Binary Search Trees: Introduction

Daniel Kane

Department of Computer Science and Engineering
University of California, San Diego

Data Structures
Data Structures and Algorithms

Learning Objectives

- Provide examples of the sorts of problems we hope to solve with Binary Search Trees.
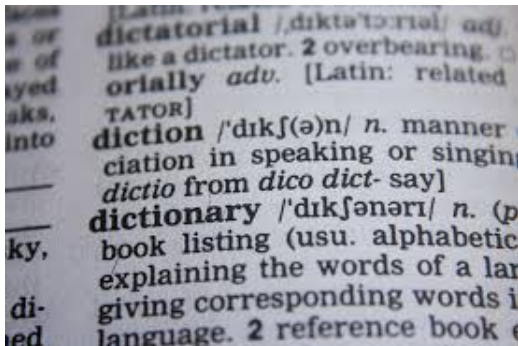- Show why data structures that we have already covered are insufficient.

Outline

Dictionary Search

Find all words that start with some given string.

Date Ranges

Find all emails received in a given period.

FROM	KNOW	TO	SUBJECT	SENT TIME ▾	
"lawiki.i2p admin" <J5uF>		Bote User <uhOd>	hi	Unknown	🗑
anonymous		Bote User <uhOd>	Sanders 2016	Aug 30, 2015 3:27 PM	🗑
anonymous		Bote User <uhOd>	I2PCon 2016	Aug 30, 2015 3:25 PM	🗑
Anon Developer <gvbM>		Bote User <uhOd>	Re: Bote changess	Aug 30, 2015 2:54 PM	🗑
I2P User <uUUx>		Bote User <uhOd>	Hello World!	Aug 30, 2015 2:51 PM	🗑

Inbox

Closest Height

Find the person in your class whose height is closest to yours.
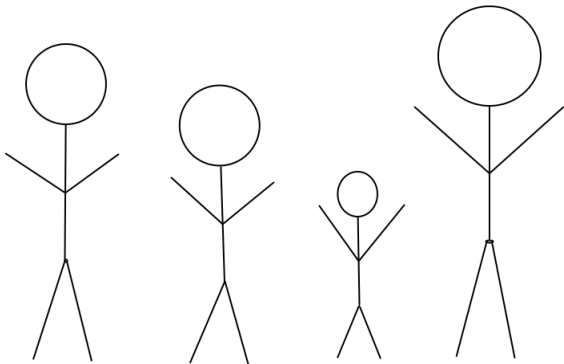
Local Search

Definition

A Local Search Datastructure stores a number of elements each with a key coming from an ordered set. It supports operations:

- $\texttt{RangeSearch}(x, y)$: Returns all elements with keys between x and y.

- $\texttt{NearestNeighbors}(z)$: Returns the element with keys on either side of z.

Example

| 1 | 4 | 6 | 7 | 10 | 13 | 15 |

Example

| 1 | 4 | 6 | 7 | 10 | 13 | 15 |

RangeSearch$(5, 12)$

| 1 | 4 | **6** | **7** | **10** | 13 | 15 |

Example

1	4	6	7	10	13	15

RangeSearch$(5, 12)$

1	4	6	7	10	13	15

NearestNeighbors(3)

1	4	6	7	10	13	15

Dynamic Data Structure

We would also like to be able to modify the data structure as we go.

- Insert(x): Adds a element with key x.
- Delete(x): Removes the element with key x.

Example

| 1 | 4 | 6 | 7 | 10 | 13 | 15 |

Example

1	4	6	7	10	13	15

Insert(3)

1	3	4	6	7	10	13	15

Example

1	4	6	7	10	13	15

`Insert(3)`

1	3	4	6	7	10	13	15

`Delete(10)`

1	3	4	6	7	13	15

Problem

If an empty data structure is given these commands what does it output at the end?

- `Insert`(3)
- `Insert`(8)
- `Insert`(5)
- `Insert`(10)
- `Delete`(8)
- `Insert`(12)
- `NearestNeighbors`(7)

Answer

Outline

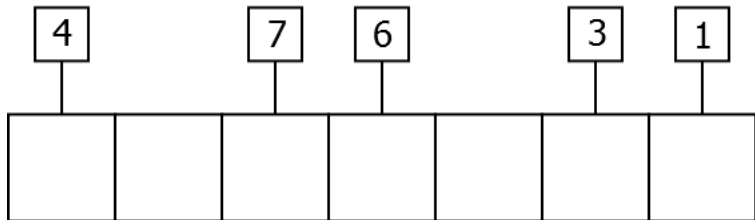# Hash Table

- `RangeSearch`:　　　　Impossible ×
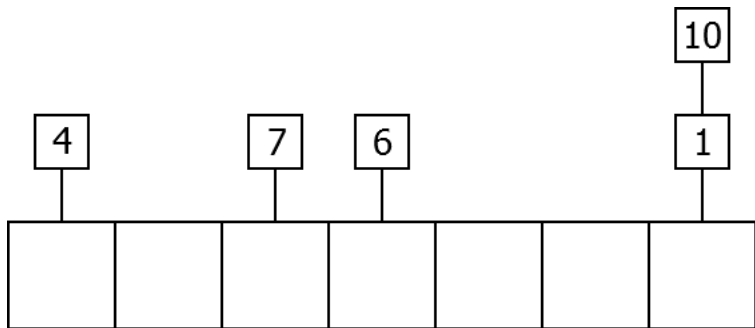
Hash Table

- `RangeSearch`:   Impossible ✗
- `NearestNeighbors`:   Impossible ✗
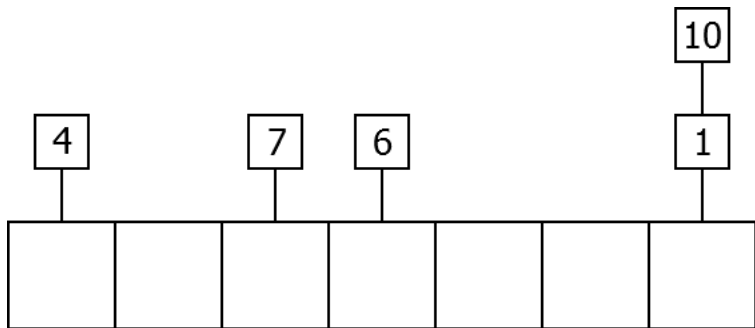
Hash Table

- RangeSearch: Impossible ✗
- NearestNeighbors: Impossible ✗
- Insert: $O(1)$ ✓
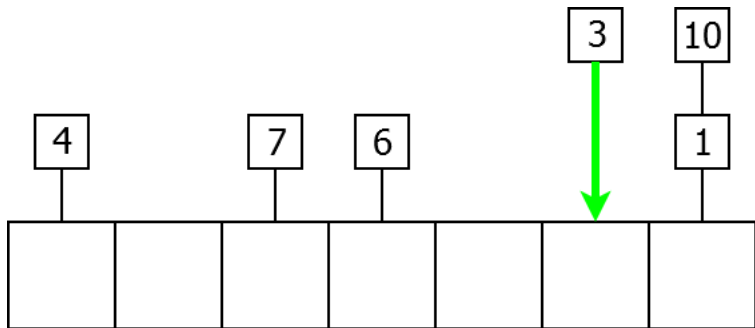
Hash Table

- `RangeSearch`:   Impossible ×
- `NearestNeighbors`:   Impossible ×
- `Insert`: $O(1)$ ✓
- `Delete`: $O(1)$ ✓

Array

- `RangeSearch`: $O(n)$ ✗

Array

- `RangeSearch`: $O(n)$ ✗
- `NearestNeighbors`: $O(n)$ ✗

| 7 | 10 | 4 | 13 | 1 | 6 | 15 |

Array

- `RangeSearch`: $O(n)$ ✗
- `NearestNeighbors`: $O(n)$ ✗
- `Insert`: $O(1)$ ✓
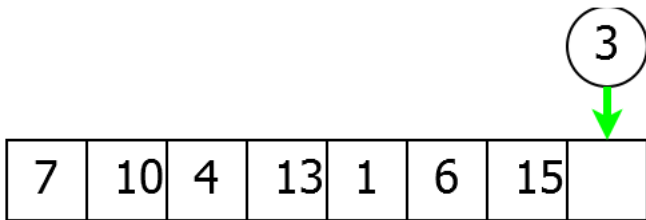
Array

- `RangeSearch`: $O(n)$ ✗
- `NearestNeighbors`: $O(n)$ ✗
- `Insert`: $O(1)$ ✓
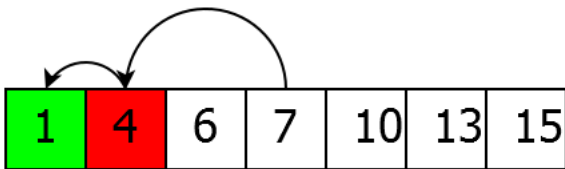- `Delete`: $O(1)$ ✓
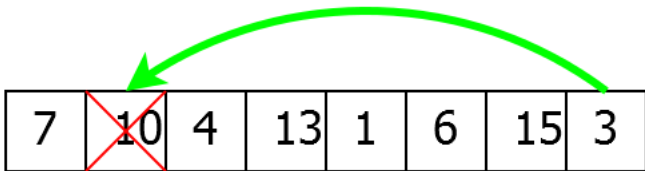
Sorted Array

- `RangeSearch`: $O(\log(n))$ ✓
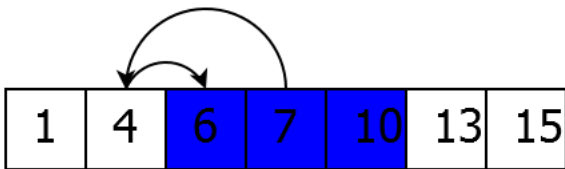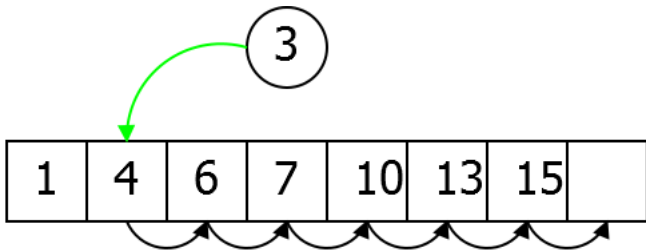
Sorted Array

- `RangeSearch`: $O(\log(n))$ ✓
- `NearestNeighbors`: $O(\log(n))$ ✓
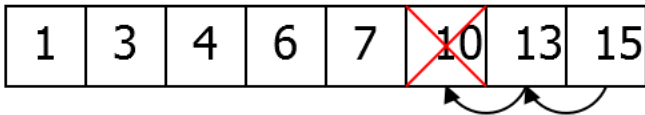
Sorted Array

- RangeSearch: $O(\log(n))$ ✓
- NearestNeighbors: $O(\log(n))$ ✓
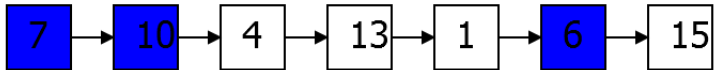- Insert: $O(n)$ ✗

Sorted Array

- `RangeSearch`: $O(\log(n))$ ✓
- `NearestNeighbors`: $O(\log(n))$ ✓
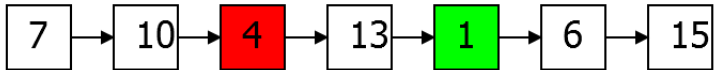- `Insert`: $O(n)$ ✗
- `Delete`: $O(n)$ ✗
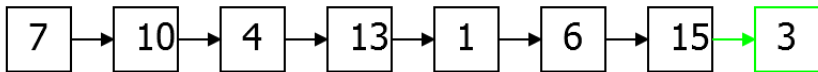
Linked List

- `RangeSearch:` $\qquad\qquad O(n)$ ×

Linked List

- `RangeSearch`: $O(n)$ ✗
- `NearestNeighbors`: $O(n)$ ✗
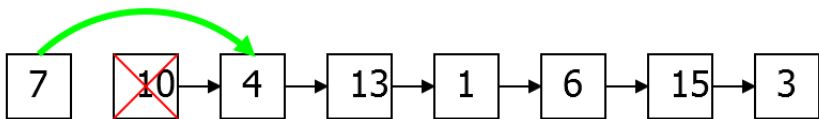
Linked List

- `RangeSearch`: $O(n)$ ×
- `NearestNeighbors`: $O(n)$ ×
- `Insert`: $O(1)$ ✓

Linked List

- `RangeSearch`: $O(n)$ ✗
- `NearestNeighbors`: $O(n)$ ✗
- `Insert`: $O(1)$ ✓
- `Delete`: $O(1)$ ✓

Need Something New

Problem

Previous data structures won't work. We need something new.