# 二分木ヒープ
# (Binary Heap)

# 二分木ヒープとは

- 集合・リストから「最小な」要素を取り出す
  - 二分木ヒープは、そのための標準的データ構造

- 二分木ヒープを保存するデータ構造
- 二分木ヒープの操作のメソッド

- 対象となるデータクラス
  - 識別のためのlabelフィールド
  - 値を保持するvalueフィールド

# 二分木ヒープとは、どういう二分木か

- ある頂点の要素$p$のvalueは、その子$c$の要素のvalueより大きくない

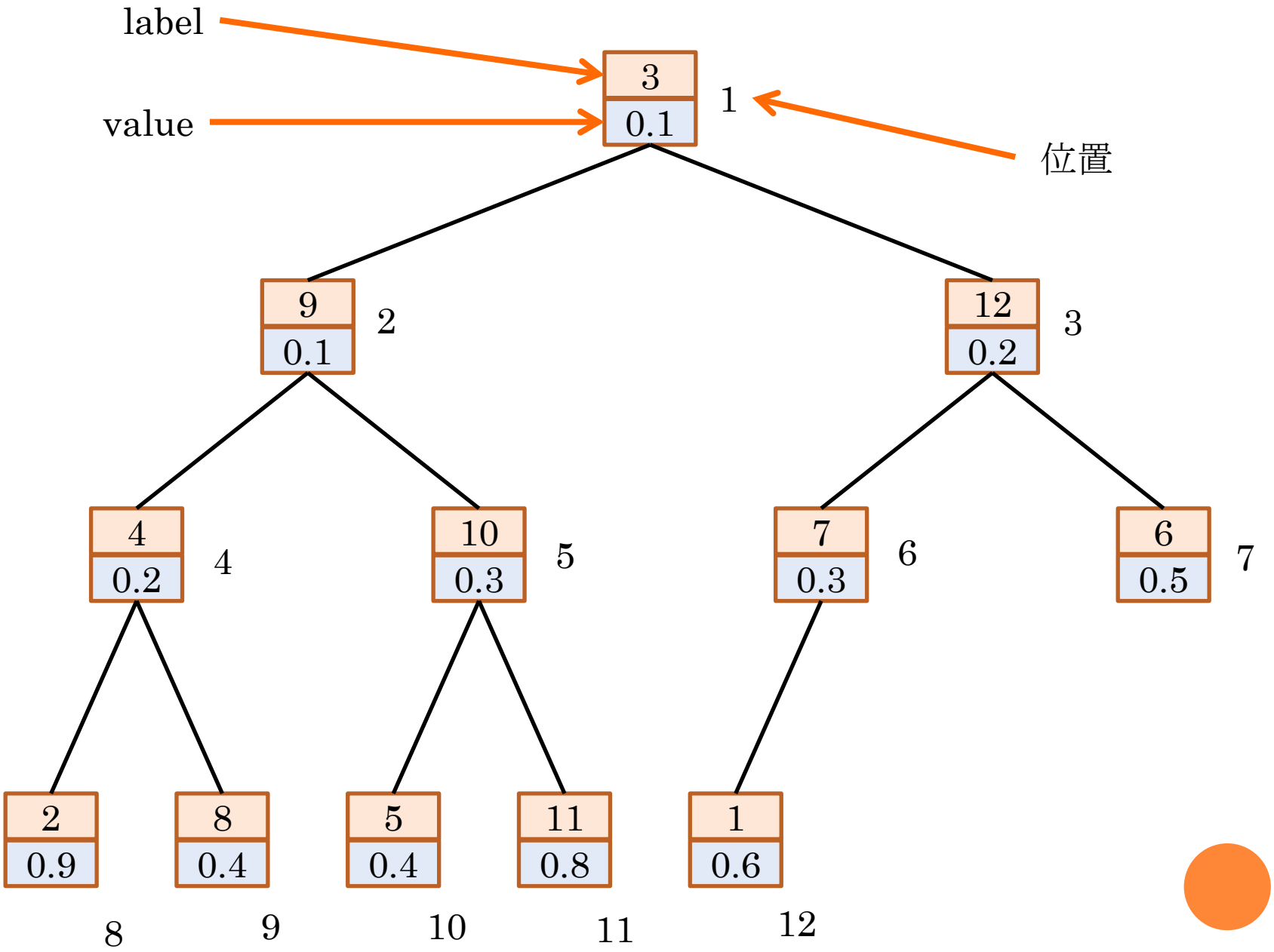
$$p.value \leq c.value$$

  - 半順序木になっている

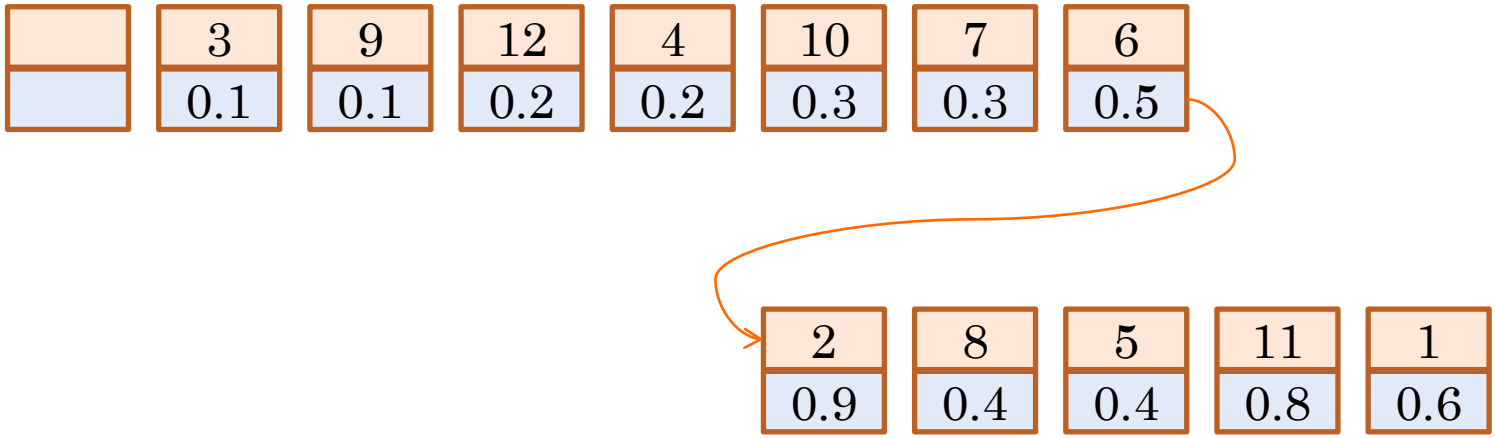
- 完全二分木である
  - 最下層以外の第$k$層には、$2^{k-1}$個の頂点がある。
  - 最下層は、左から詰めて頂点がある。

# 半順序集合(PARTIALLY ORDERED SET)または順序集合(ORDERED SET)

- 集合$P$とその上の関係$\leq$に以下の法則が常になりたつとき、Pを半順序集合と呼ぶ。ここで、$a,b,c \in P$となる任意の要素とする。
  - 反射律 (reflectivity) $a \leq a$
  - 推移律 (transitivity) $(a \leq b) \wedge (b \leq c) \Rightarrow a \leq c$
  - 反対照律 (antisymmetry) $(a \leq b) \wedge (b \leq a) \Rightarrow a = b$
- 任意の元の組$a,b \in P$に対して$a \leq b$または$b \leq a$の何れかが必ず成り立つとき、$P$を全順序集合(totally ordered set)と呼ぶ。


label
value
3
0.1
1
位置
9
0.1
2
12
0.2
3
4
0.2
4
10
0.3
5
7
0.3
6
6
0.5
7
2
0.9
8
0.4
5
0.4
11
0.8
1
0.6
8
9
10
11
12

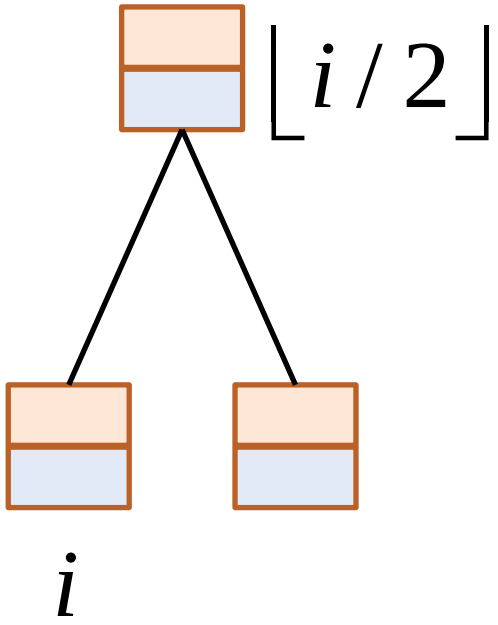
# データの保持形式

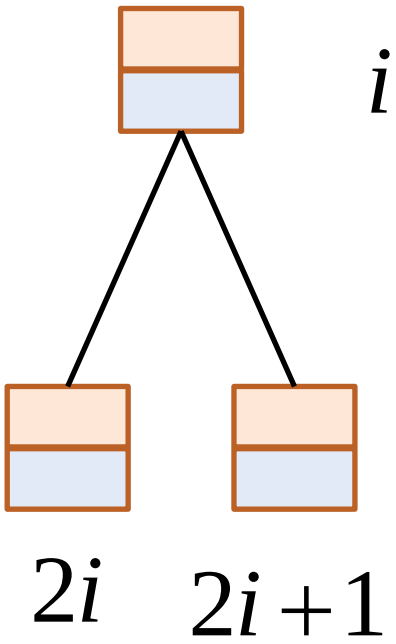



- リストで保持
  - 0番：空（リストのインデクスが1から始まるプログラミング言語では不要）
  - 1番：根の要素
  - $2^k$番：第$k$層の左端の要素

# 親子の番号の関係

- 親の番号
- 子の番号



# 要素の追加

- リストの終端に要素を追加する
  - 木の最下層の一番右に追加、または新たな層を作って、その左端に追加

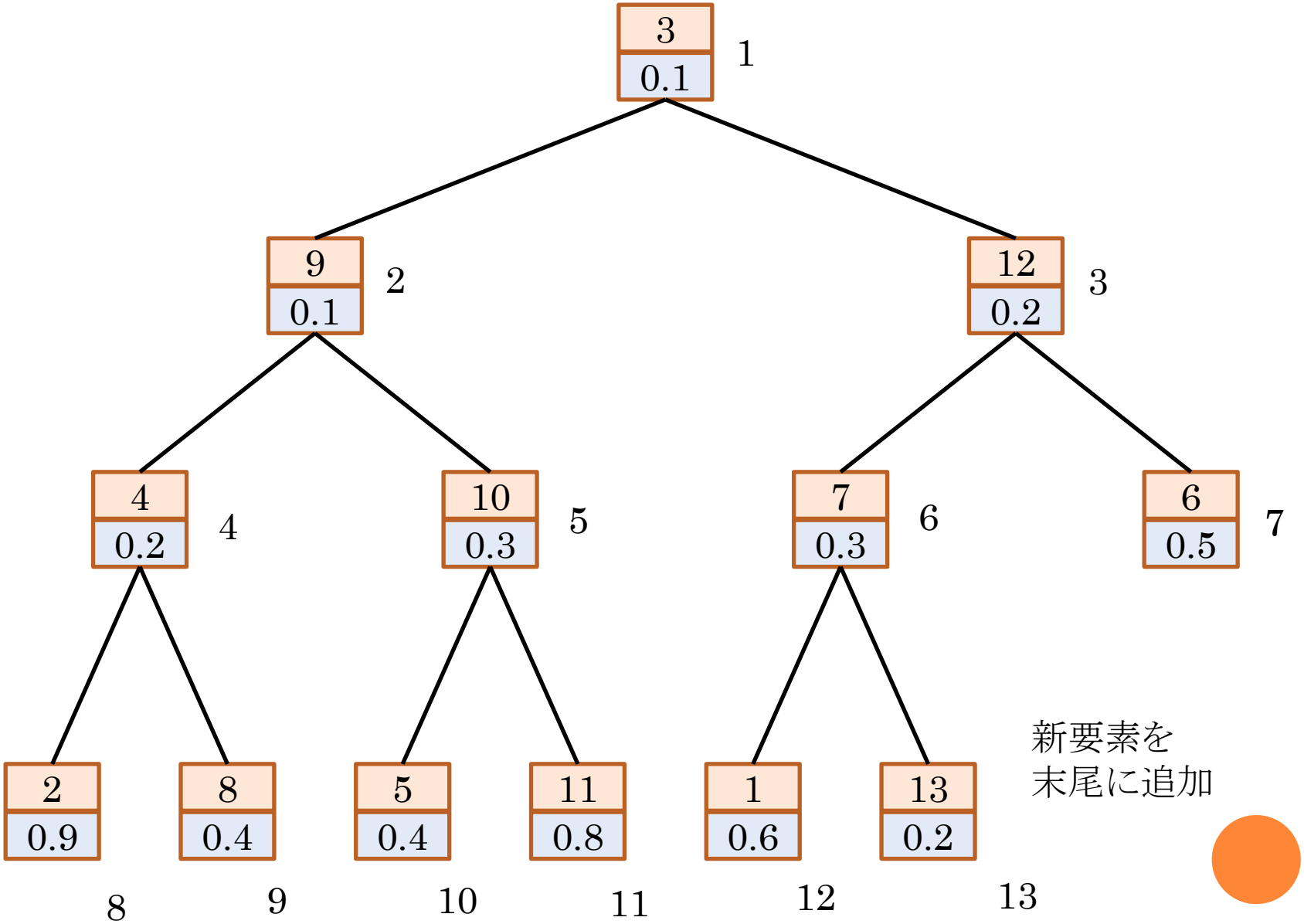
```
void add(O o){
        int  n = |L|;
        L.append(o);
        n++;
        shiftUp(n);//要素を正しい位置へ移動
}
```

# 要素の追加:シフトアップ

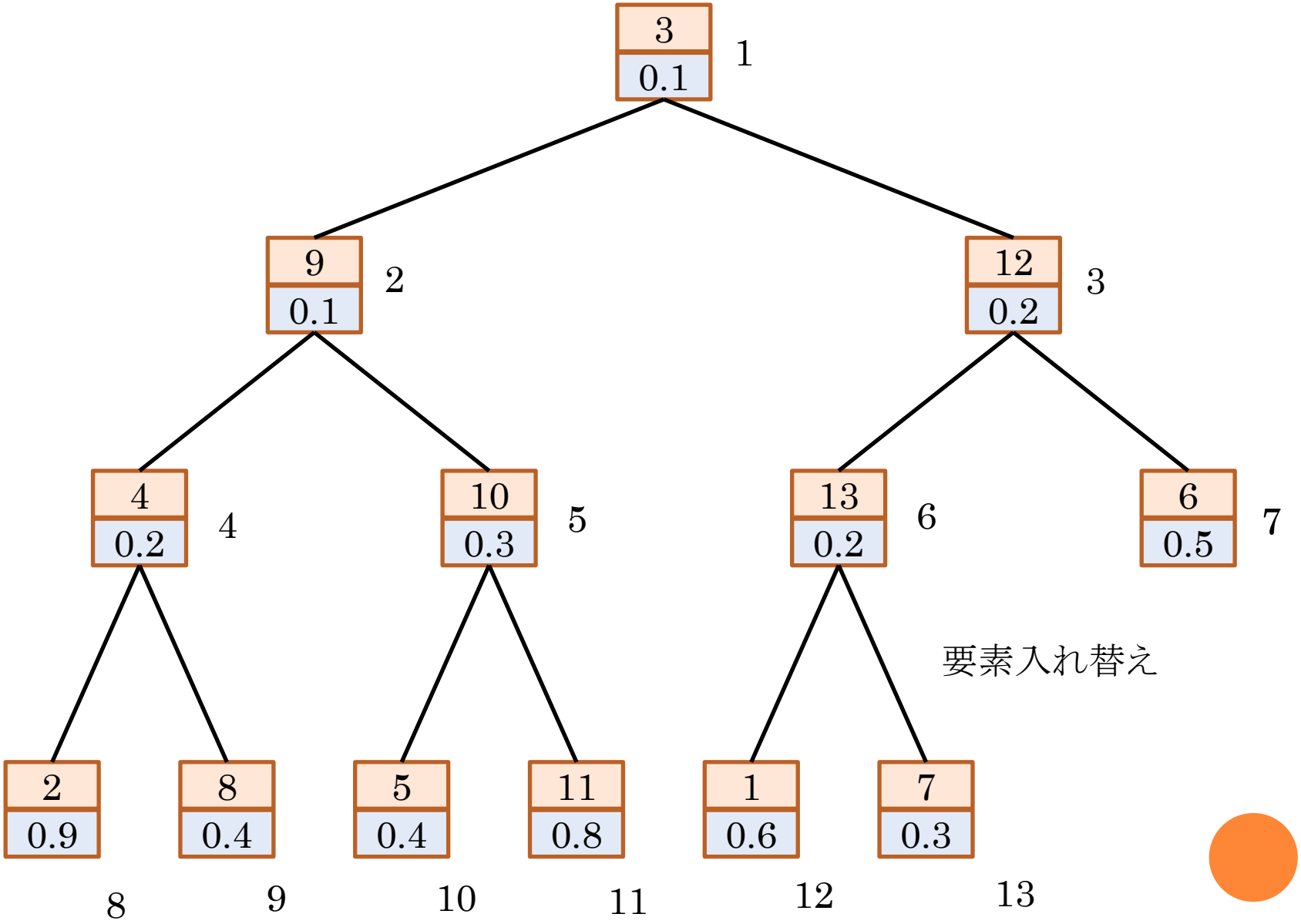
- 追加した新要素を正しい位置へ移動
- 位置$k$の要素が、親の位置 $\lfloor k/2 \rfloor$ の要素よりも小さいならば、二つの要素を入れ替える

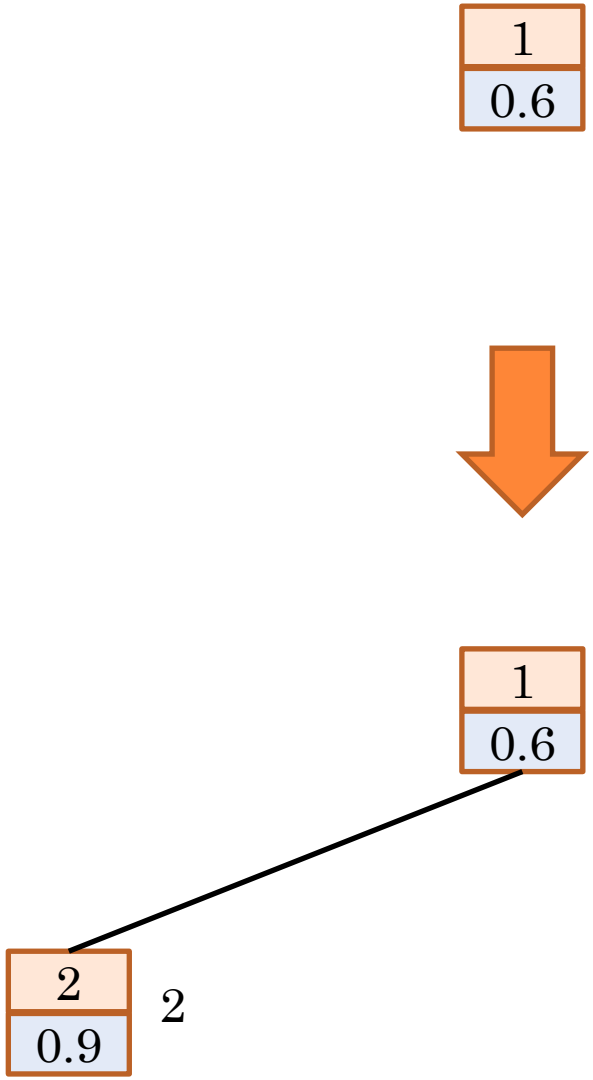
```
void shiftUp(int k){
    if(k > 1 && isLess(k, ⌊k/2⌋)){
        swap(k, ⌊k/2⌋);
        k = ⌊k/2⌋;
        shiftUp(k);
    }
}
```

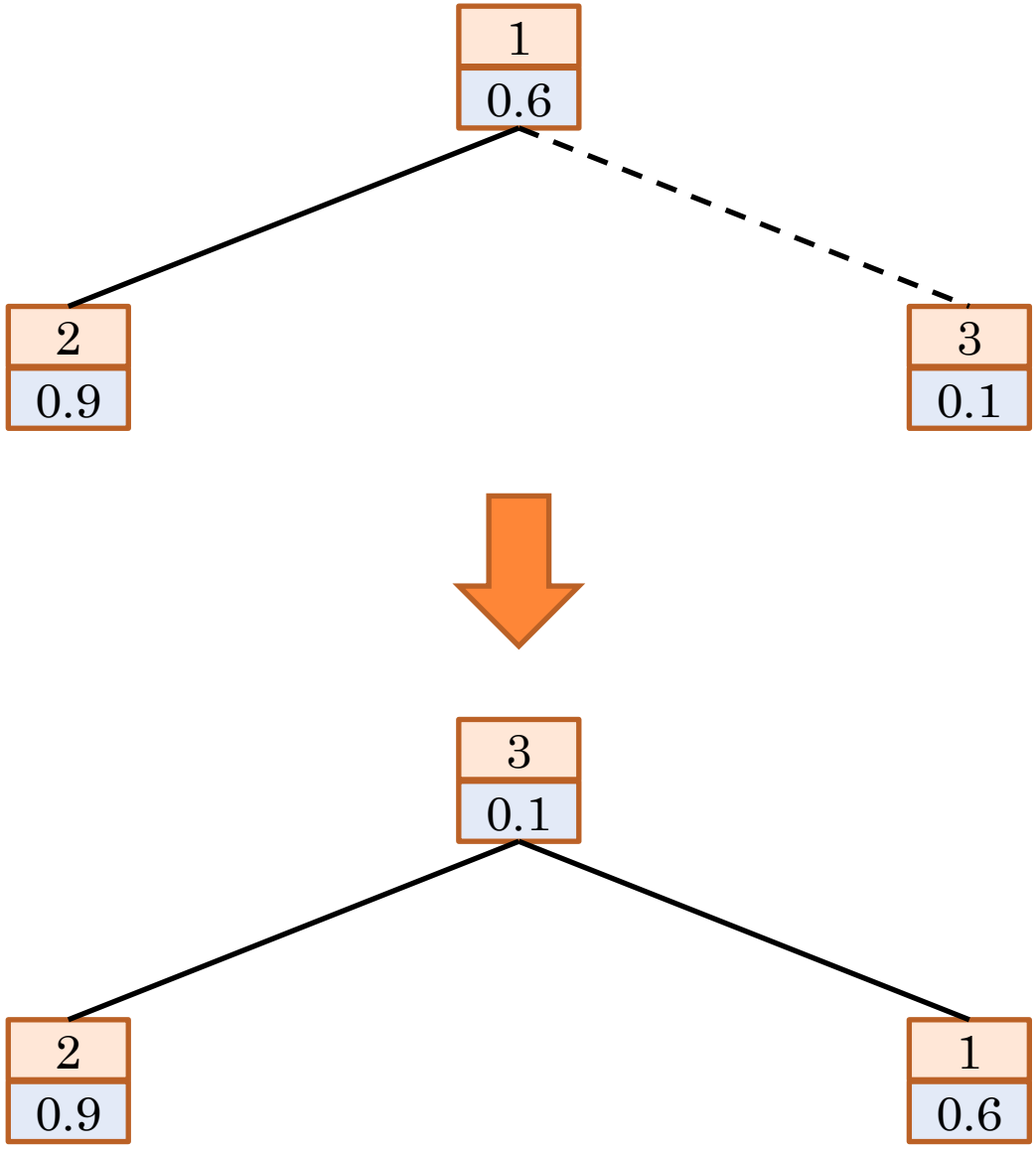
- isLess(i,j)
  - $o_i.value < o_j.value$ のとき真
- swap(i,j):要素入れ替え

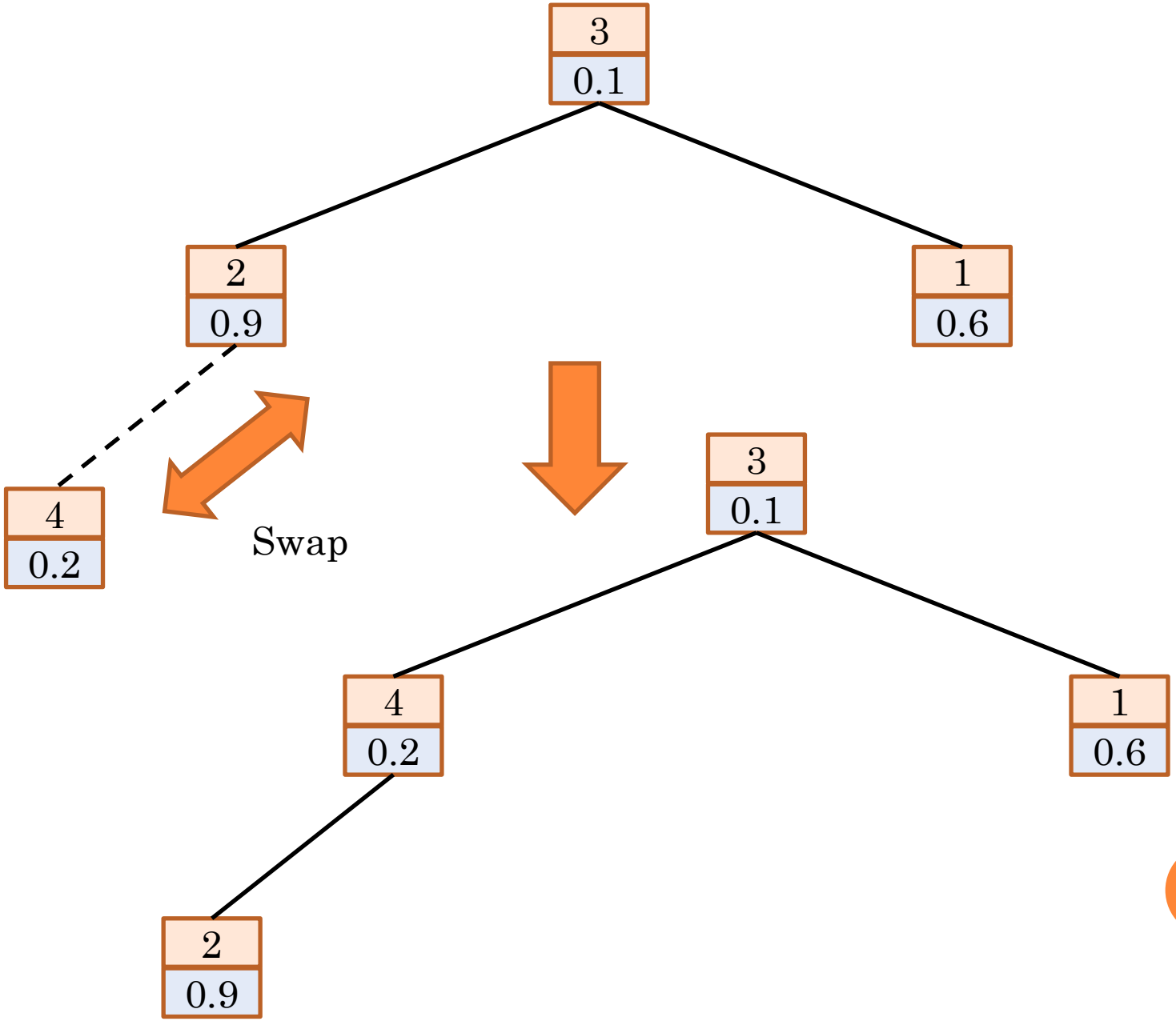

3
0.1
1
9
0.1
2
12
0.2
3
4
0.2
4
10
0.3
5
7
0.3
6
6
0.5
7
2
0.9
8
0.4
5
0.4
11
0.8
1
0.6
13
0.2
8
9
10
11
12
13

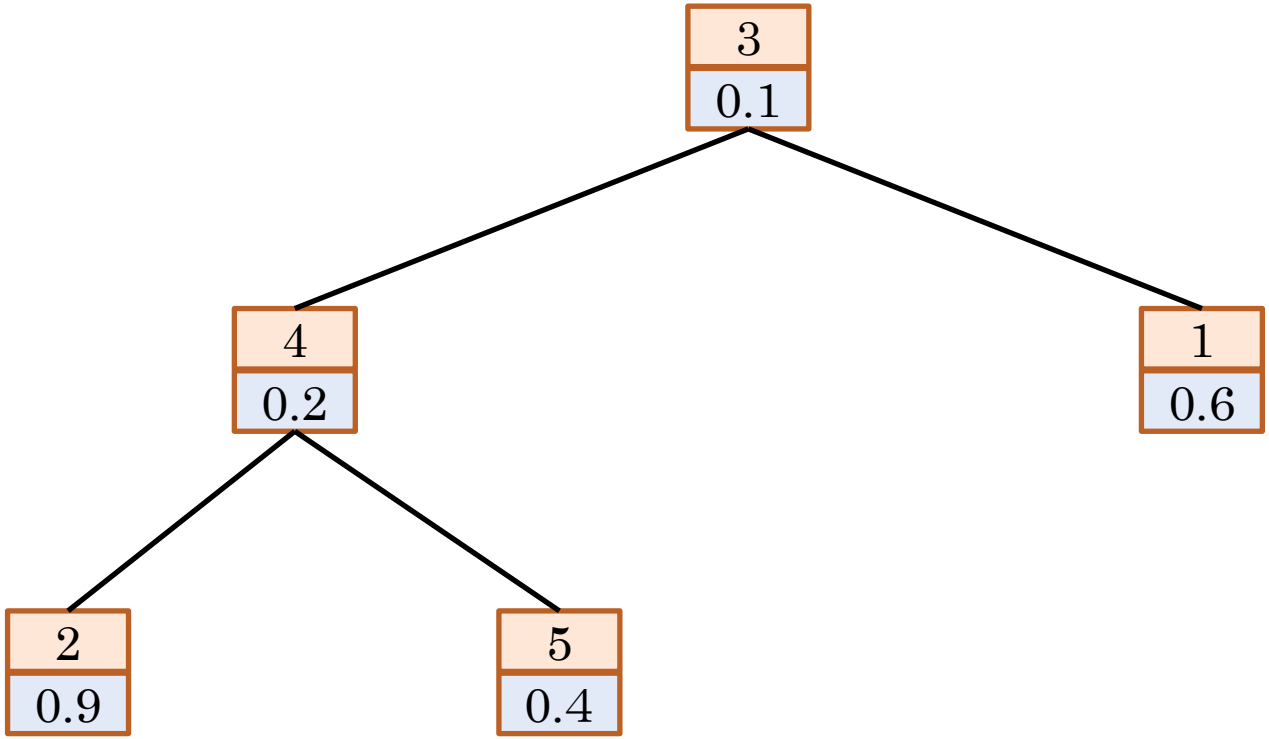
新要素を
末尾に追加

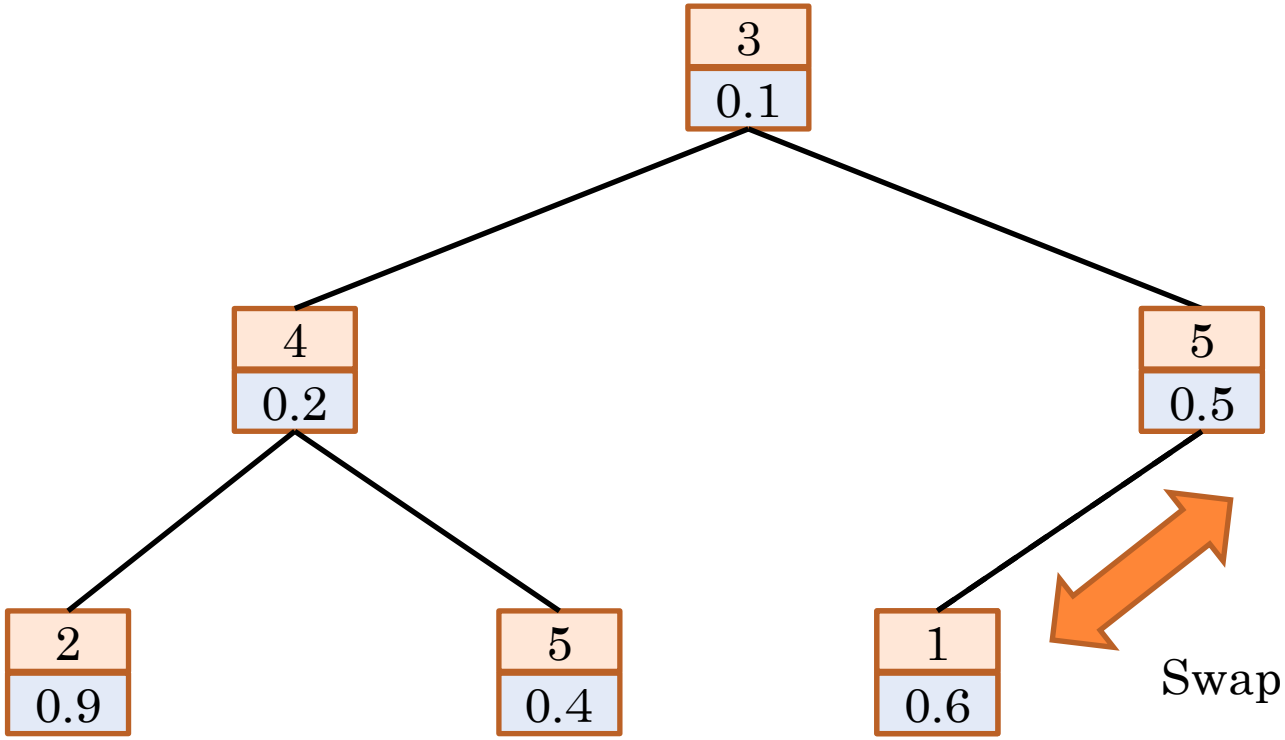

3
0.1
1
9
0.1
2
12
0.2
3
4
0.2
4
10
0.3
5
13
0.2
6
6
0.5
7
要素入れ替え
2
0.9
8
0.4
5
0.4
11
0.8
1
0.6
7
0.3
8
9
10
11
12
13

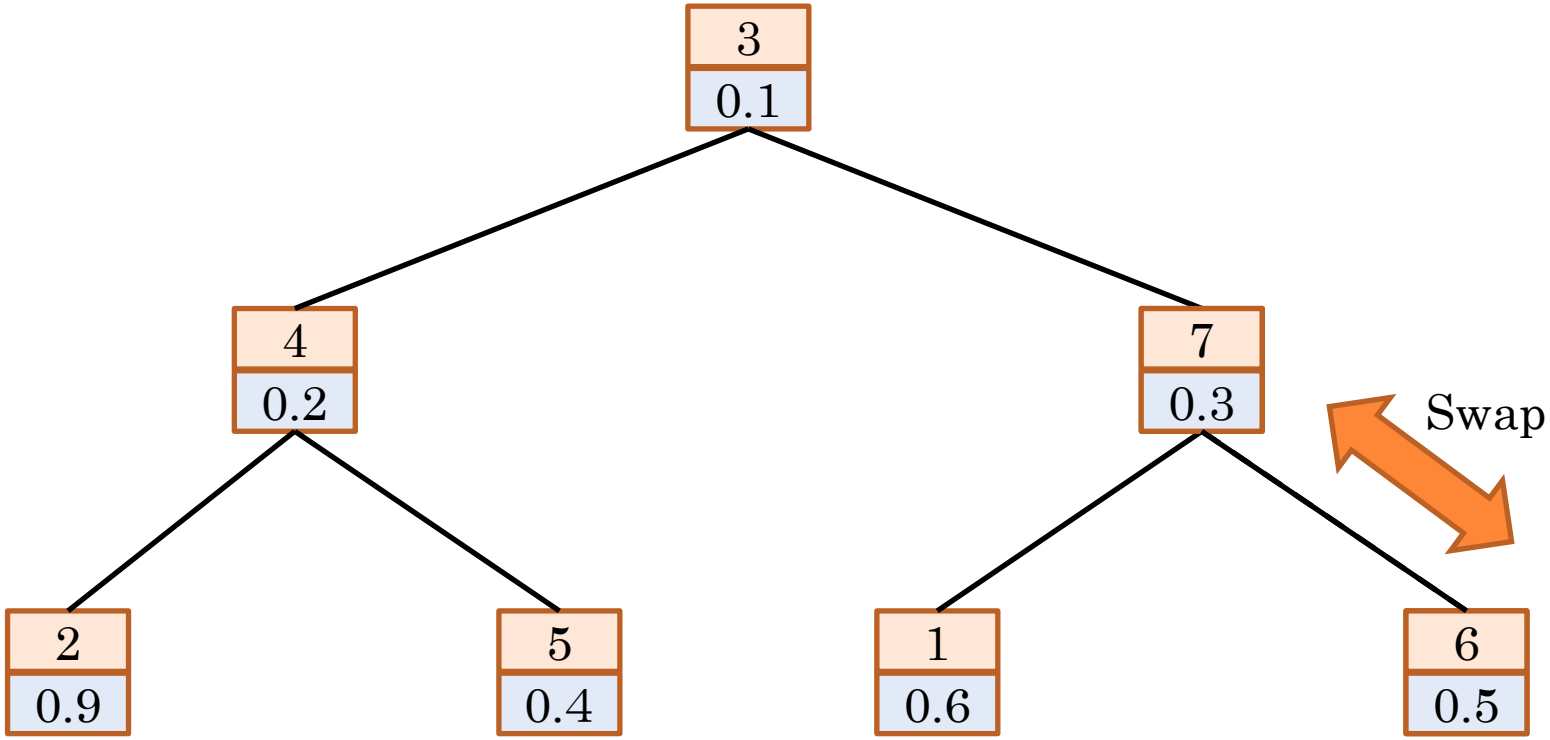

1
0.6
1
0.6
2
0.9
2

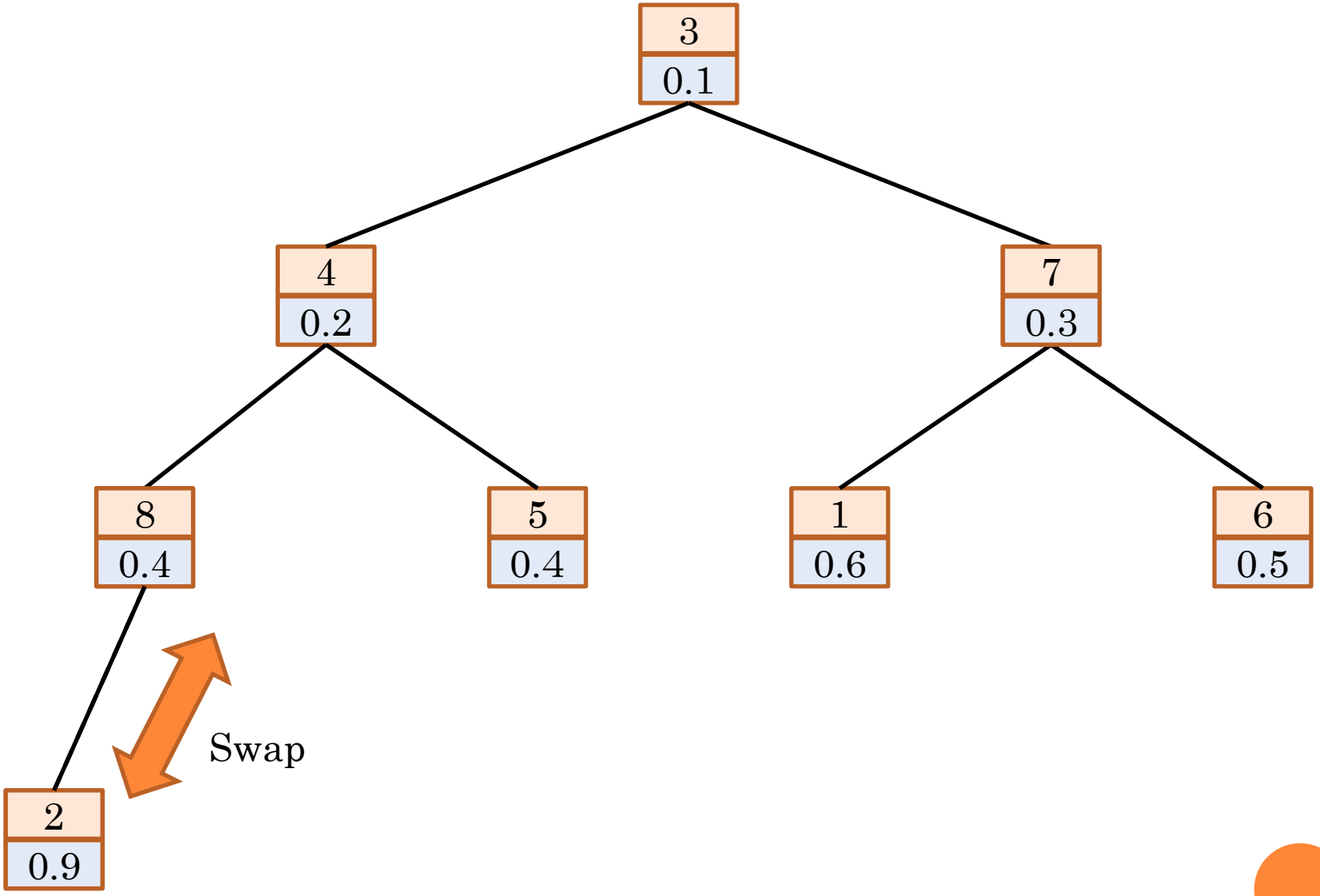

1
0.6
2
0.9
3
0.1
3
0.1
2
0.9
1
0.6


3
0.1
2
0.9
1
0.6
4
0.2
Swap
3
0.1
4
0.2
1
0.6
2
0.9


3
0.1
4
0.2
1
0.6
2
0.9
5
0.4


3
0.1
4
0.2
5
0.5
2
0.9
5
0.4
1
0.6
Swap


3
0.1
4
0.2
7
0.3
Swap
2
0.9
5
0.4
1
0.6
6
0.5


3
0.1
4
0.2
7
0.3
8
0.4
5
0.4
1
0.6
6
0.5
Swap
2
0.9

# 最小要素の取出し

- 最小要素は木の根として保存されている
- 最小要素を取り除き、再構築する
  - 最小要素の取り除き:リスト中の1番が空く
  - 最後尾の要素を取り除き、リストの1番に入れる
  - 適切な位置へシフトダウンする

```
O poll(){
        O  t = L.get(1);
        O  x = L.removeLast();
        L.set(1, x);
        shiftDown(1);
        return  t;
}
```

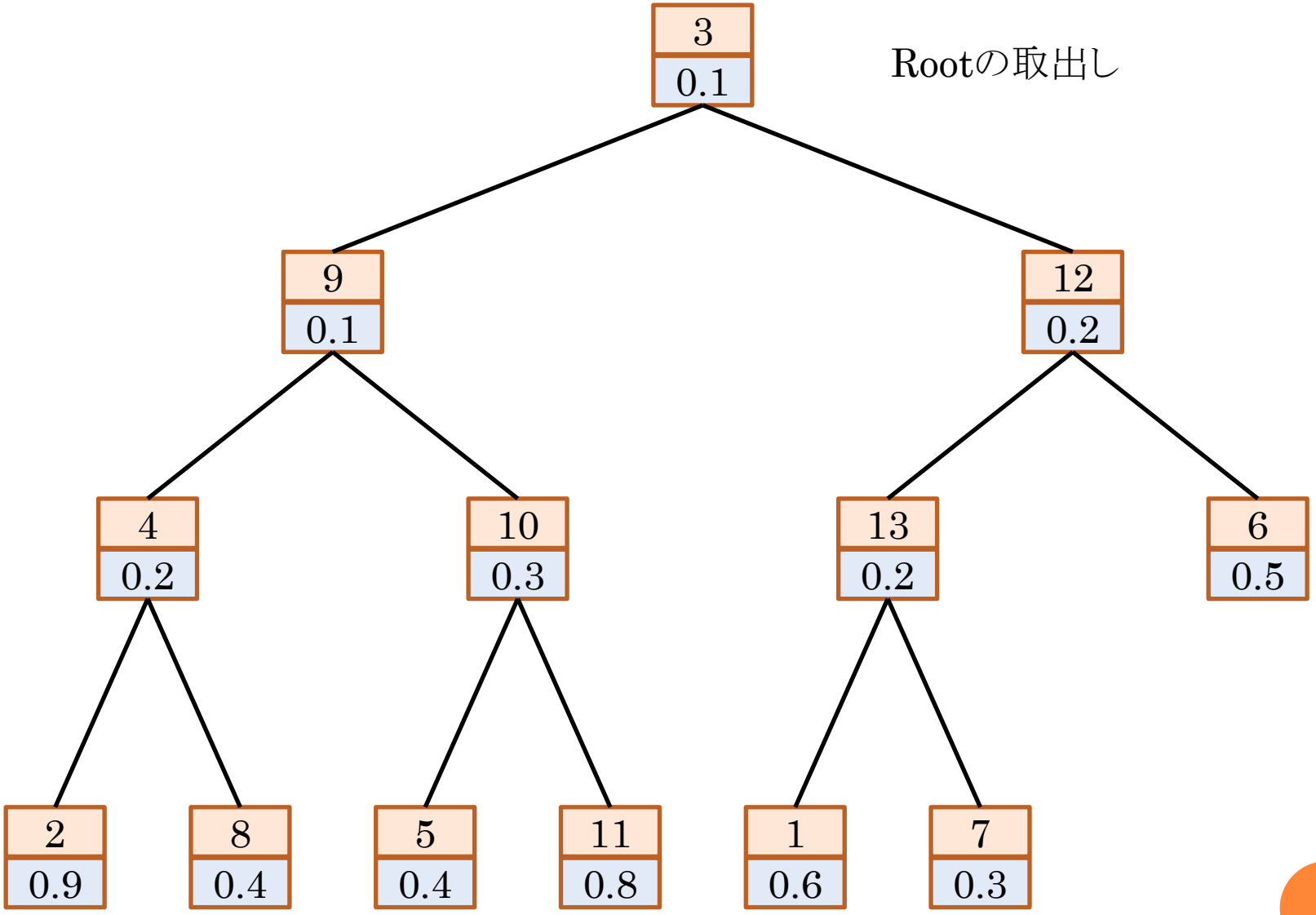
# 要素の取出し:シフトダウン

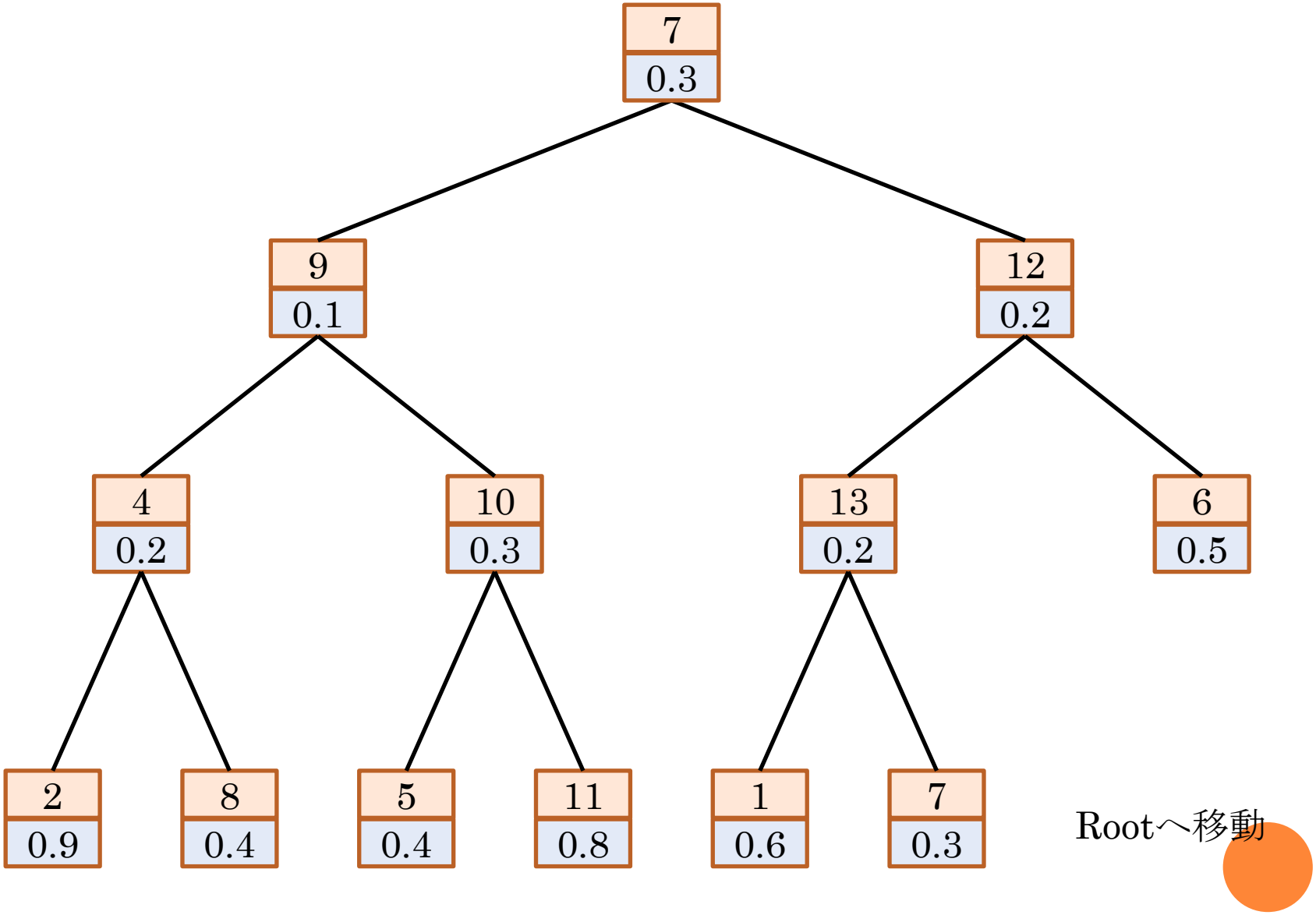
- 追加した新要素を正しい位置へ移動
- 位置$k$の要素が、子の要素の位置$2k$と$2k+1$の小さいほうの値より大きい場合、その小さい値の子を入れ替える

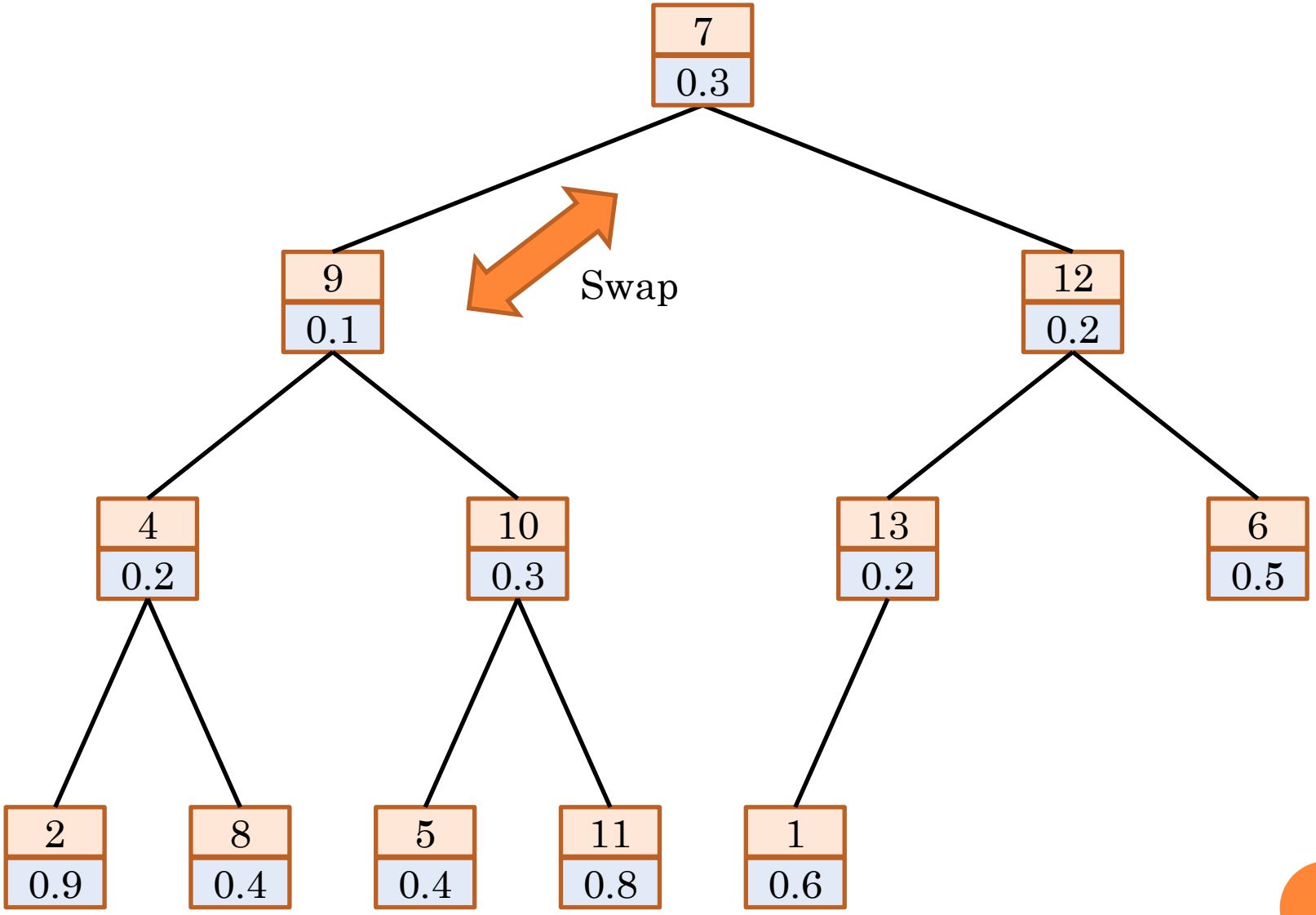
# 要素の取出し:シフトダウン

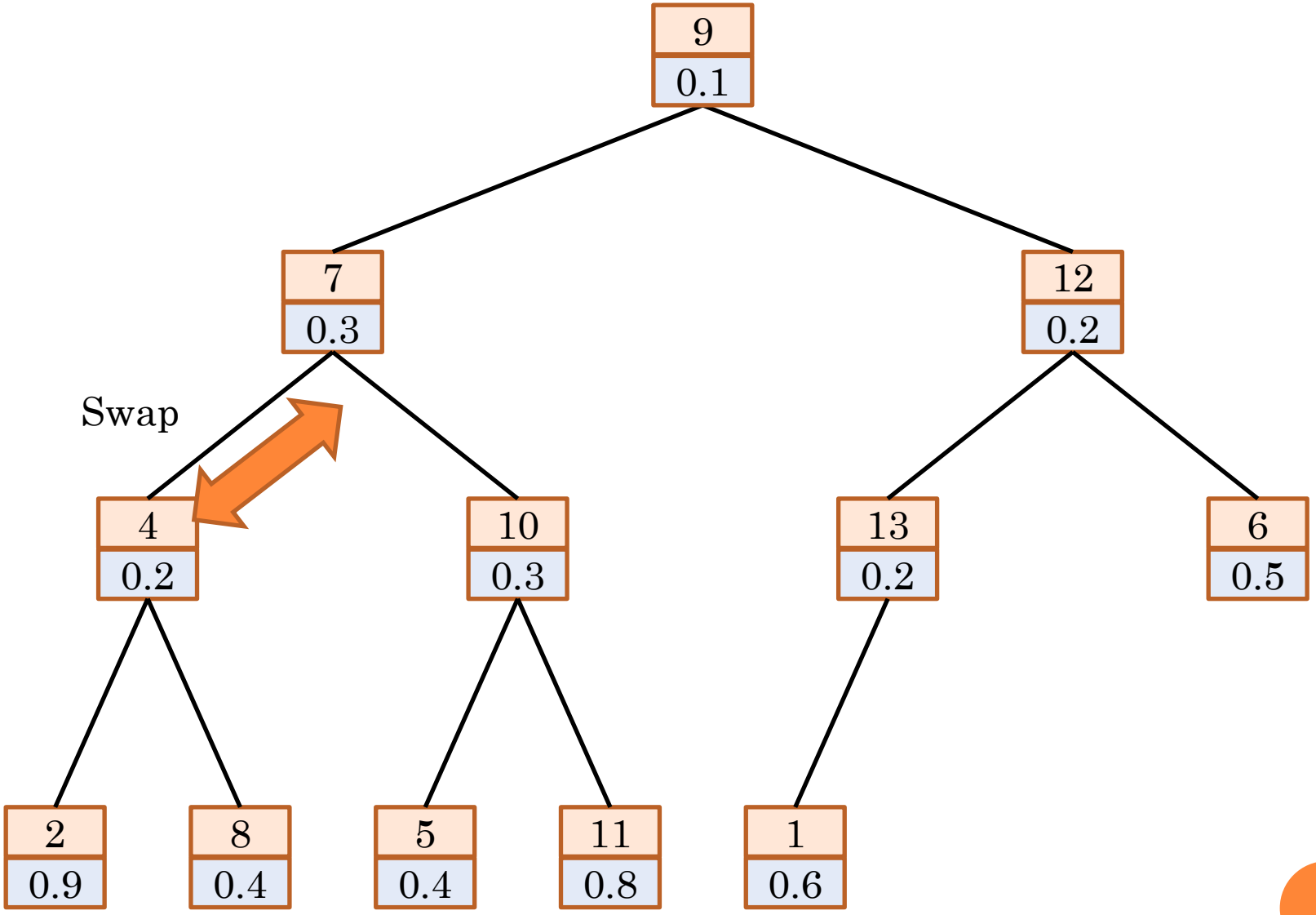
```
void shiftDown(int k){
        int n = |L|;
        if(2×k ≤ n){
                int j = 2×k;
                if(j < n && isLess(j+1, j)) j++;
                if(isLess(k, j))return;
                swap(k, j);
                shiftDown(j);
        }
}
```

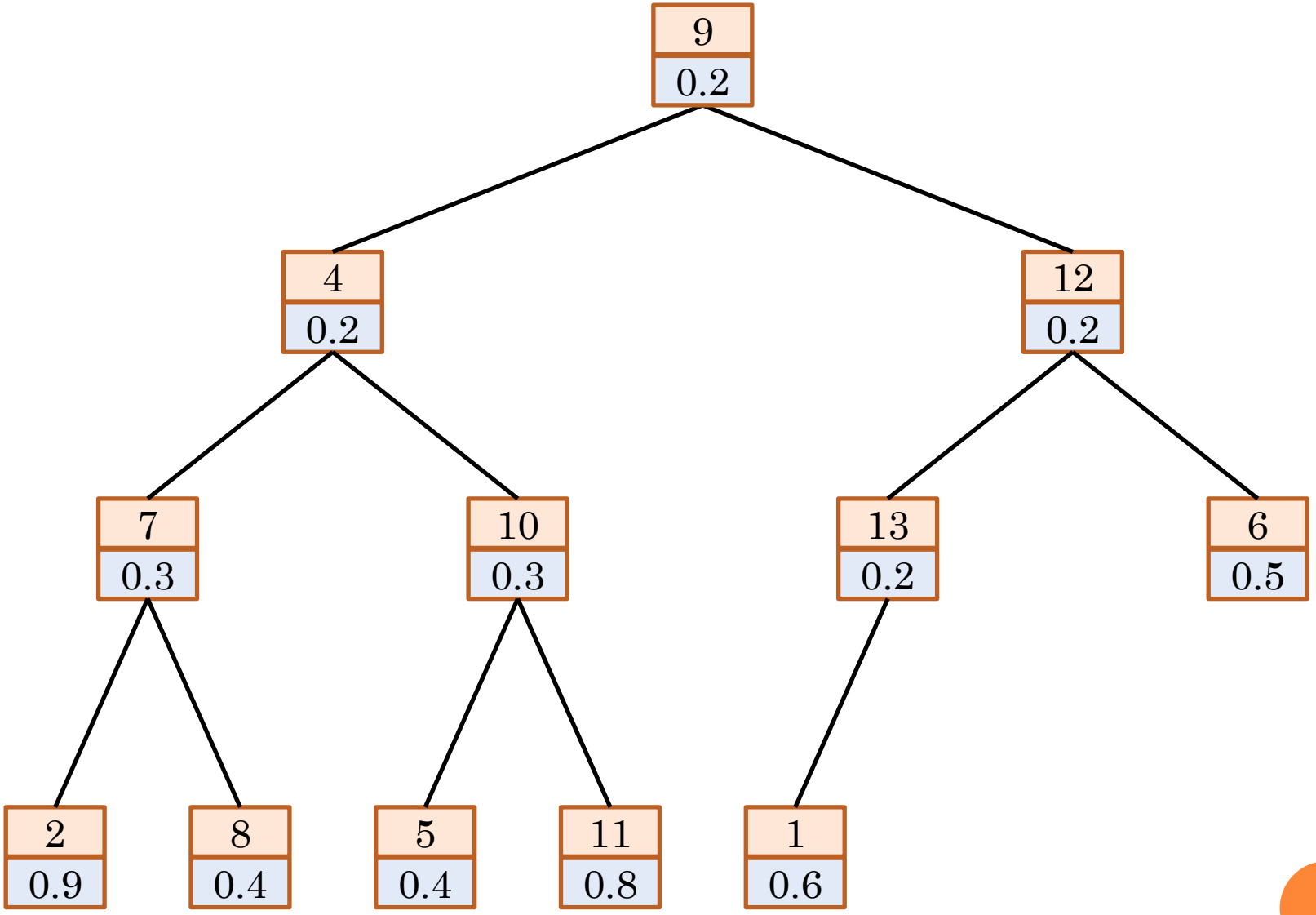
Rootの取出し




7
0.3
9
0.1
12
0.2
4
0.2
10
0.3
13
0.2
6
0.5
2
0.9
8
0.4
5
0.4
11
0.8
1
0.6
7
0.3
Rootへ移動


7
0.3
9
0.1
Swap
12
0.2
4
0.2
10
0.3
13
0.2
6
0.5
2
0.9
8
0.4
5
0.4
11
0.8
1
0.6


9
0.1
7
0.3
12
0.2
Swap
4
0.2
10
0.3
13
0.2
6
0.5
2
0.9
8
0.4
5
0.4
11
0.8
1
0.6


9
0.2
4
0.2
12
0.2
7
0.3
10
0.3
13
0.2
6
0.5
2
0.9
8
0.4
5
0.4
11
0.8
1
0.6

# 特定の要素の値を小さくする

- その要素のインデクス*k*
- shiftUpを*k*を起点に実施

```
void reduceValue(O o){
        k = o のリスト中のインデクス
        shiftUp(k)
}
```

# 特定の要素の値を大きくする

- その要素のインデクス*k*
- shiftDownを*k*を起点に実施

```
void raiseValue(O o){
        k = o のリスト中のインデクス
        shiftDown(k)
}
```

BinaryHeap.java

```java
package utils;

import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 *
 * @author tadaki
 */
public class BinaryHeap<T> {

    /** データを保持するリスト */
    private List<T> list;
    /** 要素を比較する方法 */
    private Comparator<T> comparator = null;
    /** 要素数 */
    private int n;

    /**
     * コンストラクタ：比較方法を指定する場合
     * 比較方法を指定しない場合は、
     *     要素はインターフェイスComparableを実装していること
     * @param comparator
     */
    public BinaryHeap(Comparator<T> comparator) {
        this();
        this.comparator = comparator;
    }

    public BinaryHeap() {
        list = Collections.synchronizedList(new ArrayList<T>());
        list.add(null);
        n = 0;
    }

    /**
     * 新しい要素を追加する
     * @param t
     * @return
     */
    public boolean add(T t) {
        boolean b = list.add(t);
        if (b) {
```

```
            n++;
            shiftUp(n);
        }
        return b;
    }

    /**
     * 最小の要素を得る：削除しない
     * @return
     */
    public T peek() {
        if (n == 0) {
            return null;
        }
        return list.get(1);
    }

    /**
     * 最小の要素を取り出し、削除する
     * @return
     */
    public T poll() {
        T t = null;
        if (n == 0) {
            return t;
        }
        if (n == 1) {//残りの要素が一つ
            t = list.remove(n);
            n--;
        } else {
            T x = list.remove(n);
            t = list.get(1);
            n--;
            list.set(1, x);
            shiftDown(1);
        }
        return t;
    }

    /**
     * 特定の要素の値を小さくした場合の再配置
     * @param t
     */
    public void reduceValue(T t) {
        int k = list.indexOf(t);
        shiftUp(k);
```

```
    }

    /**
     * 特定の要素の値を大きくした場合の再配置
     * @param t
     */
    public void raiseValue(T t){
        int k = list.indexOf(t);
        shiftDown(k);
    }

    /**
     * リストの取得
     * @return
     */
    public List<T> getList() {
        return list;
    }

    public boolean isEmpty() {
        return (n == 0);
    }

    public boolean contains(T t) {
        return list.contains(t);
    }


/*********************************************************************/
    /**
     * あるk にあるobjectを上位の適切な位置に置く
     * @param k
     */
    private void shiftUp(int k) {
        if (k > 1 && isLess(k, (int) (k / 2))) {
            int j = (int) (k / 2);
            swap(k, j);
            shiftUp(j);
        }
    }

    /**
     * あるk にあるobjectを下位の適切な位置に置く
     * @param k
     */
    private void shiftDown(int k) {
```

```
        if (2 * k <= n) {
            int j = 2 * k;
            if (j < n && isLess(j + 1, j)) {
                j++;
            }
            if (isLess(k, j)) {
                return;
            }
            swap(k, j);
            shiftDown(j);
        }
    }

    @SuppressWarnings("unchecked")
    private boolean isLess(int i, int j) {
        int a;
        T x = list.get(i);
        T y = list.get(j);
        if (comparator == null
                && x instanceof Comparable && y instanceof Comparable) {
            a = ((Comparable) x).compareTo((Comparable) y);
        } else {
            a = comparator.compare(x, y);
        }
        return (a < 0);
    }

    private void swap(int i, int j) {
        T o = list.get(i);
        list.set(i, list.get(j));
        list.set(j, o);
    }


/*********************************************************************/
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        /** サンプルとなるデータオブジェクト */
        class Data {

            int label;
            double value;

            public Data(int label, double value) {
```

```
                this.label = label;
                this.value = value;
            }
        }
        /** サンプルとなるデータの比較方法 */
        class CompareData implements Comparator<Data> {

            @Override
            public int compare(Data o1, Data o2) {
                int a = 0;
                if (o1.value > o2.value) {
                    a = 1;
                }
                if (o1.value < o2.value) {
                    a = -1;
                }
                return a;
            }
        }
        BinaryHeap<Data> h = new BinaryHeap<>(new CompareData());
        /** データ追加 */
        int n = 20;
        for (int i = 0; i < n; i++) {
            Data d = new Data(i + 1, Math.random());
            h.add(d);
        }
        /** 結果取得 */
        List<Data> list = h.getList();

        /** 最小値を削除 */
        for (int i = 0; i < 2; i++) {
            Data d = h.poll();
            n--;
        }
        /** 出力 */
        NumberFormat format = NumberFormat.getInstance();
        format.setMaximumFractionDigits(4);

        int l = (int) (Math.log((double) n) / Math.log(2.) + 0.1);
        for (int i = 0; i <= l; i++) {
            int m = (int) (Math.pow(2., (double) i) + 0.1);
            System.out.println();
            for (int j = 0; j < m; j++) {
                int k = m + j;
                if (k != 0 && k <= n) {
                    System.out.print("(");
```

```
                    System.out.print(list.get(k).label);
                    System.out.print(",");
                    System.out.print(format.format(list.get(k).value));
                    System.out.print(") ");
                }
            }
        }
        System.out.println();
    }
}
```