Searching I

- Search Algorithms
- Search Structures: Dictionaries

Searching

Key question: How can we store data (often a set of key-value pairs) so that we can then search for specific items efficiently?

Keys and Values

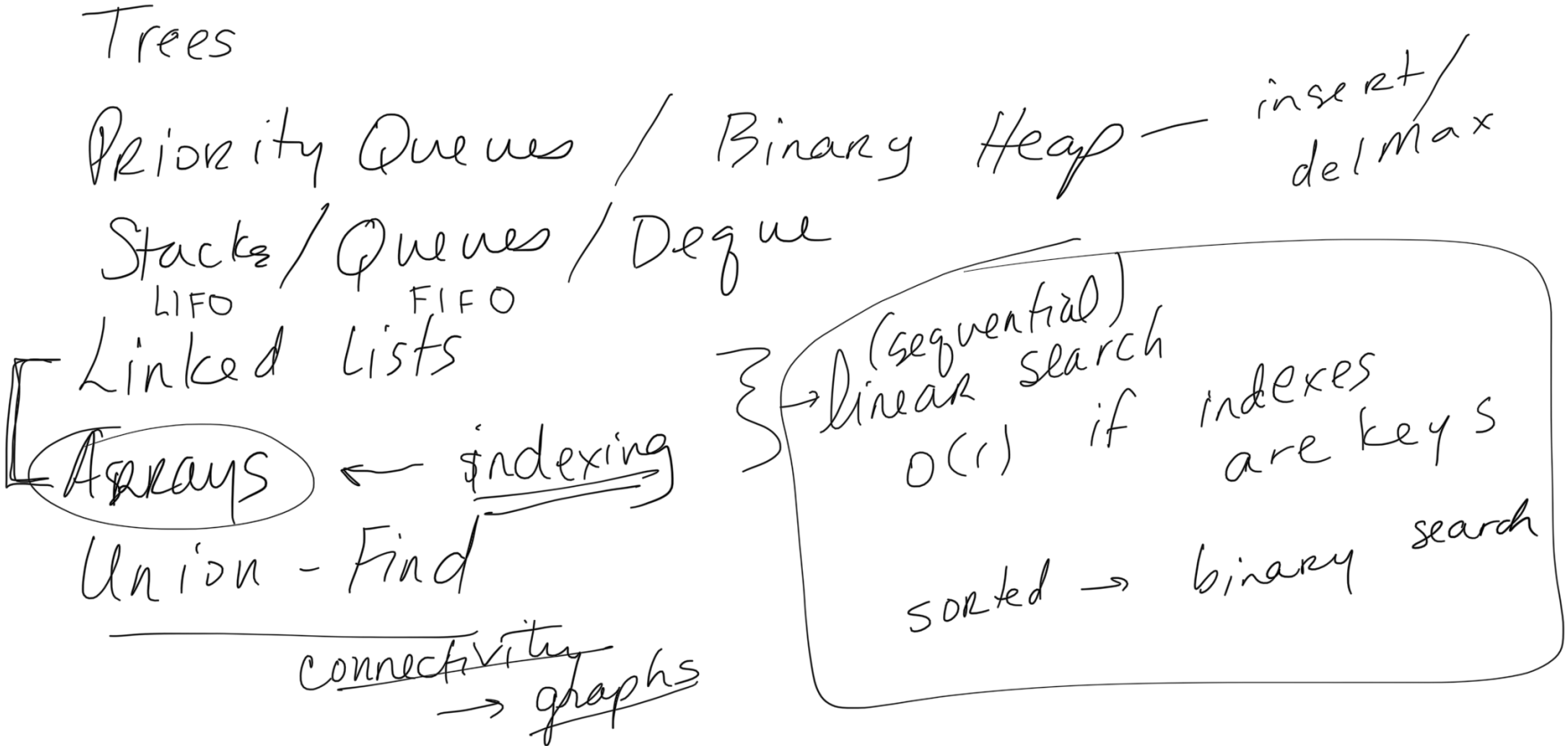
V - value → actual data

K → key → used for search / insert

(K, V)

Keys may be comparable.

Review of data structures…



Symbol Table
Index

The Dictionary ADT

→ storing (key, value) pairs

operations:

(put) insert (key, value)

(get) search (key)

(remove) delete (key)

Applications of Dictionaries

- Dictionary
- Database
- Symbol Table

Should we allow duplicate keys?

How would you implement a Dictionary?

(Hashtable) Array of Linked Lists

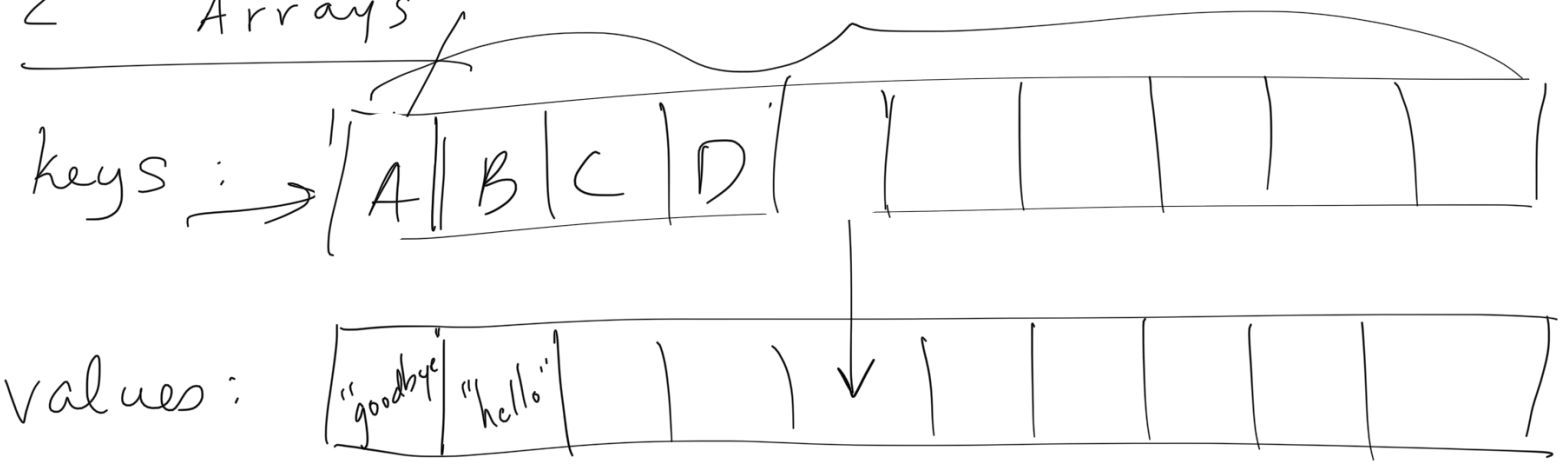
two arrays → keys & values

sorted ↓ binary search

BST

Tree
Trie

2 Arrays



(1, "hello")

(0, "goodbye")

array of pairs

(key, value)

insert : O(N)
search : O(log N)

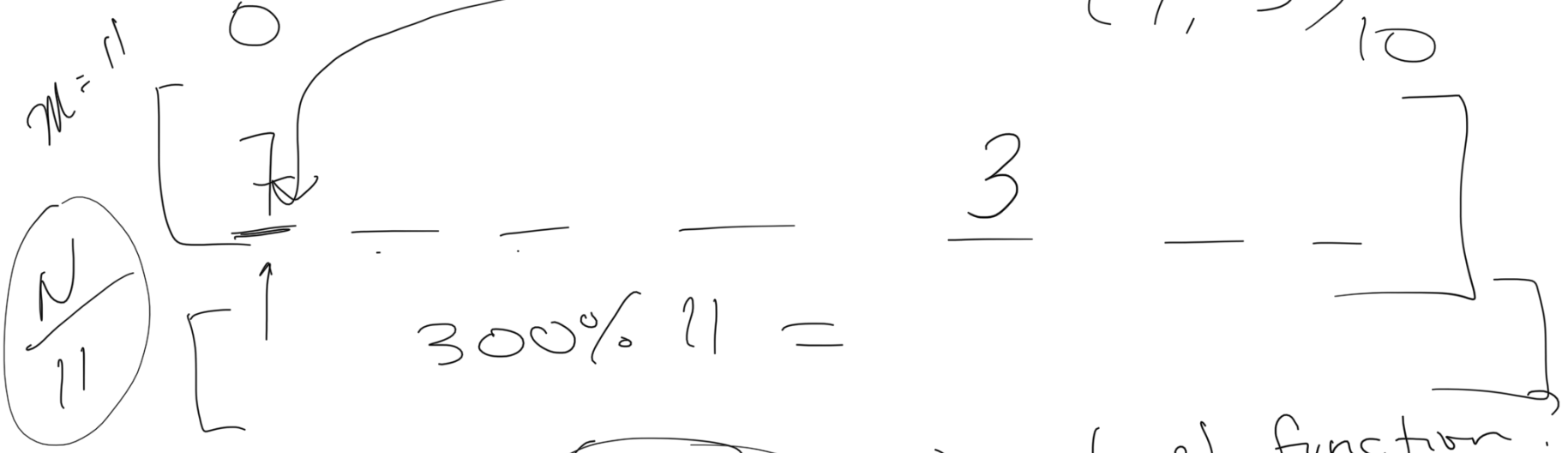
insert = O(1)
search : O(N)

Binsort

range of keys (0, 7)

(4, 3)

M = 11



"hello"
"ohell"

insert ("hello", A)

hash function:

hash ("hello") = "h" + "e" + "l" + "l" + "o" =

300

SHA256

hash(SSN + salt)

Ordered vs. Unordered Dictionary

Ordered Dictionary:
Keeps the keys in order

public class ST<Key extends Comparable<Key>, Value>		
	ST()	*create an ordered symbol table*
void	put(Key key, Value val)	*put key-value pair into the table (remove* key *from table if value is* null*)*
Value	get(Key key)	*value paired with* key *(*null *if* key *is absent)*
void	delete(Key key)	*remove* key *(and its value) from table*
boolean	contains(Key key)	*is there a value paired with* key*?*
boolean	isEmpty()	*is the table empty?*
int	size()	*number of key-value pairs*
Key	min()	*smallest key*
Key	max()	*largest key*
Key	floor(Key key)	*largest key less than or equal to* key
Key	ceiling(Key key)	*smallest key greater than or equal to* key
int	rank(Key key)	*number of keys less than* key
Key	select(int k)	*key of rank* k
void	deleteMin()	*delete smallest key*
void	deleteMax()	*delete largest key*
int	size(Key lo, Key hi)	*number of keys in* [lo..hi]
Iterable<Key>	keys(Key lo, Key hi)	*keys in* [lo..hi]*, in sorted order*
Iterable<Key>	keys()	*all keys in the table, in sorted order*

Picture from [1]

API for a generic ordered symbol table

References

[1] *Algorithms, Fourth Edition;* Robert Sedgewick and Kevin Wayne (and associated slides)