HEAPS

David Kauchak
CS 62 – Fall 2026

1

Admin

Autocomplete assignment

2

Lambda expressions in Java

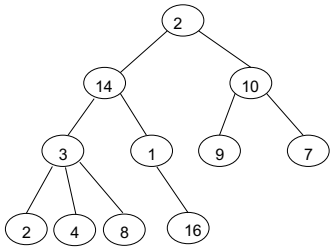
https://github.com/pomonaCS62-fa2026/LectureCode/tree/main/LambaExpressions

3

Trees

A set of nodes based on a parent-child relationship

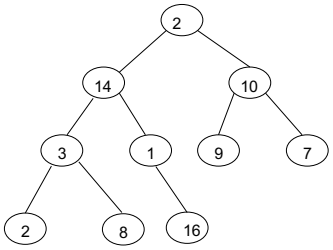
- Each node has one parent
- Root has no parent



4

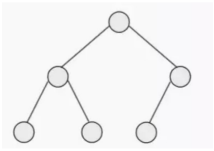
Binary tree

Each parent has at most 2 children

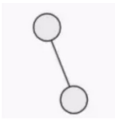


5

Full + Complete?



12

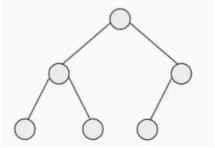


Full tree: a binary tree where every node has 0 or 2 children

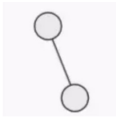
Complete: All levels except the last are completely filled and all nodes on the last level are on the left

6

Full + Complete?



12



Complete Full + Complete Neither

7

Implementing a binary tree

```
public class BinaryTree<E> {
    private Node root;

    private class Node {
        private E value;
        private Node left;
        private Node right;

        public Node(Node left, Node right, E value) {
            this.left = left;
            this.right = right;
            this.value = value;
        }
    }
```

8

Recursive data structure!

```
public class BinaryTree<E> {
    private Node root;

    private class Node {
        private E value;
        private Node left;
        private Node right;

        public Node(Node left, Node right, E value) {
            this.left = left;
            this.right = right;
            this.value = value;
        }
    }
```

```
public class LinkedList<E> {
    private Node head;

    private class Node {
        private E value;
        private Node next;

        public Node(Node next, E value) {
            this.next = next;
            this.value = value;
        }
    }
```

9

Binary heap

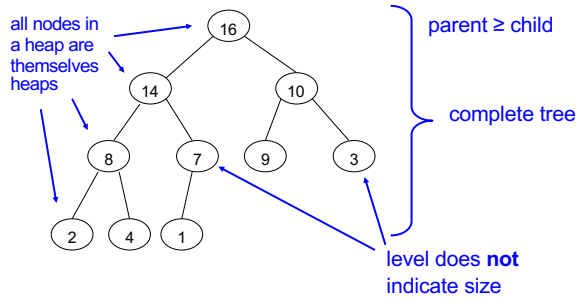
A binary tree where the value of a parent is greater than or equal to the value of its children

Additional restriction: the tree must be **complete**!

Max heap vs. min heap

10

Binary heap - references



11

Binary heap - array

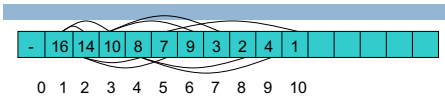
$\textsc{Parent}(i)$
  return $\lfloor i/2 \rfloor$

$\textsc{Left}(i)$
  return $2i$

$\textsc{Right}(i)$
  return $2i + 1$

12

Binary heap - array



Note 0 is empty!

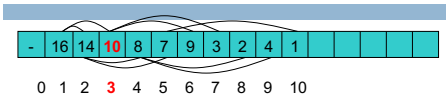
PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

13

Binary heap - array



Left child of A[3]?

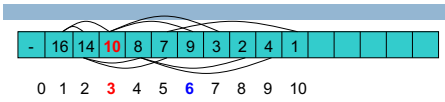
PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

14

Binary heap - array



Left child of A[3]?

2*3 = 6

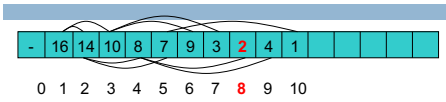
PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

15

Binary heap - array



Parent of A[8]?

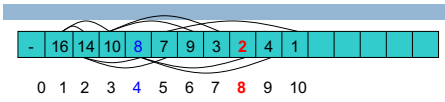
PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

16

Binary heap - array



PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

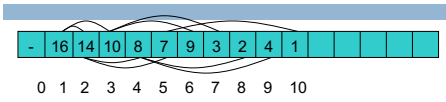
RIGHT(i)
  return $2i + 1$

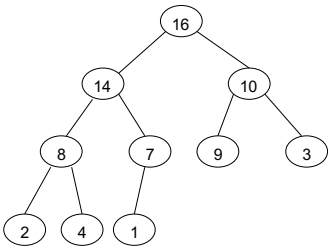
Parent of A[8]?

$\lfloor 8/2 \rfloor = 4$

17

Binary heap - array





18

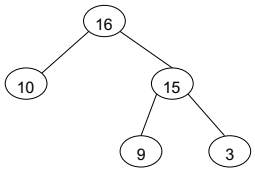
Identify the valid heaps: worksheet 1

[-, 15, 12, 3, 11, 10, 2, 1, 7, 8]

PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$





[-, 20, 18, 10, 17, 16, 15, 9, 14, 13]

19

Heaps in code

```
public class Heap <E extends Comparable<E>> {
    private ArrayList<E> data = new ArrayList<E>();
```

https://github.com/pomonaCS62-fa2026/LectureCode/blob/main/Trees/trees/Heap.java

20

Heaps in code

```
private int left(int i) {
    return 2*i;
}

private int right(int i) {
    return 2*i + 1;
}

private int parent(int i) {
    return i/2;
}

private void swap(int f, int s) {
    E temp = data.get(f);
    data.set(f,  data.get(s));
    data.set(s, temp);
}
```

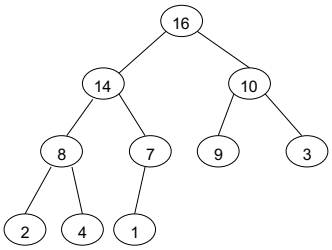
LEFT(i)
    return $2i$

RIGHT(i)
    return $2i + 1$

PARENT(i)
    return $\lfloor i/2 \rfloor$

21

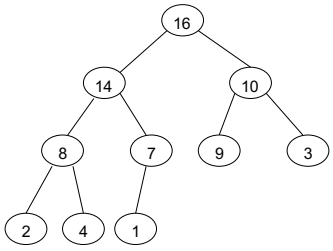
What are heaps good for?



22

What are heaps good for?

What's the largest value in this heap?

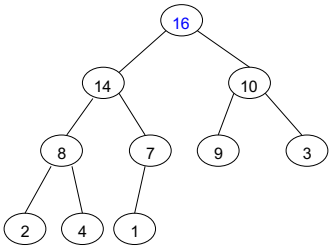


23

What are heaps good for?

What's the largest value in this heap?

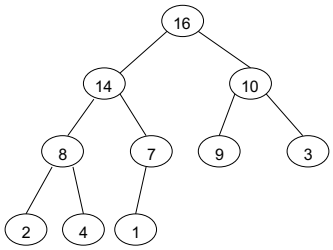
Heaps are good at min/max operations (depending on min/max ordering)!



24

What are heaps good for?

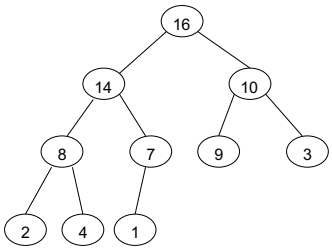
What's the 2nd largest value? The 3rd? The 4th?



25

ExtractMax

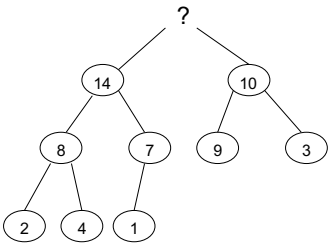
Return and remove the largest element in the set. The rest of the data should stay as a heap



26

ExtractMax

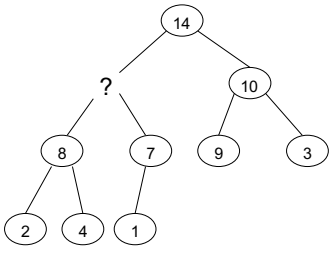
Return and remove the largest element in the set. The rest of the data should stay as a heap



27

ExtractMax

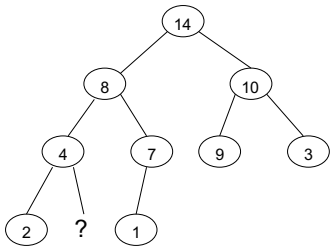
Return and remove the largest element in the set. The rest of the data should stay as a heap



28

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

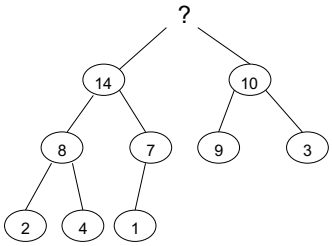


Now what? Is this a heap?

29

ExtractMax

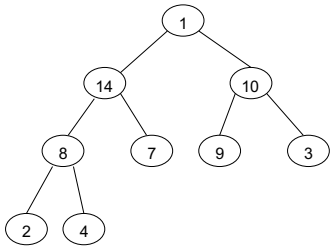
Return and remove the largest element in the set. The rest of the data should stay as a heap



30

ExtractMax

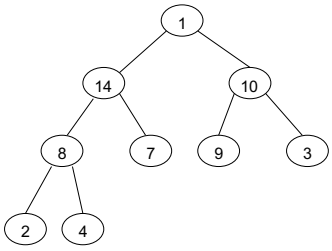
Return and remove the largest element in the set. The rest of the data should stay as a heap



31

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

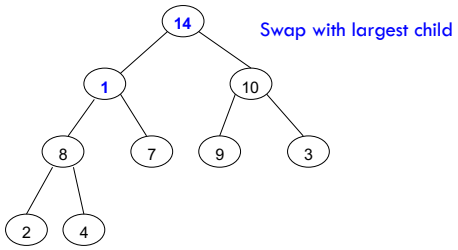


Now what?

32

ExtractMax

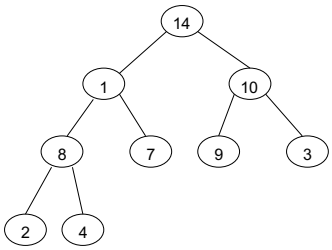
Return and remove the largest element in the set. The rest of the data should stay as a heap



33

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

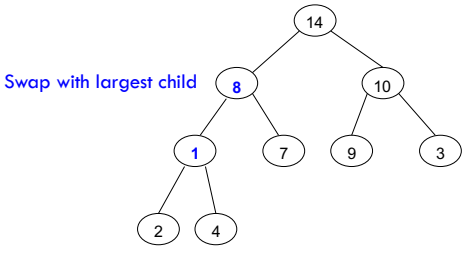


Now what?

34

ExtractMax

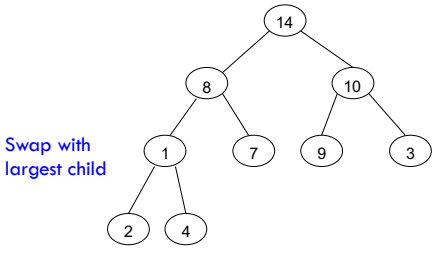
Return and remove the largest element in the set. The rest of the data should stay as a heap



35

ExtractMax

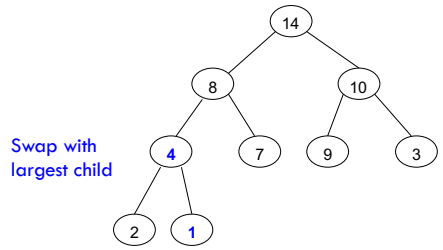
Return and remove the largest element in the set. The rest of the data should stay as a heap



36

ExtractMax

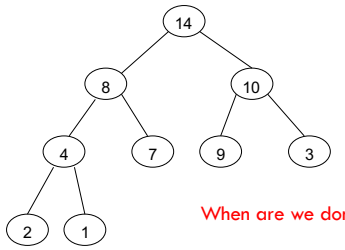
Return and remove the largest element in the set. The rest of the data should stay as a heap



37

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

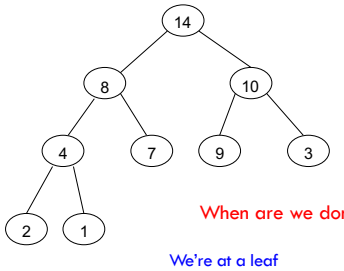


When are we done?

38

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap



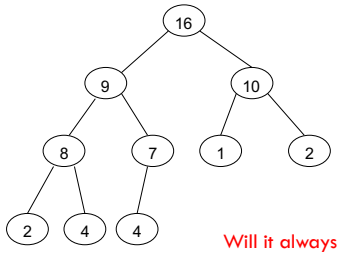
When are we done?

We're at a leaf

39

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

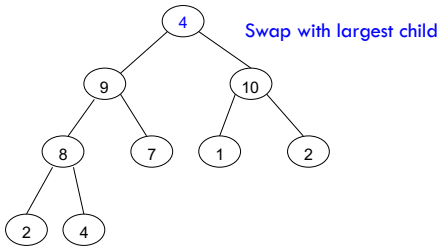


Will it always be a leaf?

40

ExtractMax

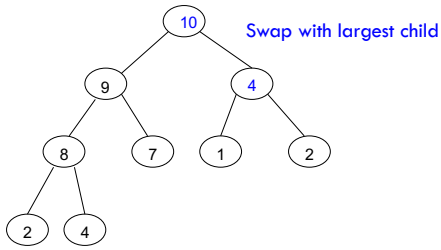
Return and remove the largest element in the set. The rest of the data should stay as a heap



41

ExtractMax

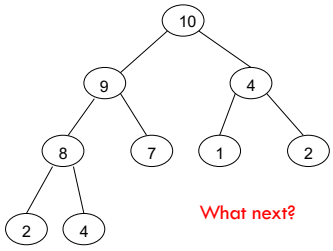
Return and remove the largest element in the set. The rest of the data should stay as a heap



42

ExtractMax

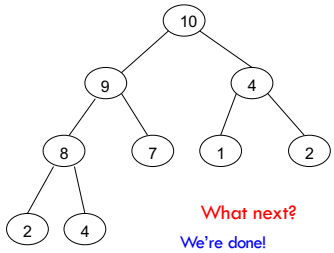
Return and remove the largest element in the set. The rest of the data should stay as a heap



43

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap



44

ExtracMax in code

```
private int maxChildIndex(int i) {
    if( right(i) < data.size() ) {
        E leftVal = data.get(left(i));
        E rightVal = data.get(right(i));

        return leftVal.compareTo(rightVal) > 0 ?
                left(i) : right(i);
    } else {
        return left(i);
    }
}
```

What does this function do? Any assumptions/pre-conditions?

45

sink/heapify/demote

```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

46

sink runtime

```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

What is the worst case runtime?

47

sink runtime

```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

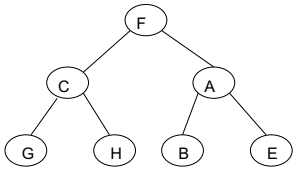
        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

What is the worst case runtime? O(height of tree)

48

Nodes in a binary tree



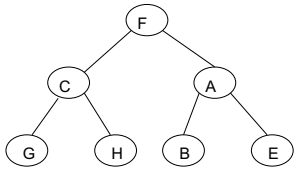
$$\log(n+1) - 1 \leq h \leq n - 1$$

Height is somewhere between log(n) of nodes and n nodes

49

Nodes in a binary tree

When does this case happen?



$$\log(n+1) - 1 \leq h \leq n - 1$$

Height is somewhere between log(n) of nodes and n nodes

50

sink runtime

```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

What is the worst case runtime? O(log n)

51

ExtractMax

```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

52

ExtractMax

```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

What are these lines doing?

53

ExtractMax

```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

- largest value is at the root (index 1)
- move the last value to the root
- remove the last item
- call sink on the root
- return the largest value

54

ExtractMax

```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

What is the worst-case runtime?

55

ExtractMax

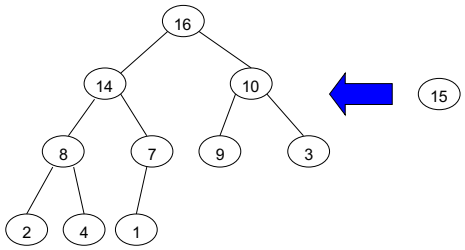
```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

What is the worst-case runtime? O(log n)

56

Insert

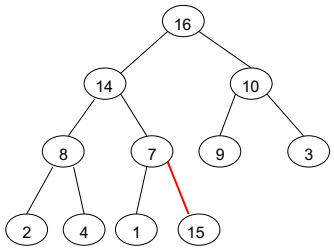
How do we insert a value into a heap?



57

Insert

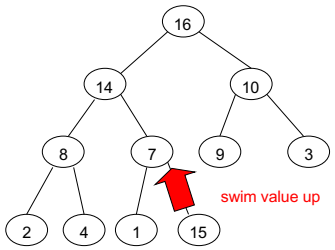
Insert the value at the end of the array (or as the rightmost leaf)



58

Insert

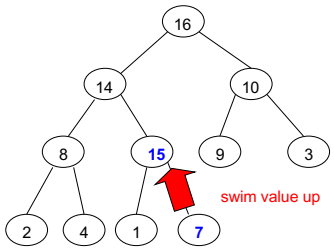
Swap the value up until it's in the right place



59

Insert

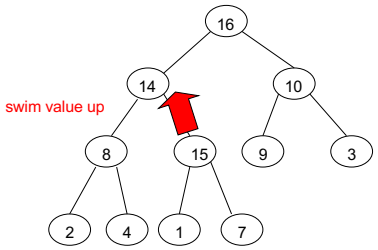
Swap the value up until it's in the right place



60

Insert

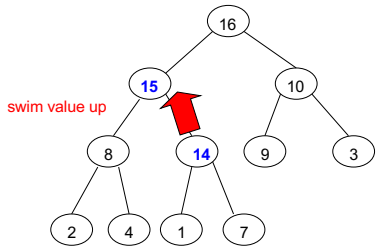
Swap the value up until it's in the right place



61

Insert

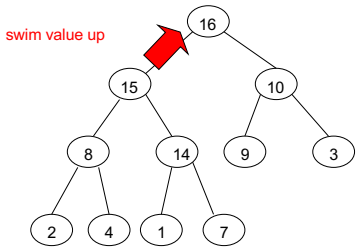
Swap the value up until it's in the right place



62

Insert

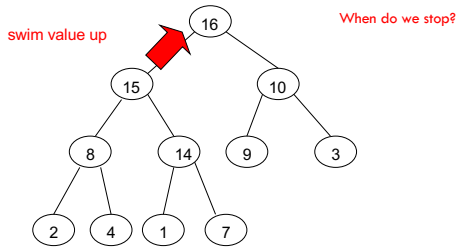
Swap the value up until it's in the right place



63

Insert

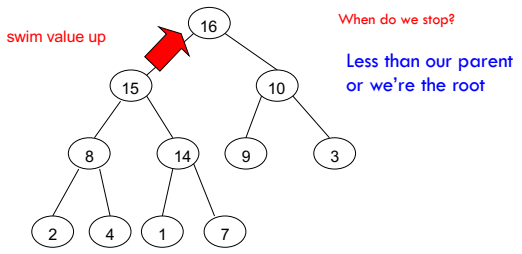
Swap the value up until it's in the right place



64

Insert

Swap the value up until it's in the right place



65

swim/percolate up

```
private void swim(int i) {
    if( i > 1 ) {
        E value = data.get(i);
        E parentVal = data.get(parent(i));

        if( value.compareTo(parentVal) > 0 ) {
            swap(i, parent(i));
            swim(parent(i));
        }
    }
}
```

66

swim/percolate up

```
private void swim(int i) {
    if( i > 1 ) {
        E value = data.get(i);
        E parentVal = data.get(parent(i));

        if( value.compareTo(parentVal) > 0 ) {
            swap(i, parent(i));
            swim(parent(i));
        }
    }
}
```

What's the worst case runtime?

67

swim/percolate up

```
private void swim(int i) {
    if( i > 1 ) {
        E value = data.get(i);
        E parentVal = data.get(parent(i));

        if( value.compareTo(parentVal) > 0 ) {
            swap(i, parent(i));
            swim(parent(i));
        }
    }
}
```

What's the worst case runtime? O(height of tree) = O(log n)

68

insert

```
public void insert(E val) {
    data.add(val);
    swim(data.size()-1);
}
```

69

insert

```
public void insert(E val) {
    data.add(val);
    swim(data.size()-1);
}
```

Worst-case runtime: O(log n)

70

Heaps summarized

Very good at extracting min/max (depending on heap ordering)

	best	worst	average
max			
extractMax		O(log n)	
insert		O(log n)	
change node		O(log n)	

71

Heaps summarized

Very good at extracting min/max (depending on heap ordering)

	best	worst	average
max	O(1)	O(1)	O(1)
extractMax	O(1)	O(log n)	O(log n)
insert	O(1)	O(log n)	O(log n)
change node	O(1)	O(log n)	O(log n)

72