Algorithms and Data Structures for Data Science
Search

CS 277
Brad Solomon

March 8, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Lab_recursion Feedback

Average score: 87%

PL average time: 62 minutes

Only two students filled out survey

Both seemed to find it reasonably good!

Illinois Data Science Club

JOIN OUR TEAM!

PROJECT LEAD

- **EXPERIENCE WITH ML, AI & PYTHON**
- **HELP LEAD A SEMESTER LONG PROJECT HACKATHON**
- **CAN COMMIT TO AT LEAST 5 HOURS A WEEK**

MARKETING DIRECTOR

- **EXPERIENCE WITH SOCIAL MEDIA AND WEBSITE MANAGEMENT**
- **ESTABLISH IDSC'S BRAND & PRESENCE ON CAMPUS**
- **INITIATE MARKETING AND SOCIAL MEDIA STRATEGIES**
- **CAN COMMIT TO AT LEAST 3 HOURS A WEEK**

UIUCDSC UIUCDSC@GMAIL.COM

Learning Objectives

Introduce the fundamental search problem

Introduce and implement binary search

Review hash tables, sorting, and search

The Search Problem

Given a collection of objects, C, with comparable values and an object of interest, q, find the first instance of $q \in C$.

Input:

4	5	6	7	8	9	10	11	12	13

Output: Index of q if it exists, -1 otherwise

Naive Linear Search

```
def naive_linear(inList, val):

    for i, obj in enumerate(inList):

        if val == obj:

            return i
```

6	1	0	3	7	9	2	4

Naive Sorted Search

`Find(3)`

0	1	2	4	5	6	7	8	9	10

Naive Sorted Search

```
def naive_sorted(inList, val):

    for i, obj in enumerate(inList):

        if val == obj:

            return i

        elif val > obj:

            return -1
```

0	1	2	3	4	6	7	9

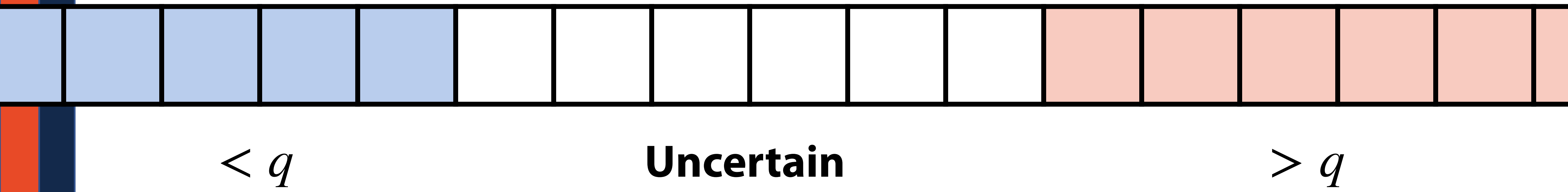
Binary Search

`Find(7)`

0	1	2	3	4	5	6	7	8	9

Binary Search

A binary search (for object q) partitions the search space into three regions

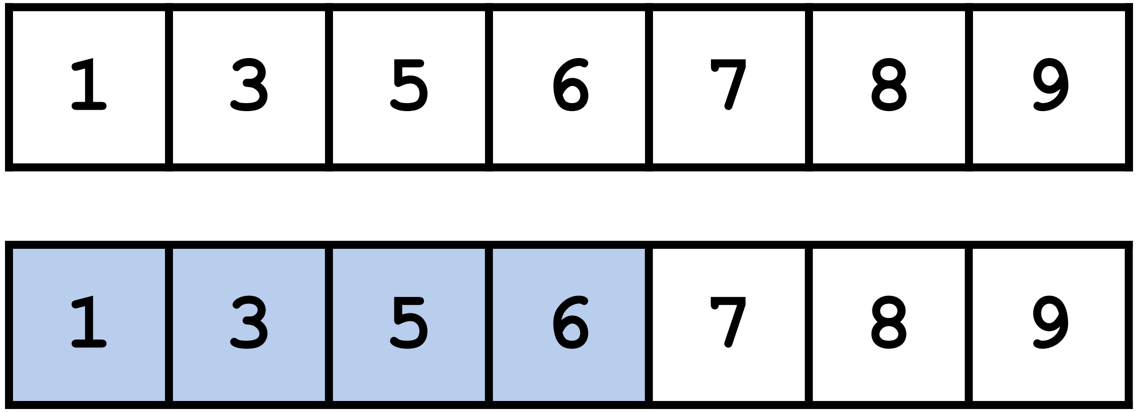


If we are looking for q, where might we find it?

How can we track this information?

Binary Search

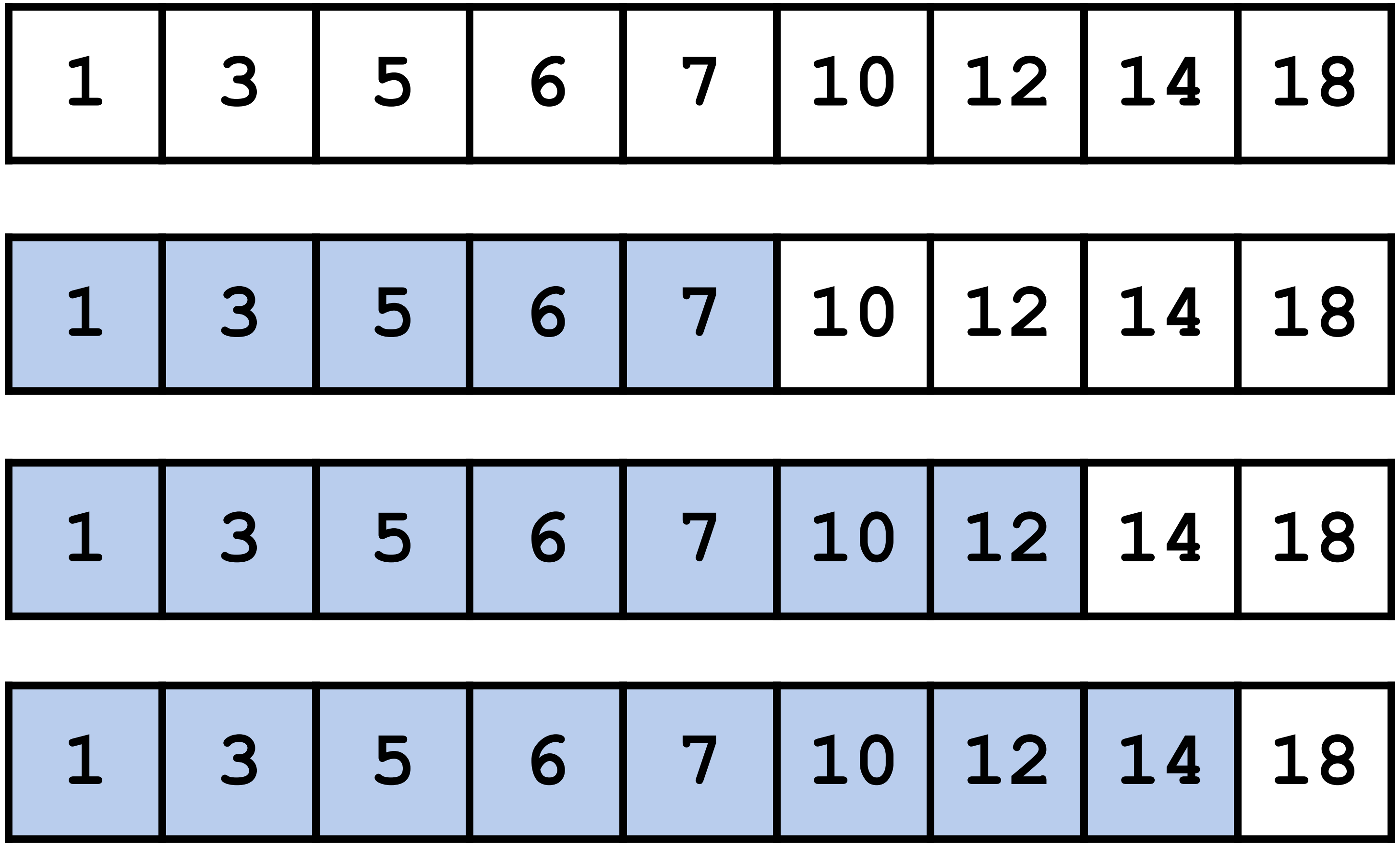
`Find(8)`



1. **Find midpoint**
2. **Compare midpoint**
3. **Update range**

Binary Search

`Find(18)`



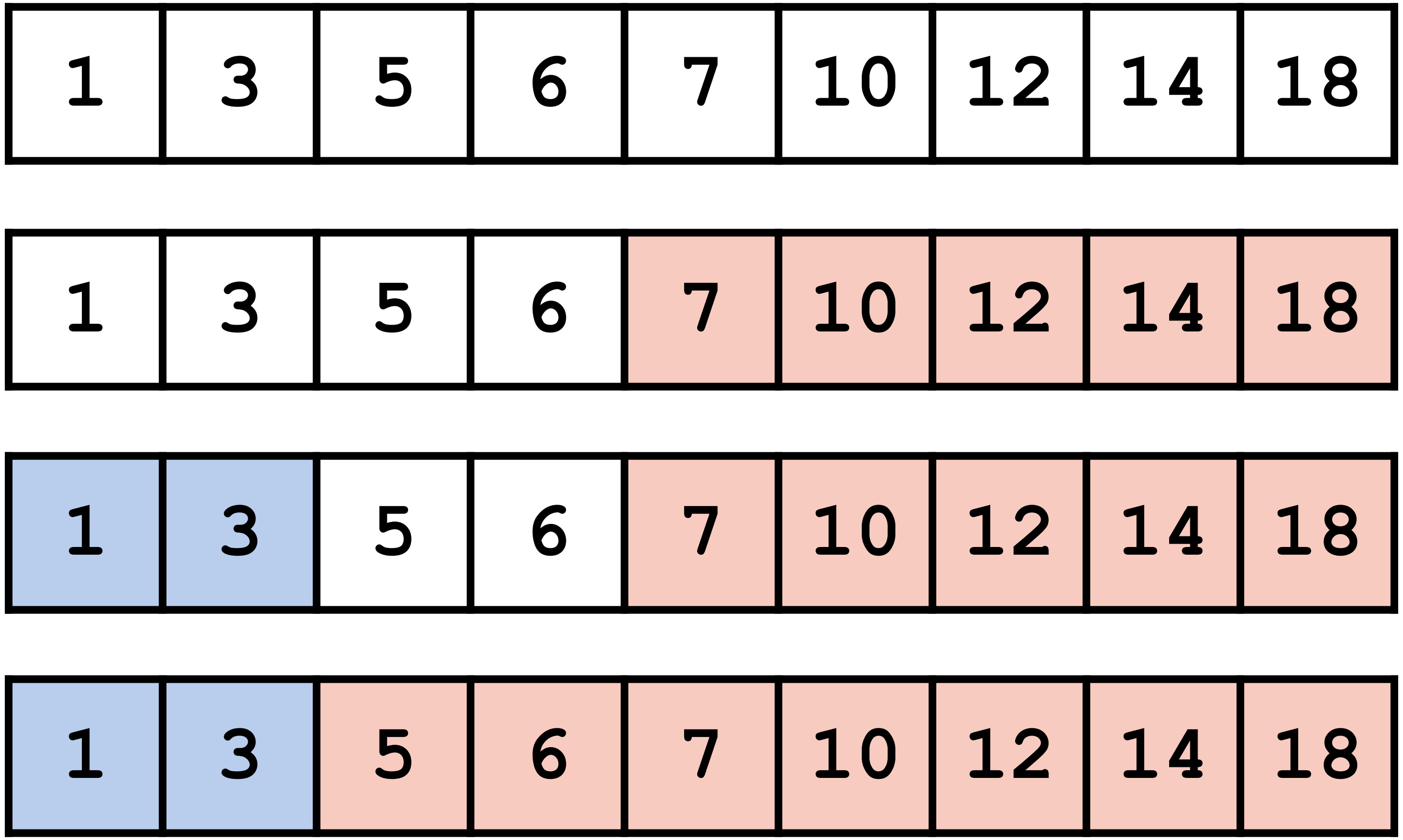
1. Find midpoint

2. Compare midpoint

3. Update range

Binary Search

Find(4)



1. Find midpoint

2. Compare midpoint

3. Update range

Recursive Binary Search

0	3	7	5	8	9	2	1	4	6

Base Case:

Recursive Step:

Combining:

Binary Search

```
def binary_search(inList, q):










def recursive_BS(inList, q, start, end):
```

0	1	2	3	4	6	7	9

Binary Search Efficiency

0	1	2	6	7	8	9	10	11

0	1	2	6

2	6

6

Hash Tables vs Binary Search

The hash table is generally superior for storing unordered objects

What are some situations where you can't use a hash table?

Range Search

Given a collection of objects, C, with comparable values and an object of interest, q, find ~~the first~~ ALL instance(s) of $q \in C$.

Input:	0	1	2	2	2	2	2	3	4	5

Output: Range of indices matching q if it exists, $(-1, -1)$ otherwise

Binary Range Search

Find(3)

Observation: All matching values are going to be consecutive

0	2	2	2	3	3	3

1. Perform binary search

2. 'Extend' in both directions

Binary Range Search

Find(3)

Observation: All matching values are going to be consecutive

3	3	3	3	3	3	3

1. Perform binary search

2. ‘Extend’ in both directions

Binary Range Search

Find(3)

Observation: My search is looking for two *specific* values

2	3	3	3	3	4	4

1. Modify binary search to find the *first* or *last* matching value

Binary Search: Get largest match

Find(3)

```
    # THIS IS PSEUDOCODE

        if mid == q:

            # Match case:
            # Treat like query is smaller
            # Remember last match!

        elif mid > q:

            # query is smaller case
        else:

            # query is larger case

    # Final Return Snippet
    if saw_match:
        return last_match
    else:
        return -1
```

2	3	3	3	3	4	4

Binary Search: Get largest match

Find(2)

```
# THIS IS PSEUDOCODE

    if mid == q:

        # Match case:
        # Treat like query is smaller
        # Remember last match!

    elif mid > q:

        # query is smaller case
    else:

        # query is larger case

# Final Return Snippet
if saw_match:
    return last_match
else:
    return -1
```

2	2	2	2	2	2	4

Binary Search: Get smallest match

Find(3)

```
# THIS IS PSEUDOCODE

    if mid == q:

        # Match case:
        # Treat like query is larger
        # Remember last match!

    elif mid > q:

        # query is smaller case
    else:

        # query is larger case

# Final Return Snippet
if saw_match:
    return last_match
else:
    return -1
```

2	3	3	3	3	4	4

Binary Search: Get smallest match

Find(2)

```
# THIS IS PSEUDOCODE

    if mid == q:

        # Match case:
        # Treat like query is larger
        # Remember last match!

    elif mid > q:

        # query is smaller case
    else:

        # query is larger case

# Final Return Snippet
if saw_match:
    return last_match
else:
    return -1
```

2	2	2	2	2	2	4

Exam 2: Review Material

Stacks and Queues

Hashing (and Hash Tables)

Sorting Algorithms

Binary Search

Stack and Queue

Understand LIFO / FIFO

Know the allowable operations for a stack and queue and how to use them

Understand Big O efficiency

Hashing

What are the necessary properties of a hash function?

What are the necessary components of a hash table?

Describe some strategies for addressing hash collisions

Hashing

What is the worst case performance (Big O) of a general-use hash table?

What assumption did we use to examine the ***expected performance?***

Under that assumption, how does our load factor affect performance?

Sorting Algorithms

Understand the logic behind each one

Know the Big O of each method

Understand best case or worst case (when applicable)

Sorting Algorithm Tradeoffs

	Best Case Time	Worst Case time	Best Case Space	Worst Case Space
SelectionSort	$O(n^2)$	$O(n^2)$	$O(1)$	$O(1)$
InsertionSort	$O(n)$	$O(n^2)$	$O(1)$	$O(1)$
MergeSort	$O(n \log n)$	$O(n \log n)$	$O(n)$	$O(n)$
QuickSort	$O(n \log n)$	$O(n^2)$	$O(\log n)$	$O(n)$

What sorting algorithm would you use…?

0	1	2	3	4	5	6	7	8	9

What sorting algorithm would you use…?

9	8	7	6	5	4	3	2	1	0

Search Algorithms

Understand how to code and walk through binary search

Know the Big O of binary search