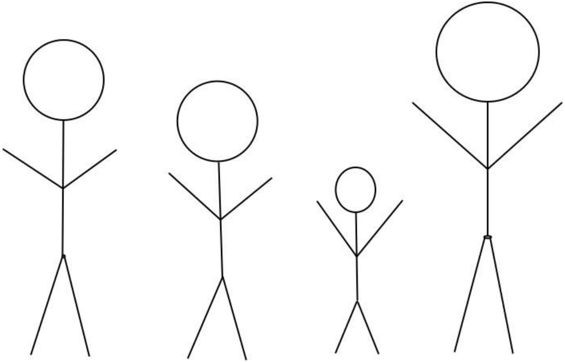
Abstract data Type: Range

Lecture 02.09

By Marina Barsky

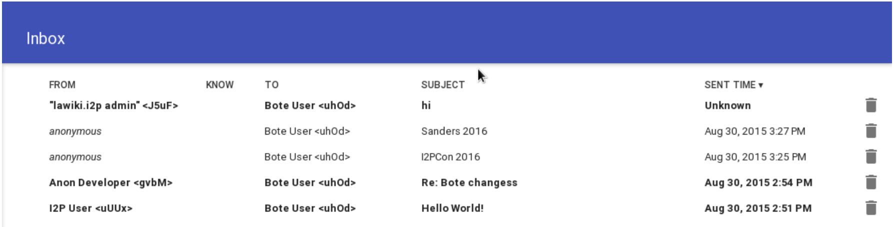
Motivation 1: Closest Height

Find 3 people in your class whose height is closest to yours.



Motivation 2: Date Ranges

Find all emails received in a given period

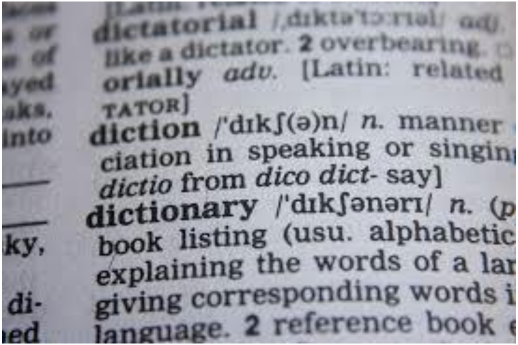


Inbox

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**

Motivation 3: Partial Search

Find all words that **start with** some given *prefix*



s or
e of
yed
aks,
into

ky,

di-
ed

[Latin
dictatorial /ˌdɪktəˈtɔːrɪəl/ *adj.*
like a dictator. **2** overbearing. □
orially *adv.* [Latin: related
TATOR]
diction /ˈdɪkʃ(ə)n/ *n.* manner
ciation in speaking or singin
dictio from *dico dict-* say]
dictionary /ˈdɪkʃənərɪ/ *n.* (p
book listing (usu. alphabetic
explaining the words of a lan
giving corresponding words i
language. **2** reference book

Abstract Data Type: Range

Specification

A ***Local Range*** **ADT** stores a number of elements each with a *key* and supports the following operations:

- ➔ ***RangeSearch****(lo, hi)*: returns all elements with keys between *lo* and *hi*
- ➔ ***NearestNeighbors****(x, k)*: returns *k* elements with keys closest to *x*

Example

1	4	6	7	10	13	15

Example

1	4	6	7	10	13	15

***RangeSearch**(5, 13)*

Example

1	4	6	7	10	13	15

***RangeSearch**(5, 13)*

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

Example

1	4	6	7	10	13	15

***RangeSearch**(5, 13)*

1	4	6	7	10	13	15

***NearestNeighbors**(5, 3)*

Example

1	4	6	7	10	13	15

***RangeSearch**(5, 13)*

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

***NearestNeighbors**(5, 3)*

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

Sorted keys

1	4	6	7	10	13	15

- Is seems that it is a good idea to store keys **in a sorted order**

Dynamic Data Structure

- ➢ Store keys in **sorted order**
- ➢ Also want to be able to add/remove keys efficiently:

Insert(x): Adds an element with key *x*

Delete(x): Removes the element with key *x*

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

Implementing Range ADT

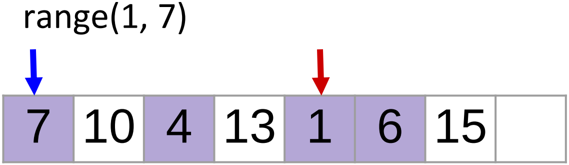
1	4	6	7	10	13	15

Let’s try known data structures:

- Array
- Sorted array
- Linked list
- Hash table

Array

- ➔ Range Search: $O(n)$ ×



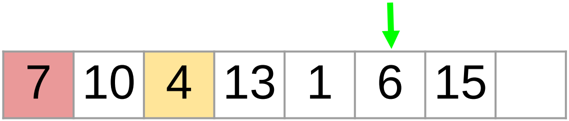
range(1, 7)

7	10	4	13	1	6	15	

Array

- ➔ Range Search: $O(n)$ ×
- ➔ Nearest Neighbors: $O(n)$ ×

nearestNeighbors(6, 2)

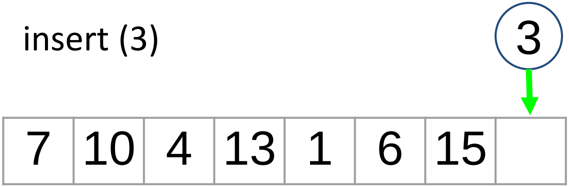


7	10	4	13	1	6	15	

Array

- ➔ Range Search: $O(n)$ ✗
- ➔ Nearest Neighbors: $O(n)$ ✗
- ➔ Insert: $O(1)$ ✓

insert (3)

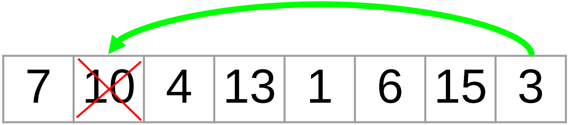


7	10	4	13	1	6	15	

Array

- ➔ Range Search: $O(n)$ ×
- ➔ Nearest Neighbors: $O(n)$ ×
- ➔ Insert: $O(1)$ ✓
- ➔ Delete: $O(1)$ ✓

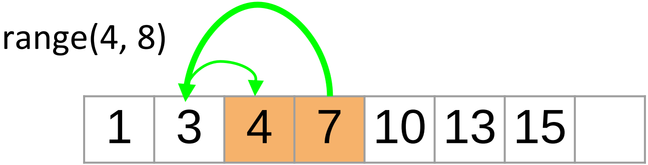
delete (10)



7	~~10~~	4	13	1	6	15	3

Sorted Array

- ➔ Range Search: $O(\log(n))$ ✓



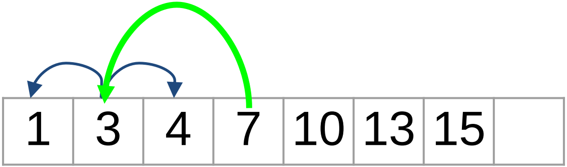
range(4, 8)

1	3	4	7	10	13	15	

Sorted Array

- ➔ Range Search: $O(\log(n))$ ✓
- ➔ Nearest Neighbors: $O(\log(n))$ ✓

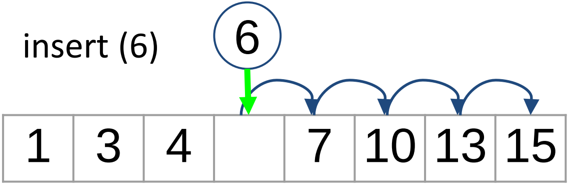
nearestNeighbors(3, 2)



Sorted Array

- ➔ Range Search: $O(\log(n))$ ✓
- ➔ Nearest Neighbors: $O(\log(n))$ ✓
- ➔ Insert: $O(n)$ ✗

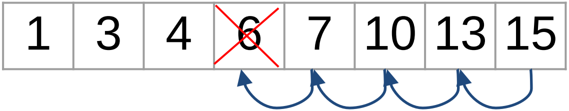
insert (6)



Sorted Array

- ➔ Range Search: $O(\log(n))$ ✓
- ➔ Nearest Neighbors: $O(\log(n))$ ✓
- ➔ Insert: $O(n)$ ✗
- ➔ Delete: $O(n)$ ✗

delete (6)

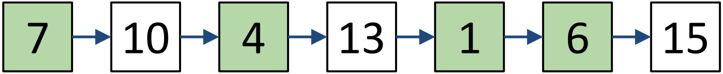


1	3	4	~~6~~	7	10	13	15

Linked List

➔ Range Search: $O(n)$ ×

range (4, 9)

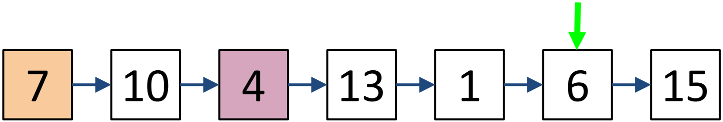


7 → 10 → 4 → 13 → 1 → 6 → 15

Linked List

- Range Search: $O(n)$ ×
- Nearest Neighbors: $O(n)$ ×

nearestNeighbors(6, 2)



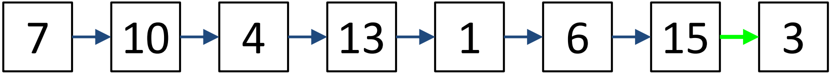
Linked List

➔ Range Search: $O(n)$ ×

➔ Nearest Neighbors: $O(n)$ ×

➔ Insert: $O(1)$ ✓

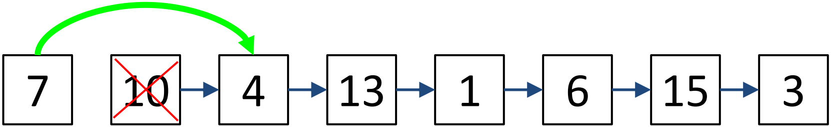
insert (3)



Linked List

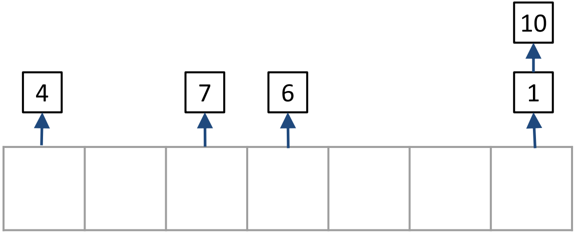
- ➔ Range Search: $O(n)$ ×
- ➔ Nearest Neighbors: $O(n)$ ×
- ➔ Insert: $O(1)$ ✓
- ➔ Delete: $O(1)$ ✓

delete (10)



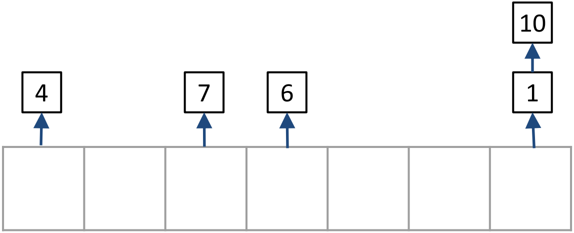
Hash Table

➔ Range Search: Impossible ✗



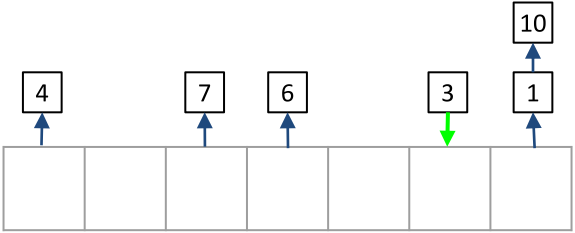
Hash Table

- ➔ Range Search: Impossible ×
- ➔ Nearest Neighbors: Impossible ×



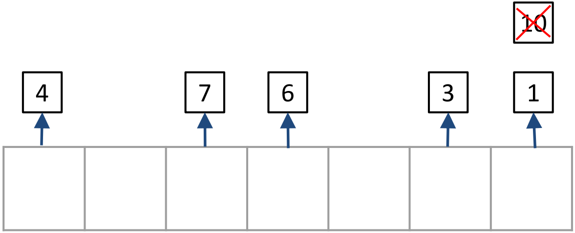
Hash Table

- ➔ Range Search: Impossible ×
- ➔ Nearest Neighbors: Impossible ×
- ➔ Insert: $O(1)$ ✓



Hash Table

- Range Search: Impossible ×
- Nearest Neighbors: Impossible ×
- Insert: $O(1)$ ✓
- Delete: $O(1)$ ✓

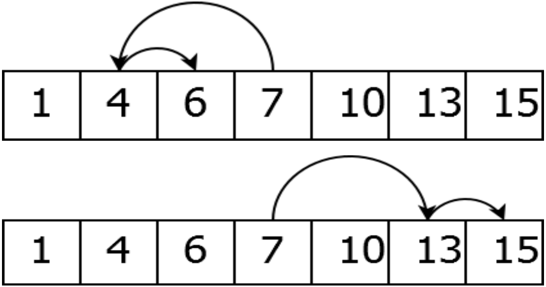


Nothing works

- We want efficient data structure for Local Range ADT
- None of the existing data structures work
- Sorted arrays are good for search but not for update

We need something new

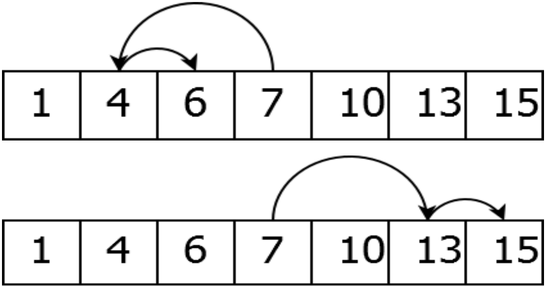
Binary Search

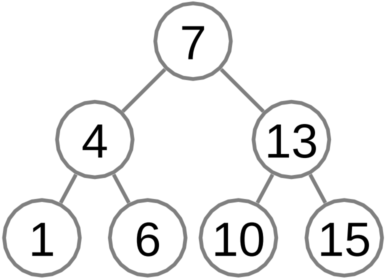


1	4	6	7	10	13	15

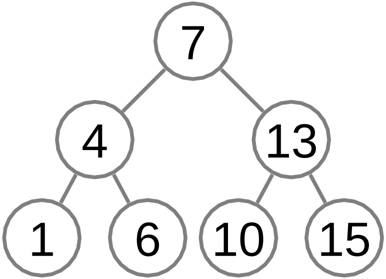
1	4	6	7	10	13	15

Record search questions





We need a tree


7
4
13
1
6
10
15