Searching III

Binary Search Trees

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]

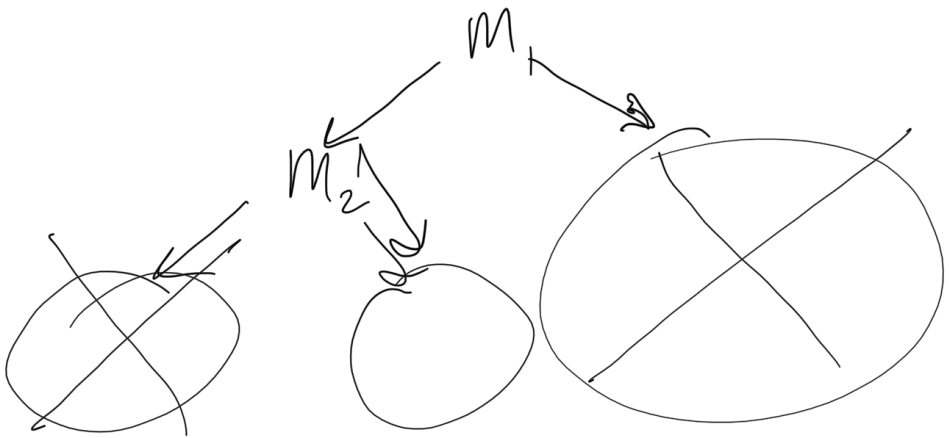
API for a generic ordered symbol table

Intuition for Binary Search Tree

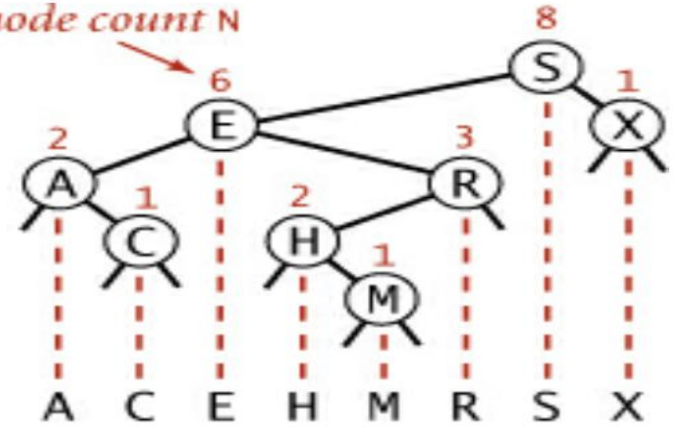
Combine the useful elements of linked lists with the useful elements of ordered arrays.

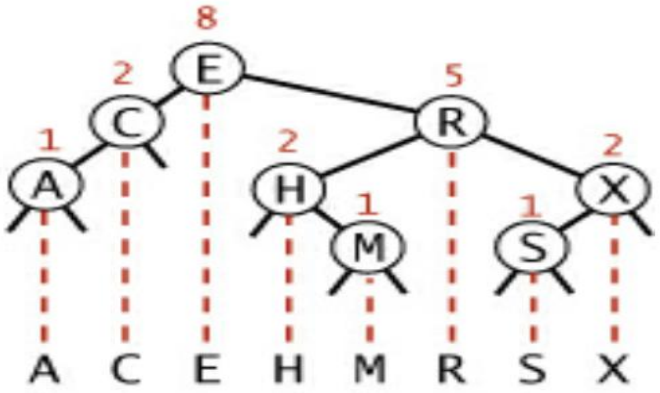
linked list : easy to grow (insert, delete)

ordered array : indexing — binary search



Binary Search Tree





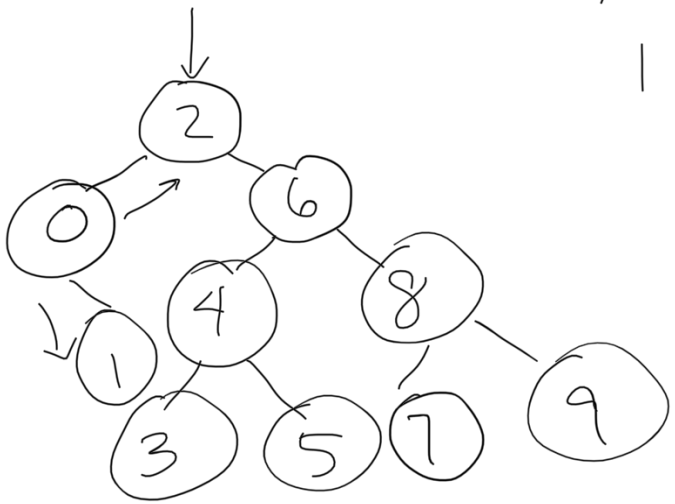
Two BSTs that represent the same set of keys

Picture from [1]

Same keys, different trees…

HOW DOES THIS HAPPEN?

Random order : 2, 6, 4, 0, 8, 5, 1, 7, 3, 9



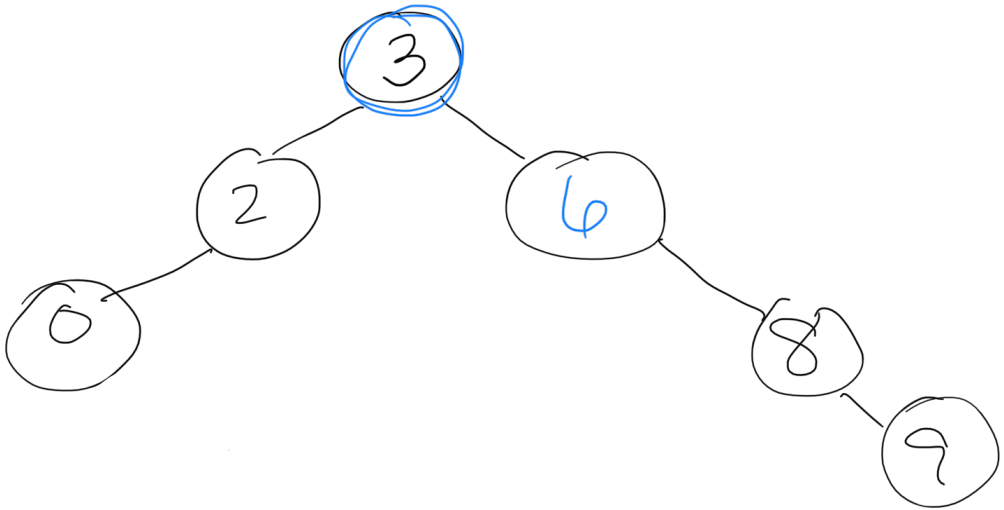
best case:

delete 1

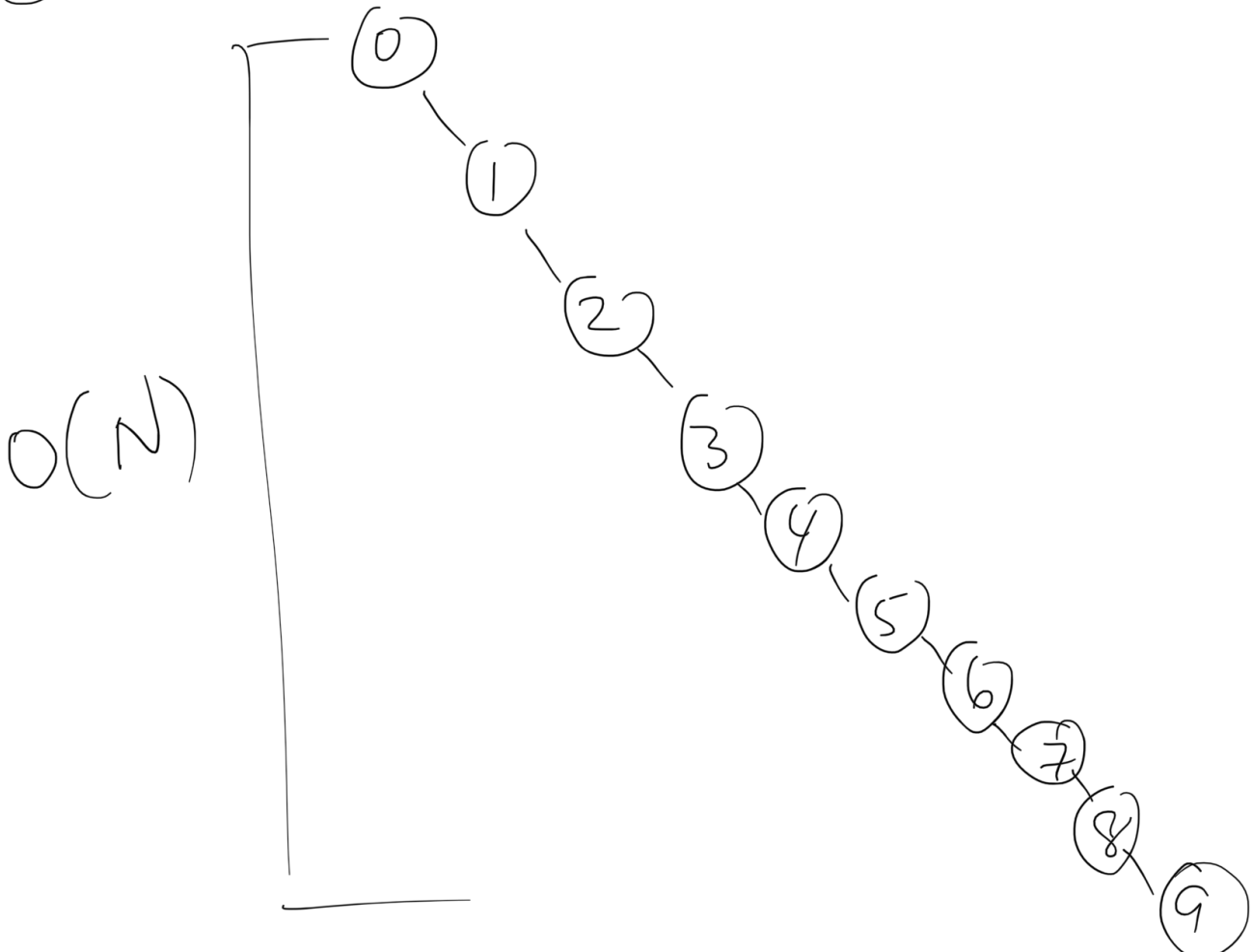
delete 4

delete 5

delete 7



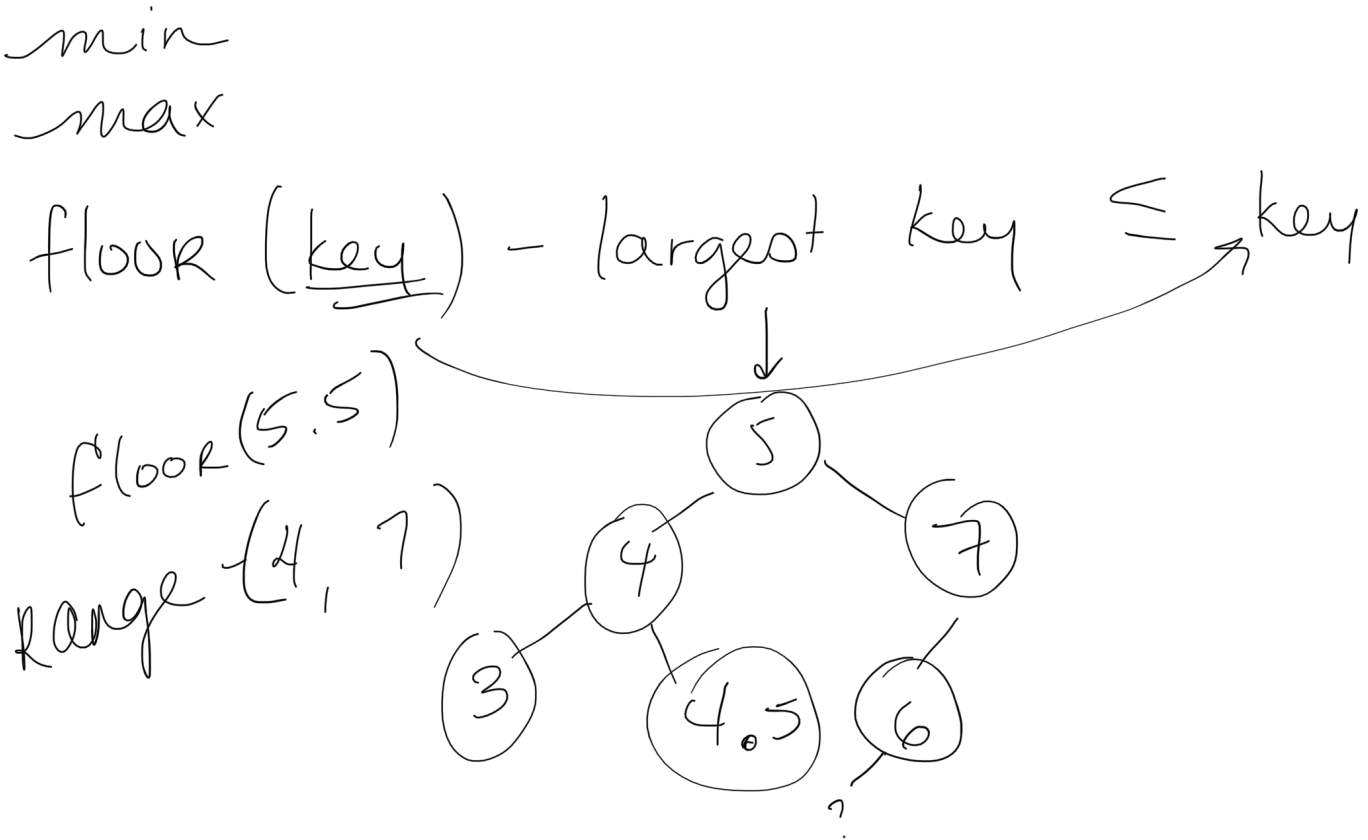
Worst case:



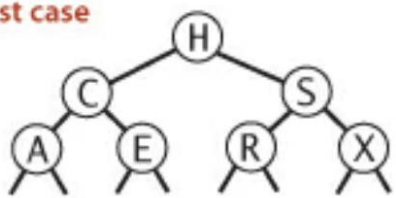
Main Operations

put -
get
delete

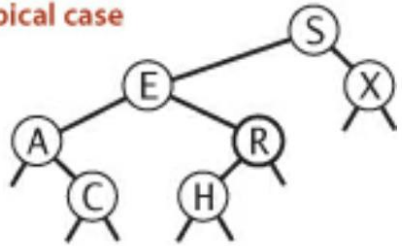
Other Operations



best case



typical case



worst case

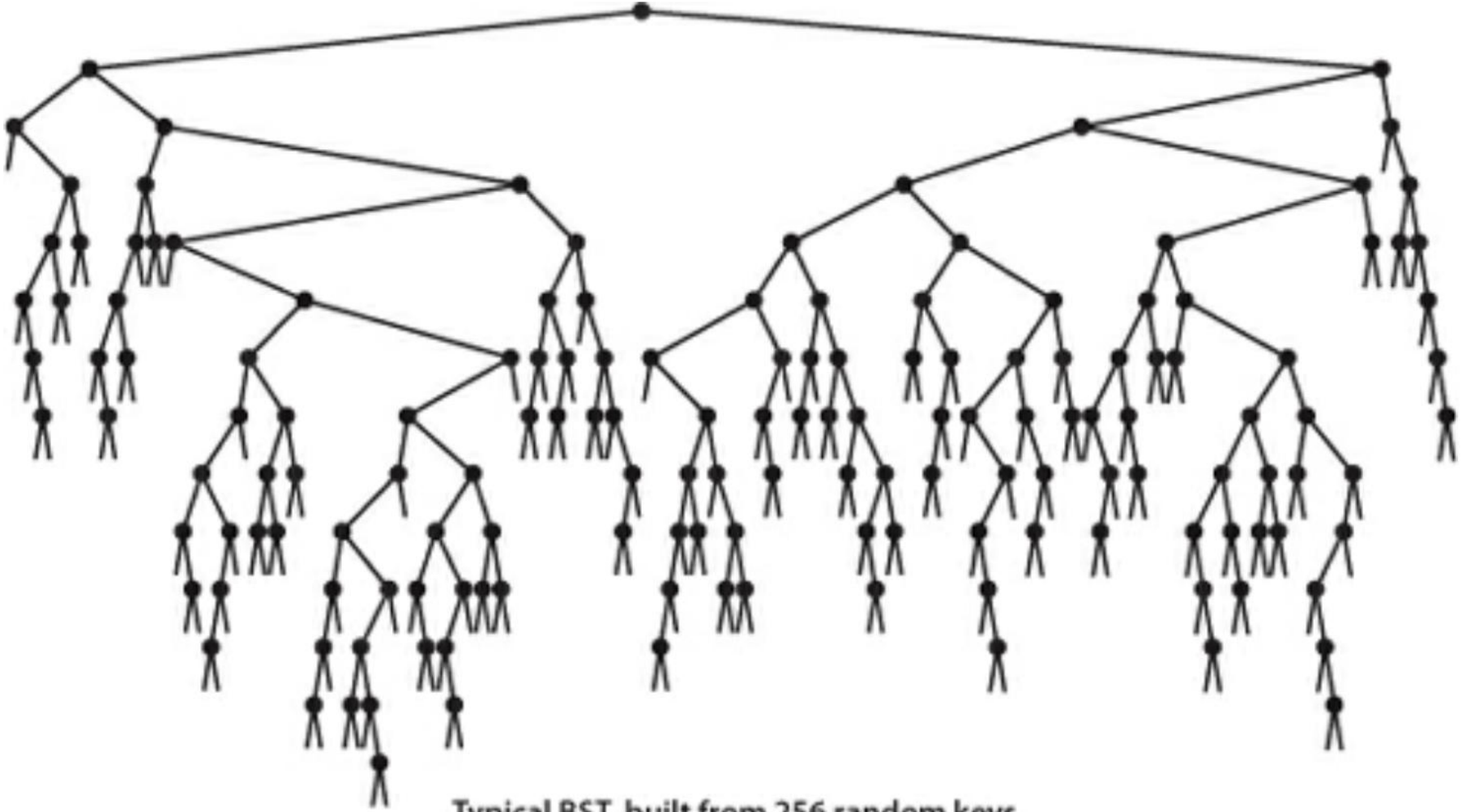
BST possibilities

algorithm (data structure)	worst-case cost (after N inserts) search	worst-case cost (after N inserts) insert	average-case cost (after N random inserts) search hit	average-case cost (after N random inserts) insert	efficiently support ordered operations?
sequential search (unordered linked list)	N	N	$N/2$	N	no
binary search (ordered array)	$\lg N$	N	$\lg N$	$N/2$	yes
binary tree search (BST)	N	N	$1.39 \lg N$	$1.39 \lg N$	yes

Cost summary for basic symbol-table implementations (updated)

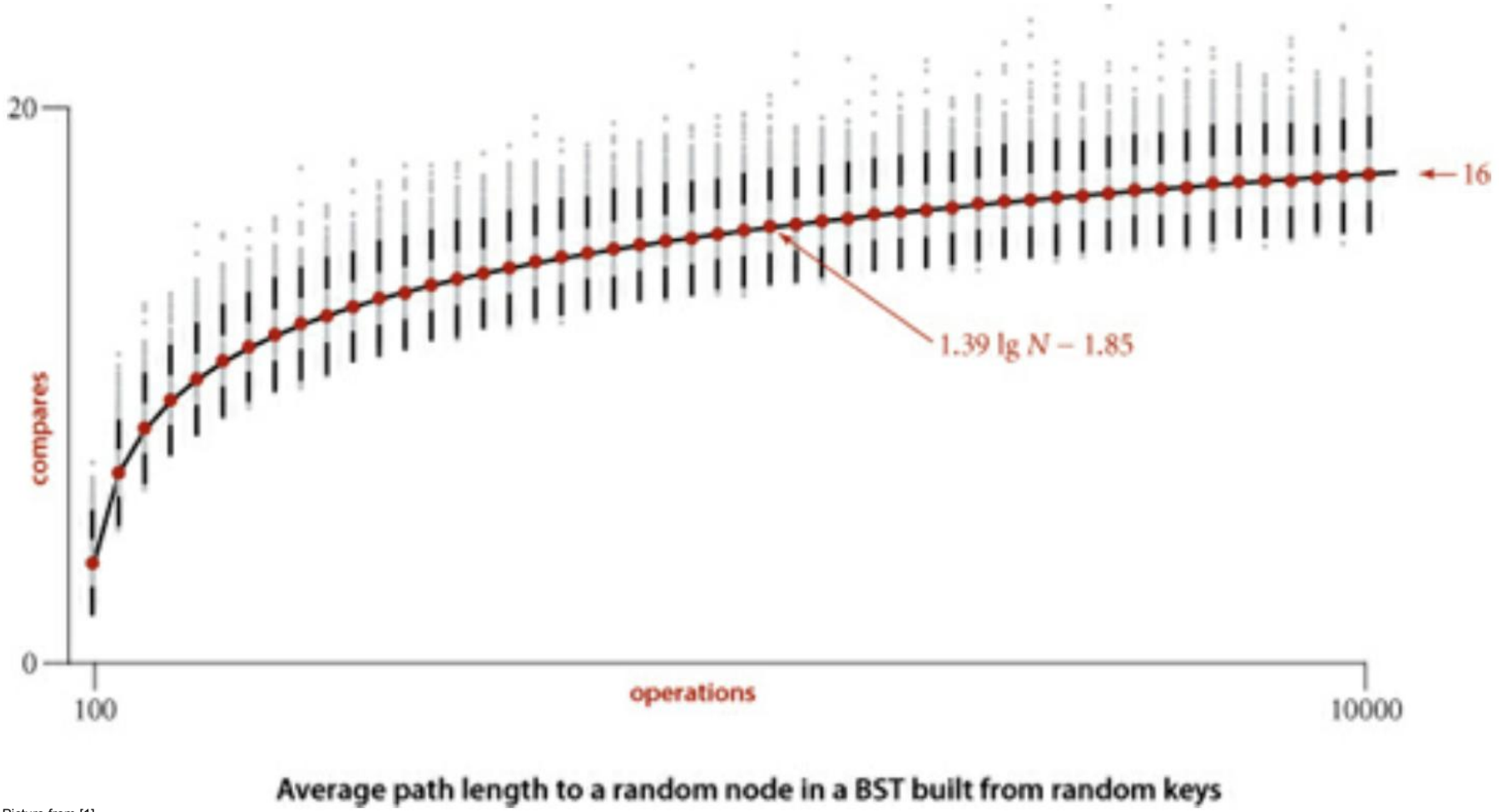
The good news is that the average case (with random data) is closer to the best case than the worst case!

Pictures from [1]



Typical BST, built from 256 random keys

Picture from [1]



Average path length to a random node in a BST built from random keys

Picture from [1]

How can we improve upon this?

References

[1] *Algorithms, Fourth Edition;* Robert Sedgewick and Kevin Wayne (and associated slides)
[2] Slides from https://www.cs.princeton.edu/~rs/talks/LLRB/RedBlack.pdf