



Week 11: Search Algorithms

CS 220: Software Design II — D. Mathias

Searching

Common activity when working with data

W Search algorithm - Wikipedia

Secure | https://en.wikipedia.org/wiki/Search_algorithm

Search

10/64

g in

Article

Talk

Read

Edit

View history

Search Wikipedia

WIKIPEDIA

The Free Encyclopedia

Main page

Contents

Featured content

Current events

Random article

Donate to Wikipedia

Wikipedia store

Interaction

Help

About Wikipedia

Community portal

Recent changes

Contact page

Tools

What links here

Related changes

Upload file

Special pages

Permanent link

Page information

Wikidata item

Cite this page

Print/export

Create a book

Download as PDF

Printable version

Search algorithm

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help [improve it](#) or discuss these issues on the [talk page](#). [\(Learn\)](#) [\[hide\]](#)

!

- This article **needs attention from an expert in computer science**. The specific problem is: **longstanding subpar state of article structure and content, which is currently list based, dated, and non-encyclopedic, and without external sourcing.** *(December 2014)*
- This article **focuses too much on specific examples** without [explaining their importance](#) to its main subject. *(December 2014)*
- This article **needs additional citations for verification.** *(April 2016)*

In computer science, a **search algorithm** is any **algorithm** which solves the **search problem**, namely, to retrieve information stored within some data structure, or calculated in the **search space** of a **problem domain**. Examples of such structures include but are not limited to a **linked list**, an **array data structure**, or a **search tree**. The appropriate **search** algorithm often depends on the data structure being **searched**, and may also include prior knowledge about the data. **Searching** also encompasses algorithms that query the data structure, such as the SQL SELECT command.^{[1][2]}

Search algorithms can be classified based on their mechanism of **searching**. **Linear search** algorithms check every record for the one associated with a target key in a linear fashion.^{[3][4]} **Binary, or half interval searches**, repeatedly target the center of the **search** structure and divide the **search** space in half. Comparison **search** algorithms improve on linear **searching** by successively eliminating records based on comparisons of the keys until the target record is found, and can be applied on data structures with a defined order.^[4] Digital **search** algorithms work based on the properties of digits in data structures that use numerical keys.^[5] Finally, **hashing** directly maps keys to records based on a **hash function**.^[6] **Searches** outside a linear **search** require that the data be sorted in some way.

Search functions are also evaluated on the basis of their complexity, or maximum theoretical run time. Binary **search** functions, for example, have a maximum complexity of $O(\log n)$, or logarithmic time. This means that the maximum number of operations needed to find the **search** target is a logarithmic function of the size of the **search** space.

keys

hash function

buckets

John Smith

Lisa Smith

Sandra Dee

00

01

02

03

:

13

14

15

521-8976

521-1234

:

521-9655

Visual representation of a hash table, a data structure that allows for fast retrieval of information.

Directory – Directory | UW-La

Secure | <https://www.uwlax.edu/info/directory/>

Admissions

Academics

Murphy Library

Arts

Athletics, Rec

Diversity & Inclusion

Campus Life

A-Z Index

Quicklinks

Search UWL

UNIVERSITY of WISCONSIN

LA CROSSE

DIRECTORY

A-Z Index

Emeritus & Retiree Directory

Other UW Directories

Search

Sauppe

NAME	EMAIL	DEPARTMENT	OFFICE	TITLE	PHONE
Sauppe, Allison V.	asauppe@uwlax.edu	Computer Science	214 Wing Technology Center	Assistant Professor	608.785.6815
Sauppe, Jason J.	jsauppe@uwlax.edu	Computer Science	207 Wing Technology Center	Assistant Professor	608.785.6807

Linear Search

```
int toFind = 86;
for (int i = 0; i < grades.length; i++) {
    if (grades[i] == toFind) {
        System.out.print(i + ", ");
    }
}
```

A *linear search* looks at each element in a data structure, in order, until encountering the desired element(s)

grades (int[])														
100	68	49	77	95	82	99	83	86	87	70	86	64	40	86
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Linear Search

```
int toFind = 86;
for (int i = 0; i < grades.length; i++) {
    if (grades[i] == toFind) {
        System.out.print(i);
        break;
    }
}
```

What if we can guarantee that there is at most one element that will match?

i.e., the element will appear either 0 or 1 times

grades (int[])														
100	68	49	77	95	82	99	83	65	87	70	86	64	40	88
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Linear Search

```
int toFind = 86;  
for (int i = 0; i < grades.length; i++) {  
  
    // to write  
  
}
```

What if there can be multiple instances of the number, and the list is sorted?

What should the loop code be to terminate as quickly as possible?

grades (int[])														
40	49	64	65	68	70	77	82	86	86	86	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Linear Search

```
int toFind = 86;  
for (int i = 0; i < grades.length; i++) {  
    if (grades[i] == toFind) {  
        System.out.print(i);  
    } else if (grades[i] > toFind) {  
        break;  
    }  
}
```

What if there can be multiple instances of the number, and the list is sorted?

What should the loop code be to terminate as quickly as possible?

grades (int[])														
40	49	64	65	68	70	77	82	86	86	86	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Exercise: Runtime Analysis

Fill in the following chart with runtimes using linear search

	unsorted list	sorted list (no repeats)
<i>search for the smallest value</i>		
<i>search for the largest value</i>		
<i>search for the median value</i>		
<i>search for a value that doesn't exist</i>		
<i>search for some random value that does exist (worst case)</i>		

e.g., we're looking for the number 100, but don't realize it's the largest number in the array

grades (int[])														
40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Exercise: Runtime Analysis

Fill in the following chart with runtimes using linear search

	unsorted list	sorted list (no repeats)
<i>search for the smallest value</i>	$O(n)$	$O(1)$
<i>search for the largest value</i>	$O(n)$	$O(1)$
<i>search for the median value</i>	$O(?)$	$O(1)$
<i>search for a value that doesn't exist</i>	$O(n)$	$O(n)$
<i>search for some random value that does exist (worst case)</i>	$O(n)$	$O(n)$

grades (int[])														
40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Exercise: Runtime Analysis

Fill in the following chart with runtimes using linear search

	unsorted list	sorted list (no repeats)
<i>search for the smallest value</i>	$O(n)$	$O(1)$
<i>search for the largest value</i>	$O(n)$	$O(1)$
<i>search for the median value</i>	$O(?)$	$O(1)$
<i>search for a value that doesn't exist</i>	$O(n)$	$O(n)$
<i>search for some random value that does exist (worst case)</i>	$O(n)$	$O(n)$

relies on absolute positions in the array

relies on relative positions in the array

grades (int[])														
40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Search Strategies

- Linear Search is not the only way to find a value in an array
- Strategies:
 - Linear search
 - Random guessing
 - Any others?

Let's play a game

- I'm going to think of a number in $[1, 1000]$.
- You have two goals:
 - Determine the number
 - Use the smallest possible number of guesses

Binary Search

A binary search uses a divide and conquer approach to subdivide a sorted list to find a number

divide and conquer approaches take a problem and break it down into smaller problems

Basic premise:

array to search in is the whole array; start at midpoint

is the number to find higher than the midpoint? search above; lower? search below

redefine the array to search in as either the above or below half; repeat

Binary Search

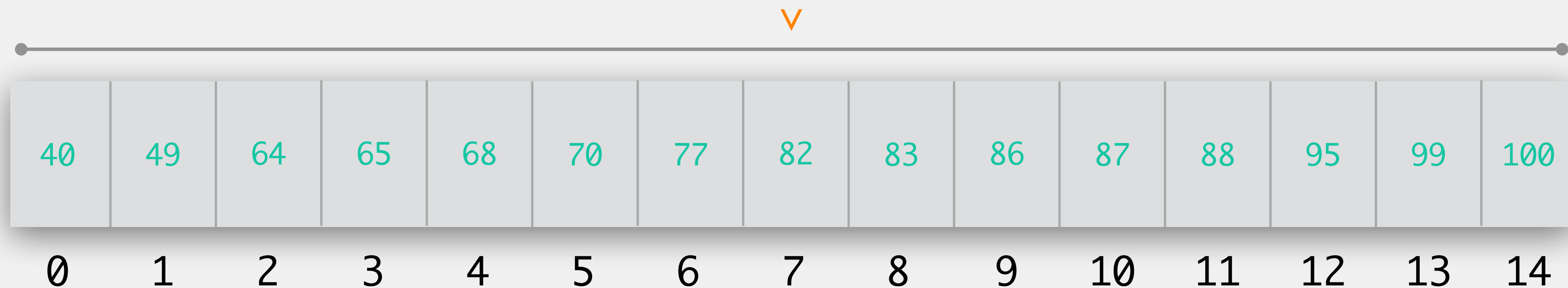
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

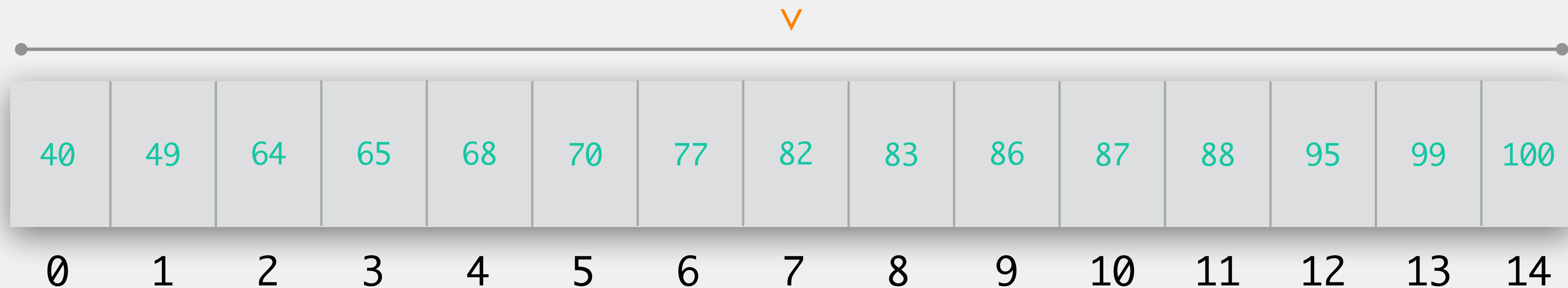
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

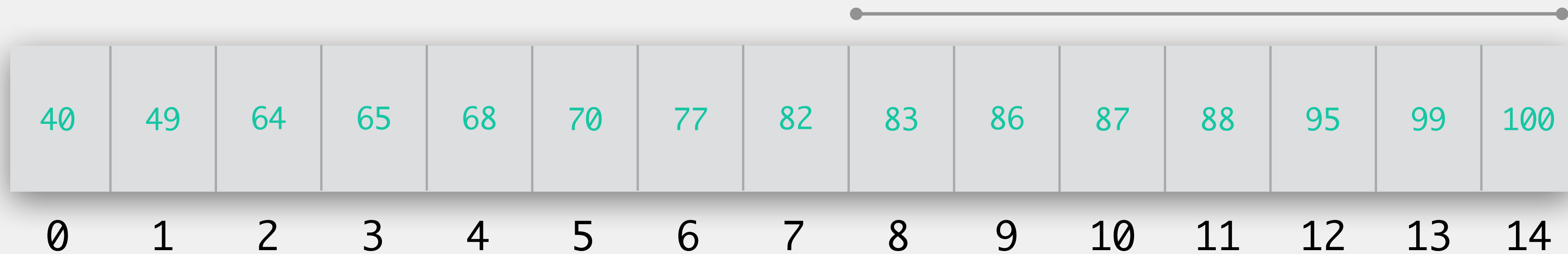
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

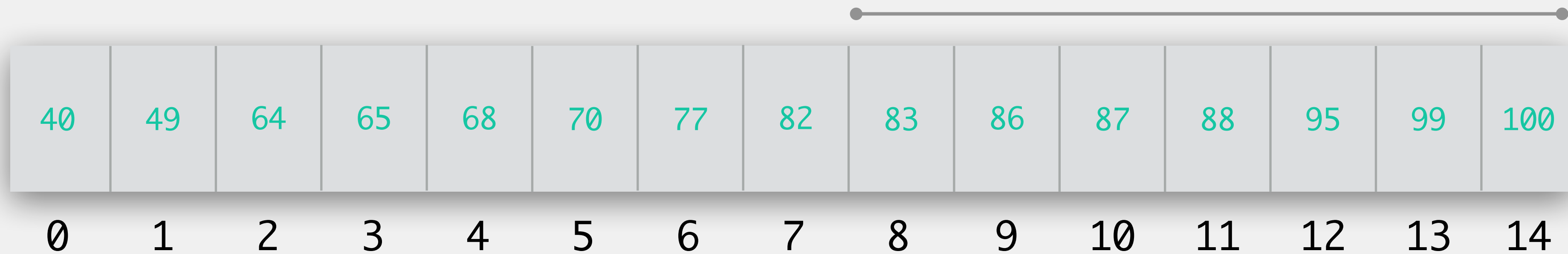
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

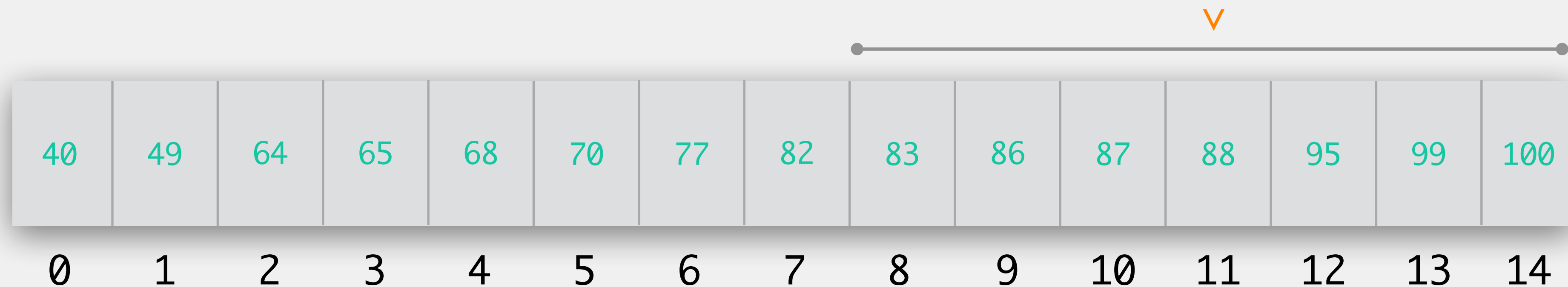
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

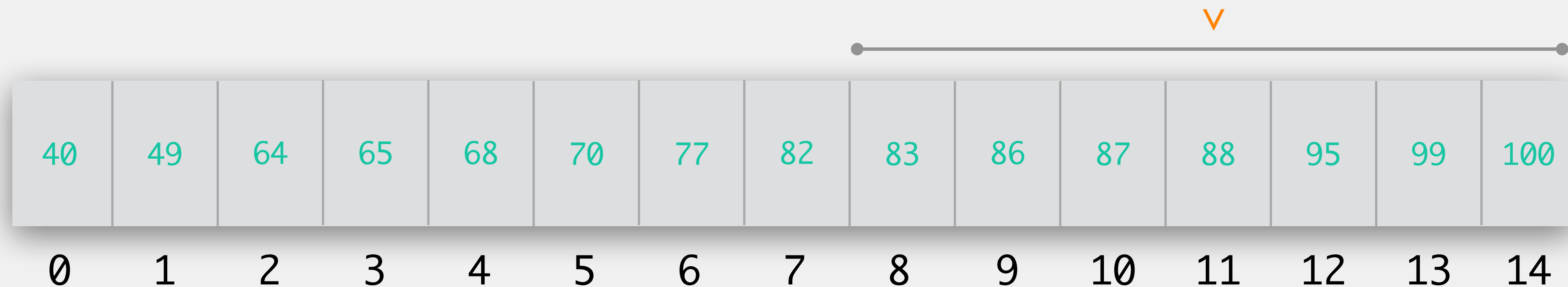
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

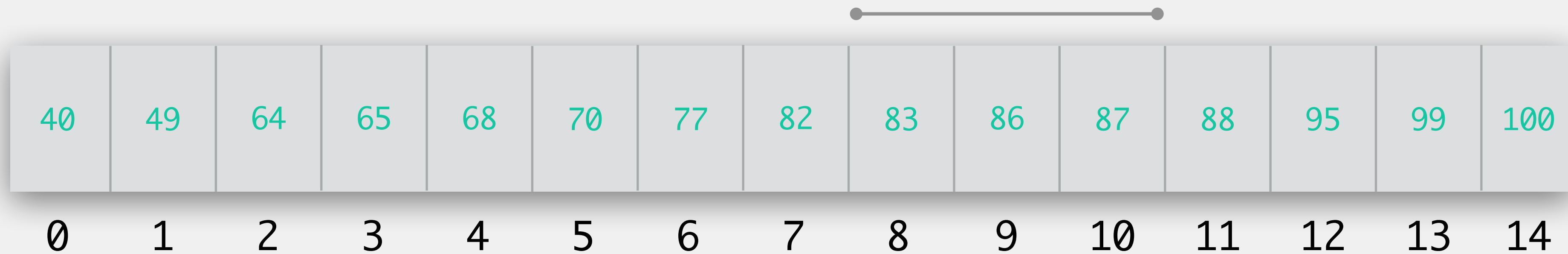
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

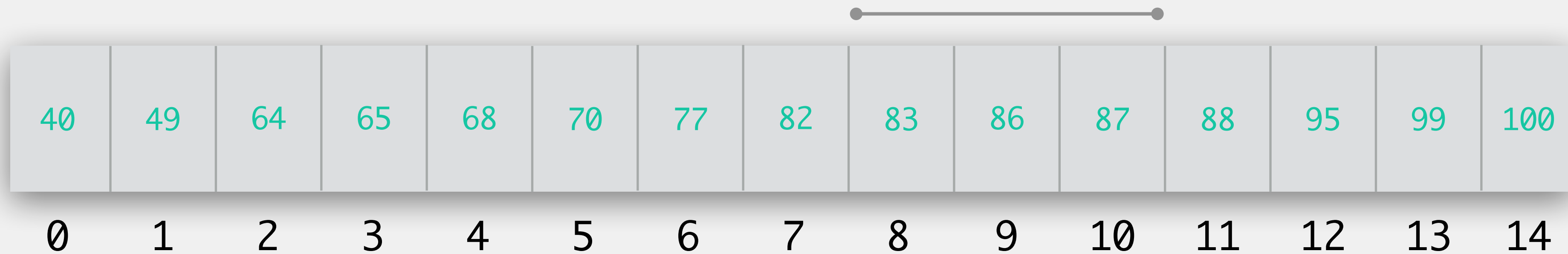
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

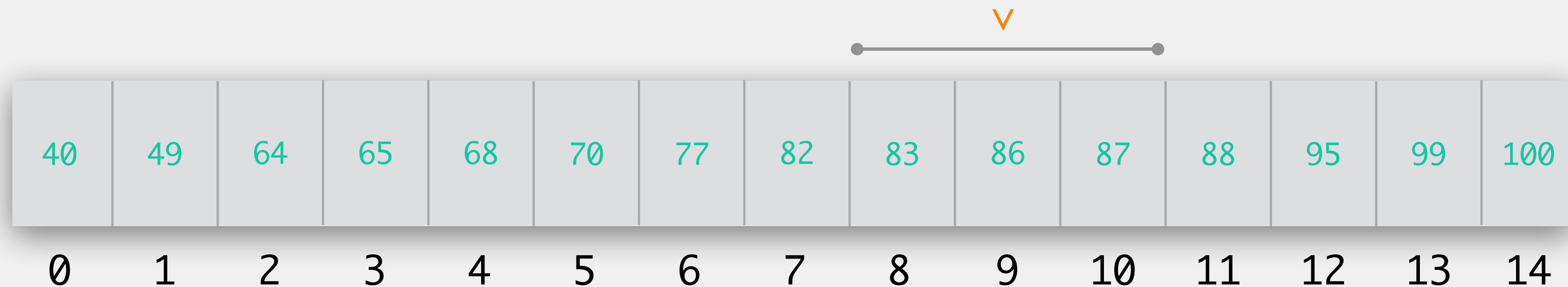
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Binary Search

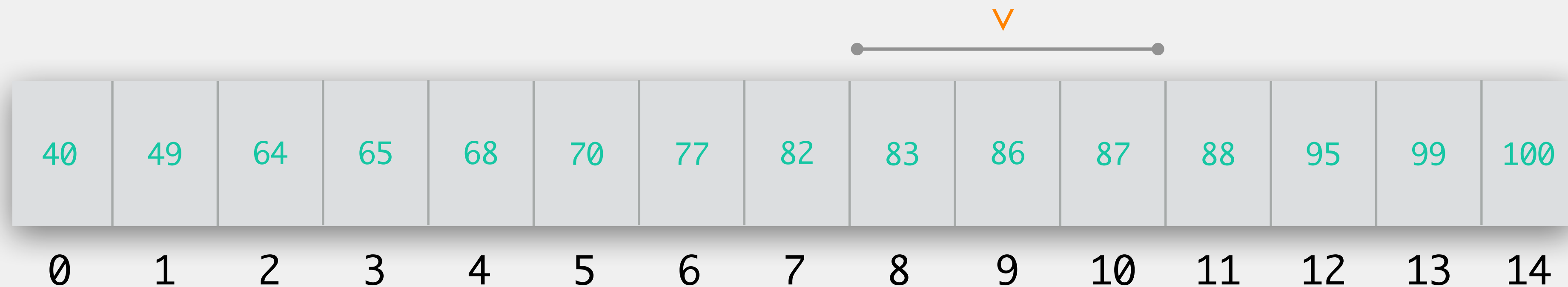
Start at midpoint in range to search in

Decide if number to find is higher or lower than midpoint

Redefine the range to either upper or lower half of array

Repeat

toFind = 86



Exercise: Runtime Analysis

Considering binary search...

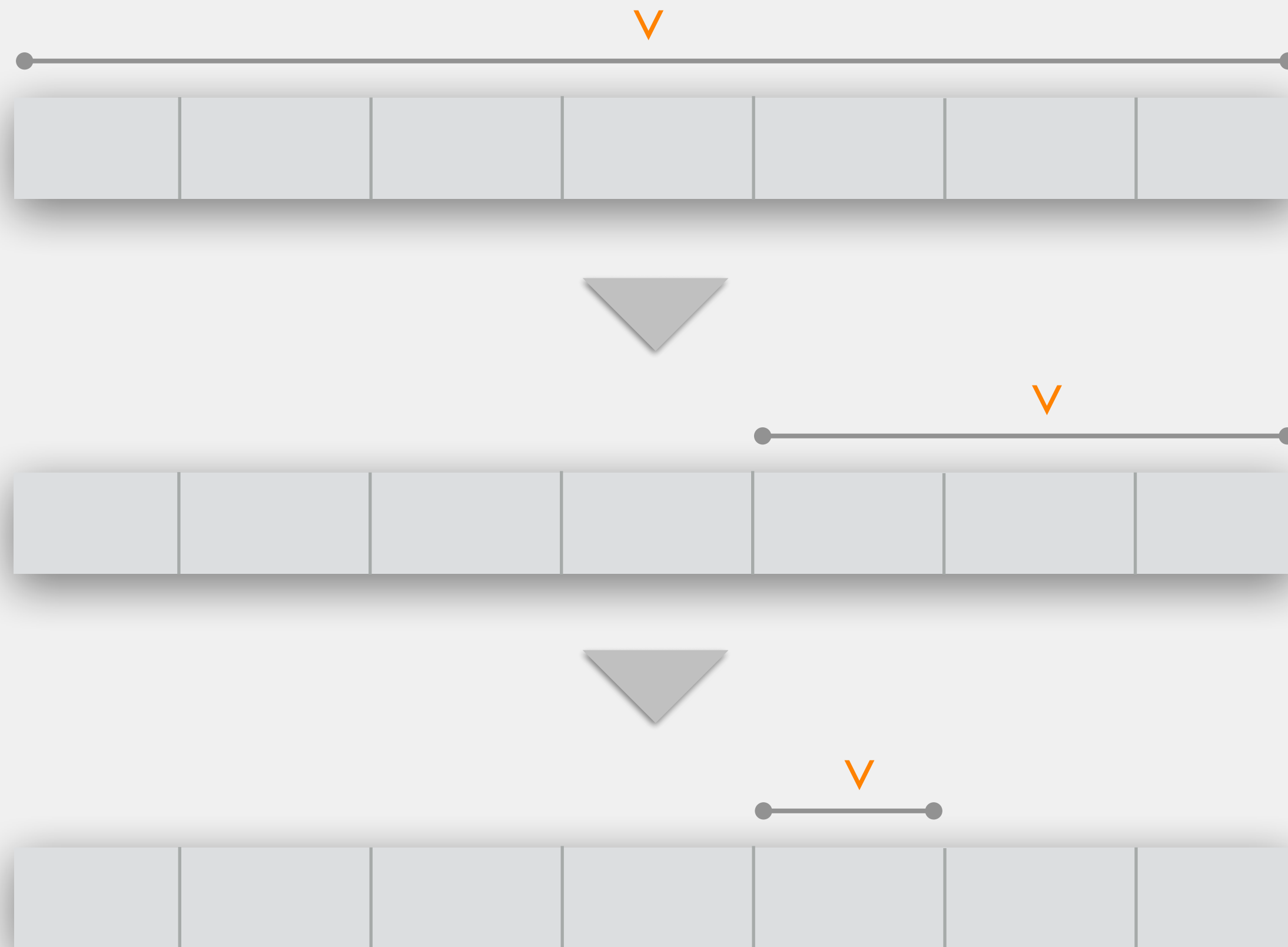
what is the best case scenario?

what is the worst case scenario?

Use the array below as an example if helpful

grades (int[])														
40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

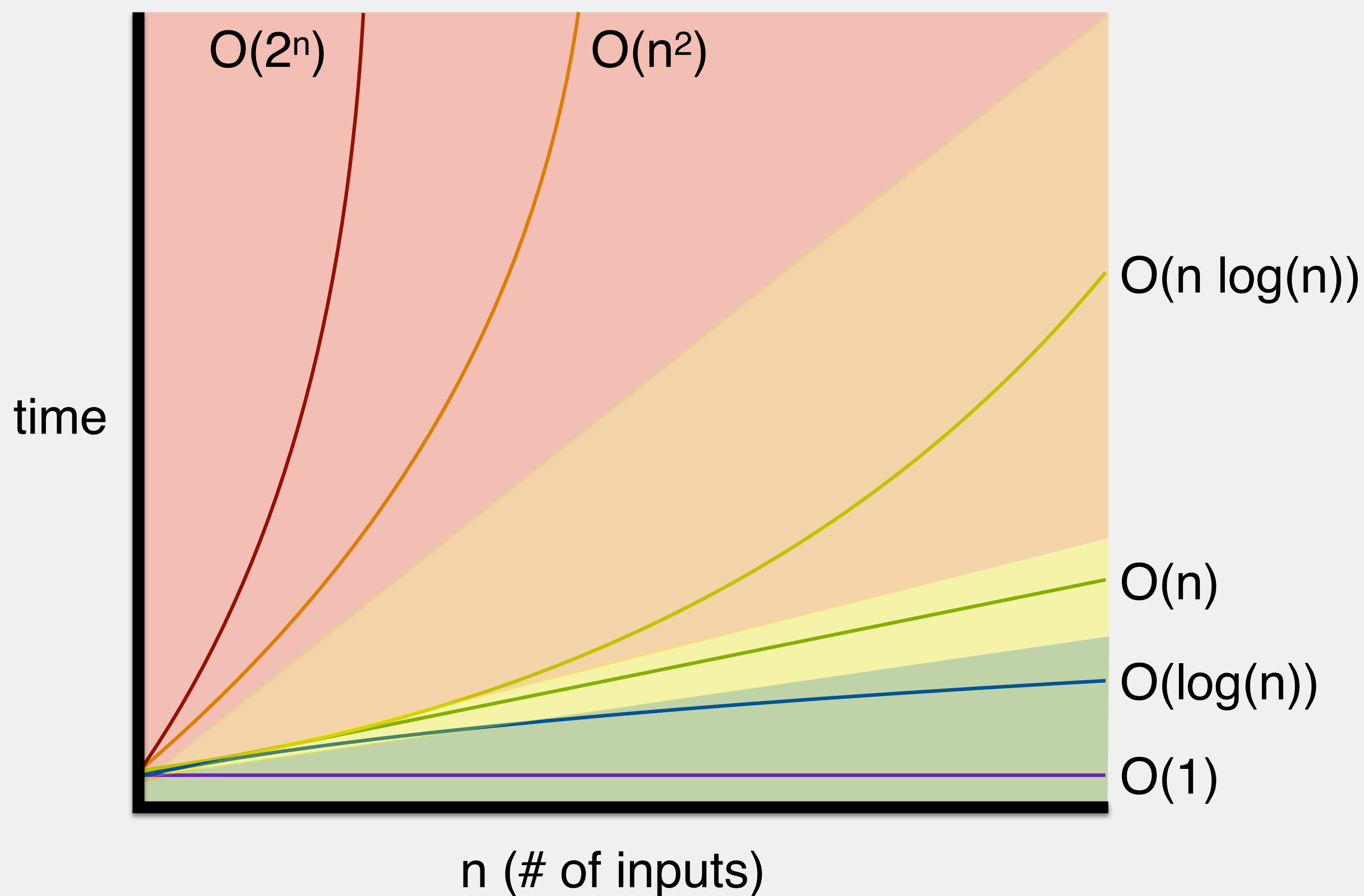
$O(\log(n))$ Algorithm



Most divide and conquer algorithms involve a worst-case runtime with a $\log(n)$ term

For Binary Search, $\log(n)$ is an approximation of the number of divisions necessary to arrive at an answer in the worst-case scenario

Big O Notation



$O(\log(n))$ is actually pretty good!

Aside: logarithms

What is $\log_b n$?

A way to think about logs:

To what power must I raise b to get n ?

What is $\log_{10} 1000$?

What is $\log_5 625$?

What is $\log_2 32768$?

Exercise: Runtime Analysis

Fill in the following chart with runtimes using binary search

	sorted list (no repeats)
<i>search for the smallest value</i>	
<i>search for the largest value</i>	
<i>search for the median value</i>	
<i>search for a value that doesn't exist</i>	
<i>search for some random value that does exist (worst case)</i>	

grades (int[])														
40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Exercise: Runtime Analysis

Fill in the following chart with runtimes using binary search

	sorted list (no repeats)
<i>search for the smallest value</i>	$O(\log(n))$
<i>search for the largest value</i>	$O(\log(n))$
<i>search for the median value</i>	$O(1)$
<i>search for a value that doesn't exist</i>	$O(\log(n))$
<i>search for some random value that does exist (worst case)</i>	$O(\log(n))$

grades (int[])														
40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Exercise: Runtime Analysis

Fill in the following chart with runtimes using binary search

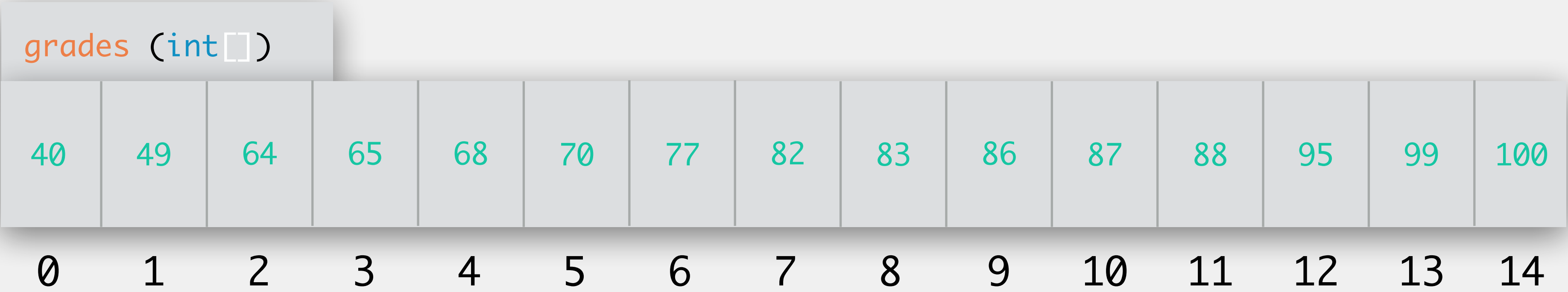
	sorted list (no repeats)
<i>search for the smallest value</i>	$\Theta(\log(n))$ $O(1)$
<i>search for the largest value</i>	$\Theta(\log(n))$ $O(1)$
<i>search for the median value</i>	$O(1)$
<i>search for a value that doesn't exist</i>	$O(\log(n))$
<i>search for some random value that does exist (worst case)</i>	$O(\log(n))$

grades (int[])														
40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Runtime Analysis

Compare and contrast - where are we doing worse vs better? Is the tradeoff worth it?

	sorted list (linear search)	sorted list (binary search)
<i>search for the smallest value</i>	$O(1)$	$\Theta(\log(n))$ $O(1)$
<i>search for the largest value</i>	$O(1)$	$\Theta(\log(n))$ $O(1)$
<i>search for the median value</i>	$O(1)$	$O(1)$
<i>search for a value that doesn't exist</i>	$O(n)$	$O(\log(n))$
<i>search for some random value that does exist (worst case)</i>	$O(n)$	$O(\log(n))$



Runtime Analysis

What value(s) are you looking for...

	sorted list (linear search)	sorted list (binary search)
<i>best case</i>		
<i>worst case</i>		

40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Runtime Analysis

What value(s) are you looking for...

	sorted list (linear search)	sorted list (binary search)
<i>best case</i>	40	
<i>worst case</i>		

40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Runtime Analysis

What value(s) are you looking for...

	sorted list (linear search)	sorted list (binary search)
<i>best case</i>	40	
<i>worst case</i>	100	

40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Runtime Analysis

What value(s) are you looking for...

	sorted list (linear search)	sorted list (binary search)
<i>best case</i>	40	82
<i>worst case</i>	100	

40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Runtime Analysis

What value(s) are you looking for...

	sorted list (linear search)	sorted list (binary search)
<i>best case</i>	40	82
<i>worst case</i>	100	40, 64, 68, 77, 83, 87, 95, 100

40	49	64	65	68	70	77	82	83	86	87	88	95	99	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Code: Binary Search (iterative)

```
private static int binarySearch(int arr[], int toFind) {  
    int begin = 0;  
    int end = arr.length - 1;  
  
    while (begin <= end) {  
        int mid = (begin + end) / 2; // Find the midpoint  
  
        if (arr[mid] == toFind) { // Found it!  
            return mid;  
        } else if (arr[mid] < toFind) { // mid value too small  
            begin = mid + 1;  
        } else { /* arr[mid] > toFind */ // mid value too large  
            end = mid - 1;  
        }  
    }  
  
    return -1; // Failed search  
}
```

Code: Binary Search (recursive)

```
private static int binarySearch(int arr[], int toFind) {
    return binSearchHelper(arr, toFind, 0, arr.length - 1);
}

private static int binSearchHelper(int arr[], int toFind, int begin, int end) {
    if (begin > end) {
        return -1; // Failed search
    }
    int mid = (begin + end) / 2; // Find the midpoint
    if (arr[mid] == toFind) { // Found it!
        return mid;
    } else if (arr[mid] < toFind) { // mid value too small
        return binSearchHelper(arr, toFind, mid + 1, end);
    } else { /* arr[mid] > toFind */ // mid value too large
        return binSearchHelper(arr, toFind, begin, mid - 1);
    }
}
```

Searching on Data Structures

Discussed linear and binary search on arrays

runtimes would be comparable for an array list

What about a singly linked list?

can linear search be performed on one?

what about binary search on a singly linked list?

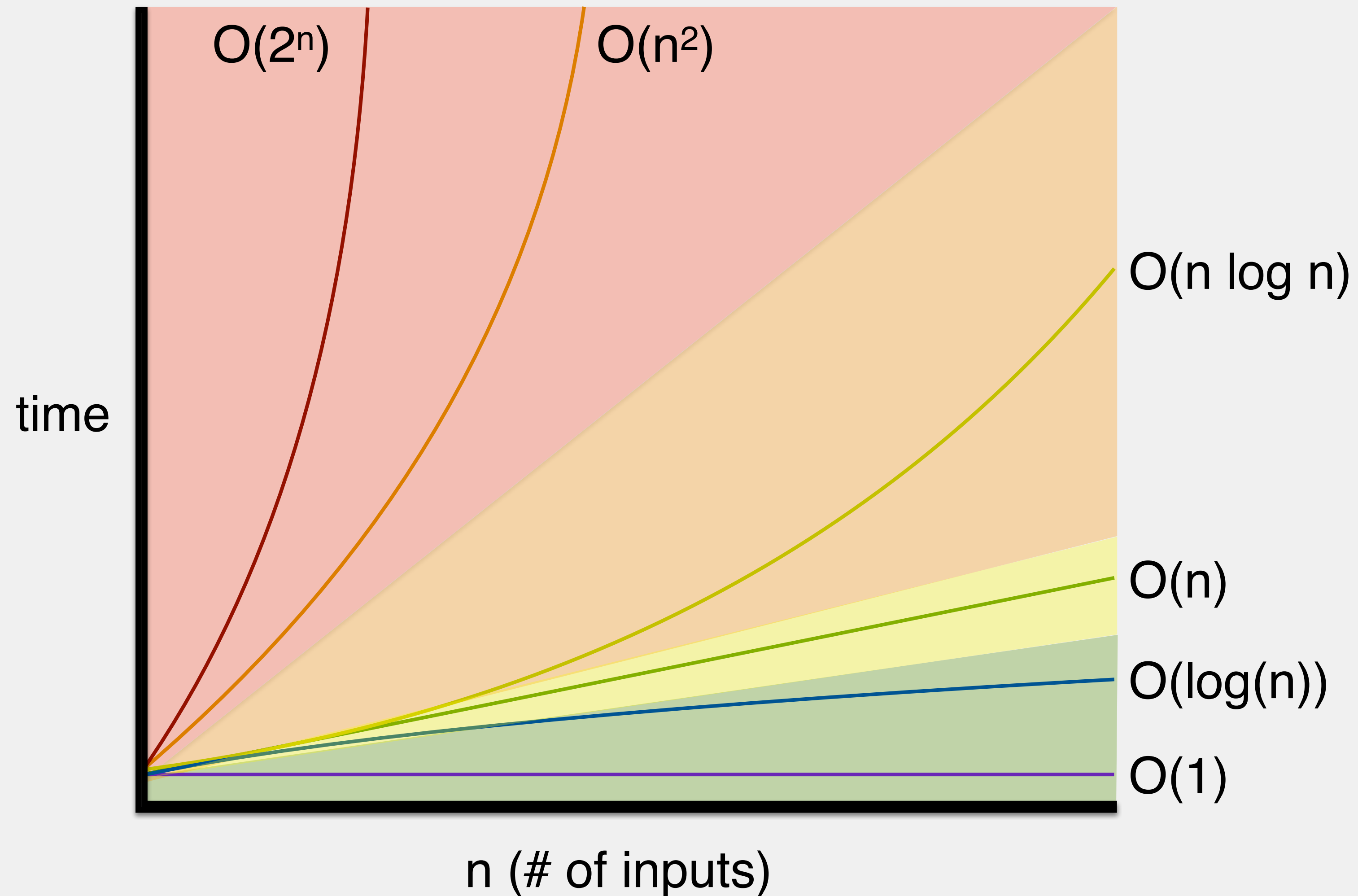
what would the runtimes be like?

Linked Lists & Search

Consider the following chart for a singly linked list

	unsorted linked list (linear search)	sorted linked list (linear search)	<i>sorted array (linear search)</i>	sorted linked list (binary search)	<i>sorted array (binary search)</i>
<i>search for the smallest value</i>	O(n)	O(1)	<i>O(1)</i>	O(1)	<i>O(log(n))</i>
<i>search for the largest value</i>	O(n)	O(n)	<i>O(1)</i>	O(n)	<i>O(log(n))</i>
<i>search for the median value</i>	O(n)	O(n)	<i>O(1)</i>	O(n)	<i>O(1)</i>
<i>search for a value that doesn't exist</i>	O(n)	O(n)	<i>O(n)</i>	O(n log(n))	<i>O(log(n))</i>
<i>search for some random value that does exist (worst case)</i>	O(n)	O(n)	<i>O(n)</i>	O(n log(n))	<i>O(log(n))</i>

Big O Notation



$O(n \log n)$ is significantly larger than $O(n)$ and $O(\log n)$ as n gets large...

Linked Lists & Search

	unsorted linked list (linear search)	sorted linked list (linear search)	<i>sorted array (linear search)</i>	sorted linked list (binary search)	<i>sorted array (binary search)</i>
<i>search for the median value</i>	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$

