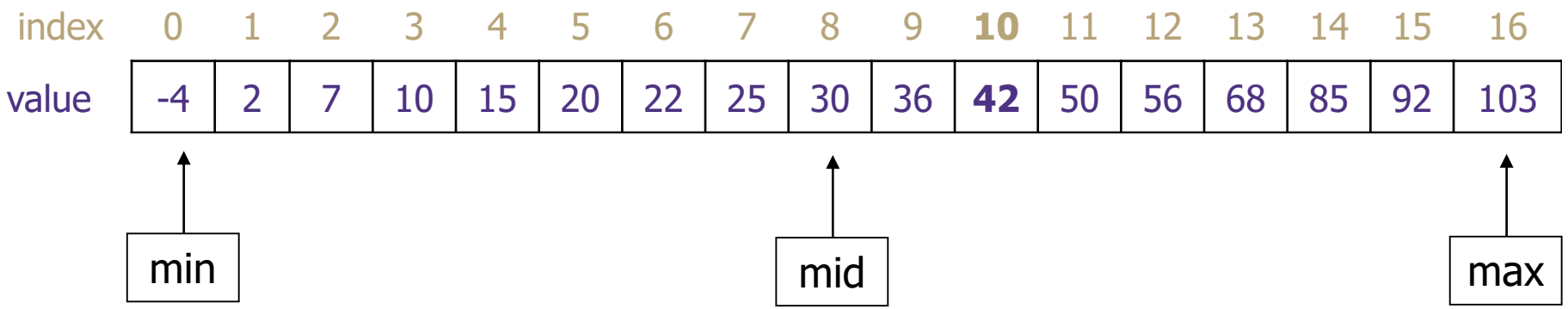


- What is the best case?
- What is the worst case?
- What is the complexity class?

Review: Binary Search

binary search: Locates a target value in a *sorted* array or list by successively eliminating half of the array from consideration.

- How many elements will it need to examine?
- Example: Searching the array below for the value **42**:



- What is the best case?
- What is the worst case?
- What is the complexity class?

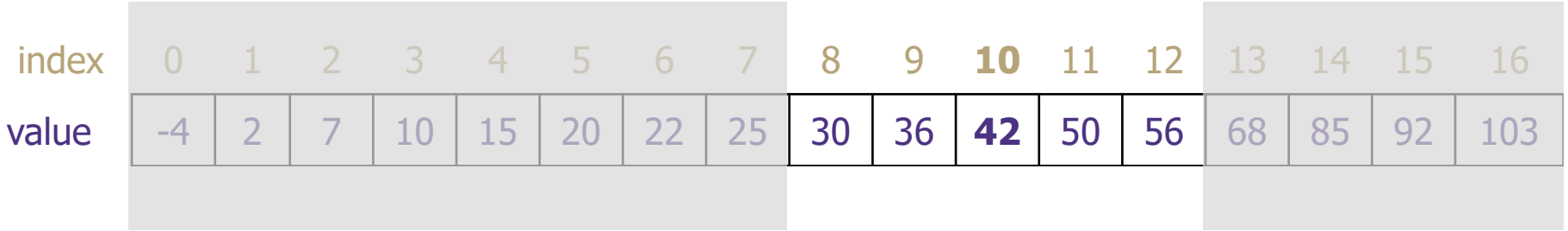
Analyzing Binary Search

What is the pattern?

- At each iteration, we eliminate half of the remaining elements

How long does it take to finish?

- 1st iteration – N/2 elements remain
- 2nd iteration – N/4 elements remain
- Kth iteration - N/2^k elements remain
- Done when N/2^k = 1



Analyzing Binary Search

Finishes when $N / 2^K = 1$

$N / 2^K = 1$

-> multiply right side by 2^K

$N = 2^K$

-> isolate K exponent with logarithm

$Log_2 N = k$

Is this exact?

- N can be things other than powers of 2
- If N is odd we can't technically use Log_2
- When we have an odd number of elements we select the larger half
- Within a fair rounding error

Asymptotic Analysis

asymptotic analysis: how the runtime of an algorithm grows as the data set grows

- bigO is the upper bound of an algorithm's asymptotic runtime

Approximations

- Basic operations take "constant" time
  - Assigning a variable
  - Accessing a field or array index
- Consecutive statements
  - Some of time for each statement
- Function calls
  - Time of function's body
- Conditionals
  - Time of condition + maximum time of branch code
- Loops
  - Number of iterations x time for loop body