1.剑指 Offer 40. 最小的k个数

普通方法

```java
public int[] getLeastNumbers(int[] arr, int k) {
    PriorityQueue<Integer> priorityQueue = new PriorityQueue<>((o1, o2) -> o2 - o1);
    for (int i : arr) {
        priorityQueue.offer(i);
        if (priorityQueue.size() > k) priorityQueue.poll();
    }
    return priorityQueue.stream().mapToInt((Integer i) -> i.intValue()).toArray();
}
```

船长方法

```java
    public int[] getLeastNumbers(int[] arr, int k) {
//        PriorityQueue<Integer> priorityQueue = new PriorityQueue<>(new Comparator<Integer>() {
//            @Override
//            public int compare(Integer o1, Integer o2) {
//                return o2 - o1;
//            }
//        });
        PriorityQueue<Integer> priorityQueue = new PriorityQueue<>((o1, o2) -> o2 - o1);
        for (int i : arr) {
            priorityQueue.offer(i);
            if (priorityQueue.size() > k) priorityQueue.poll();
        }
        return priorityQueue.stream().mapToInt(i -> i).toArray();
    }
```

优化方法

```java
public int[] getLeastNumbers(int[] arr, int k) {
    if (arr.length == 0 || k == 0) return new int[0];
    PriorityQueue<Integer> priorityQueue = new PriorityQueue<>((o1, o2) -> o2 - o1);
    for (int i = 0; i < k; i++) {
        priorityQueue.offer(arr[i]);
    }
    for (int i = k; i < arr.length; i++) {
        if (arr[i] < priorityQueue.peek()) {
            priorityQueue.poll();
            priorityQueue.offer(arr[i]);
        }
    }
    return priorityQueue.stream().mapToInt(i -> i).toArray();
}
```

2.1046. 最后一块石头的重量

```java
    public int lastStoneWeight(int[] stones) {
        PriorityQueue<Integer> priorityQueue = new PriorityQueue<>((o1, o2) -> o2 - o1);
        for (int stone : stones) {
            priorityQueue.offer(stone);
        }
        while (priorityQueue.size() > 1) {
            int x = priorityQueue.poll();
            int y = priorityQueue.poll();
            if (x > y) {
                priorityQueue.offer(x - y);
            }
        }
        return priorityQueue.isEmpty() ? 0 : priorityQueue.poll();
    }
```

3.703. 数据流中的第 K 大元素

```java
public class KthLargest {
    PriorityQueue<Integer> priorityQueue;
    int k;

    public KthLargest(int k, int[] nums) {
        priorityQueue = new PriorityQueue<Integer>();
        this.k = k;
        for (int num : nums) {
            add(num);
        }
    }

    public int add(int val) {
        priorityQueue.offer(val);
        if (priorityQueue.size() > k) priorityQueue.poll();
        return priorityQueue.peek();
    }
}
```

4.373. 查找和最小的K对数字

```java
    public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k) {
        PriorityQueue<int[]> priorityQueue = new PriorityQueue<>(new Comparator<int[]>() {
            @Override
            public int compare(int[] o1, int[] o2) {
                return o2[2] - o1[2];
            }
        });
        for (int i = 0; i < nums1.length; i++) {
            for (int j = 0; j < nums2.length; j++) {
                if (priorityQueue.size() < k || (nums1[i] + nums2[j]) < priorityQueue.peek()[2]) {
                    priorityQueue.offer(new int[]{nums1[i], nums2[j], nums1[i] + nums2[j]});
                    if (priorityQueue.size() > k) priorityQueue.poll();
                } else break;
            }
        }
        List<List<Integer>> result = new ArrayList<>();
        while (!priorityQueue.isEmpty()) {
            int[] ints = priorityQueue.poll();
            result.add(new ArrayList<Integer>() {{
                this.add(ints[0]);
                this.add(ints[1]);
            }});
        }
        return result;
    }
```

5.215. 数组中的第K个最大元素

```java
    public int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> priorityQueue = new PriorityQueue<>();
        for (int num : nums) {
            priorityQueue.offer(num);
            if (priorityQueue.size() > k) priorityQueue.poll();
        }
        return priorityQueue.peek();
    }
```

6.692. 前K个高频单词

```java
    public List<String> topKFrequent(String[] words, int k) {
        HashMap<String, Integer> map = new HashMap<>();
        for (String word : words) {
            map.put(word, map.getOrDefault(word, 0) + 1);
        }
        PriorityQueue<Map.Entry<String, Integer>> priorityQueue = new
PriorityQueue<>(new Comparator<Map.Entry<String, Integer>>() {
            @Override
            public int compare(Map.Entry<String, Integer> o1, Map.Entry<String,
Integer> o2) {
                return o1.getValue() == o2.getValue() ?
o2.getKey().compareTo(o1.getKey()) : o1.getValue() - o2.getValue();
            }
        });
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            priorityQueue.offer(entry);
            if (priorityQueue.size() > k) priorityQueue.poll();
        }
        List<String> result = new ArrayList<>();
        while (!priorityQueue.isEmpty()) {
            result.add(0, priorityQueue.poll().getKey());
        }
        return result;
    }
```

7.面试题 17.20. 连续中值

```java
public class MedianFinder {
    PriorityQueue<Integer> smallHeap;
    PriorityQueue<Integer> bigHeap;

    /**
     * initialize your data structure here.
     */
    public MedianFinder() {
        smallHeap = new PriorityQueue<>();
        bigHeap = new PriorityQueue<>((o1, o2) -> o2 - o1);
    }

    public void addNum(int num) {
        smallHeap.offer(num);
        bigHeap.offer(smallHeap.poll());
        while (bigHeap.size() > smallHeap.size()) {
            smallHeap.offer(bigHeap.poll());
        }

    }

    public double findMedian() {
        if (smallHeap.size() == bigHeap.size()) {
            return (smallHeap.peek() + bigHeap.peek()) / 2.0d;
        }
        return smallHeap.peek();
    }
}
```

8.295. 数据流的中位数

```java
public class MedianFinder {
    PriorityQueue<Integer> smallHeap;
    PriorityQueue<Integer> bigHeap;

    /**
     * initialize your data structure here.
     */
    public MedianFinder() {
        smallHeap = new PriorityQueue<>();
        bigHeap = new PriorityQueue<>((o1, o2) -> o2 - o1);
    }

    public void addNum(int num) {
        smallHeap.offer(num);
        bigHeap.offer(smallHeap.poll());
        while (bigHeap.size() > smallHeap.size()) {
            smallHeap.offer(bigHeap.poll());
        }

    }

    public double findMedian() {
        if (smallHeap.size() == bigHeap.size()) {
            return (smallHeap.peek() + bigHeap.peek()) / 2.0d;
        }
        return smallHeap.peek();
    }
}
```

9.1801. 积压订单中的订单总数

```
    public int getNumberOfBacklogOrders(int[][] orders) {
        PriorityQueue<int[]> buyQ = new PriorityQueue<>((o1, o2) -> o2[0] - o1[0]);
        PriorityQueue<int[]> sellQ = new PriorityQueue<>((o1, o2) -> o1[0] - o2[0]);
        for (int[] order : orders) {
            int price = order[0], amount = order[1], orderType = order[2];
            if (orderType == 0) {// 采购订单
                while (amount > 0 && !sellQ.isEmpty() && sellQ.peek()[0] <= price) {
                    if (amount >= sellQ.peek()[1]) {
                        amount -= sellQ.poll()[1];
                    } else {
                        int[] poll = sellQ.poll();
                        sellQ.offer(new int[]{poll[0], poll[1] - amount});
                        amount = 0;
                    }
                }
                if (amount > 0) buyQ.offer(new int[]{price, amount});
            } else {
                while (amount > 0 && !buyQ.isEmpty() && buyQ.peek()[0] >= price) {
                    if (amount >= buyQ.peek()[1]) {
                        amount -= buyQ.poll()[1];
                    } else {
                        int[] poll = buyQ.poll();
                        buyQ.offer(new int[]{poll[0], poll[1] - amount});
                        amount = 0;
                    }
                }
                if (amount > 0) sellQ.offer(new int[]{price, amount});
            }
        }
        int result = 0;
        while (!buyQ.isEmpty()) {
            result = (result + buyQ.poll()[1]) % 1000000007;
        }
        while (!sellQ.isEmpty()) {
            result = (result + sellQ.poll()[1]) % 1000000007;
        }
        return result;
    }
```

10.264. 丑数 II

```java
    public int nthUglyNumber(int n) {
        int[] ints = {2, 3, 5};
        HashSet<Long> set = new HashSet<>();
        PriorityQueue<Long> priorityQueue = new PriorityQueue<>();
        priorityQueue.offer(1l);
        set.add(1l);
        int result = 0;
        for (int i = 0; i < n; i++) {
            long curr = priorityQueue.poll();
            result = (int) curr;
            for (int anInt : ints) {
                long next = curr * anInt;
                if (set.add(next)) priorityQueue.offer(next);
            }
        }
        return result;
    }
```

11.313. 超级丑数

```java
    public int nthSuperUglyNumber(int n, int[] primes) {
        HashSet<Long> set = new HashSet<>();
        PriorityQueue<Long> priorityQueue = new PriorityQueue<>();
        priorityQueue.offer(1l);
        set.add(1l);
        int result = 0;
        for (int i = 0; i < n; i++) {
            long curr = priorityQueue.poll();
            result = (int) curr;
            for (int anInt : primes) {
                long next = curr * anInt;
                if (set.add(next)) priorityQueue.offer(next);
            }
        }
        return result;
    }
```

12.1753. 移除石子的最大得分

```java
    public int maximumScore(int a, int b, int c) {
        PriorityQueue<Integer> priorityQueue = new PriorityQueue<>((o1, o2) -> o2 - o1);
        priorityQueue.offer(a);
        priorityQueue.offer(b);
        priorityQueue.offer(c);
        int score = 0;
        while (true) {
            Integer first = priorityQueue.poll();
            Integer second = priorityQueue.poll();
            if (second == 0) break;
            score += 1;
            priorityQueue.offer(first - 1);
            priorityQueue.offer(second - 1);
        }
        return score;
    }
```

13.355. 设计推特

```
public class Twitter {
    private int timestamp = 0;

    private class Tweet {
        private int id;
        private int time;
        private Tweet next;

        public Tweet(int id, int time) {
            this.id = id;
            this.time = time;
            this.next = null;
        }
    }

    private class User {
        private int id;
        private Set<Integer> followed;
        private Tweet head;

        public User(int id) {
            this.id = id;
            followed = new HashSet<Integer>();
            this.head = null;
            followed.add(id);
        }

        public void follow(int userId) {
            followed.add(userId);
        }

        public void unFollow(int userId) {
            if (userId != this.id) followed.remove(userId);
        }

        public void post(int tweetId) {
            Tweet tweet = new Tweet(tweetId, timestamp);
            timestamp++;
            tweet.next = head;
            head = tweet;
        }
    }

    HashMap<Integer, User> userMap = new HashMap<Integer, User>();

    public Twitter() {

    }

    public void postTweet(int userId, int tweetId) {
```

```java
        if (!userMap.containsKey(userId)) {
            User user = new User(userId);
            userMap.put(userId, user);
        }
        User user = userMap.get(userId);
        user.post(tweetId);
    }

    public List<Integer> getNewsFeed(int userId) {
        ArrayList<Integer> result = new ArrayList<>();
        if (!userMap.containsKey(userId)) return result;
        Set<Integer> users = userMap.get(userId).followed;
        PriorityQueue<Tweet> priorityQueue = new PriorityQueue<>((o1, o2) -> (o2.time - o1.time));
        for (Integer id : users) {
            Tweet twt = userMap.get(id).head;
            if (twt == null) continue;
            priorityQueue.offer(twt);
        }
        while (!priorityQueue.isEmpty()) {
            if (result.size() == 10) break;
            Tweet twt = priorityQueue.poll();
            result.add(twt.id);
            if (twt.next != null) {
                priorityQueue.offer(twt.next);
            }
        }
        return result;
    }

    public void follow(int followerId, int followeeId) {
        if (!userMap.containsKey(followerId)) userMap.put(followerId, new User(followerId));
        if (!userMap.containsKey(followeeId)) userMap.put(followeeId, new User(followeeId));
        userMap.get(followerId).follow(followeeId);
    }

    public void unfollow(int followerId, int followeeId) {
        if (userMap.containsKey(followerId)) {
            userMap.get(followerId).unFollow(followeeId);
        }
    }


}
```

14.手写堆

```java
public class MyPQ {
    int[] queue;
    int size;

    public MyPQ() {
        queue = new int[11];
        size = 0;
    }

    public boolean offer(int val) {
        queue[++size] = val;
        int index = size;
        while (index > 1 && queue[index] > queue[index / 2]) {
            int temp = queue[index];
            int s = queue[index / 2];
            queue[index] = queue[index / 2];
            queue[index / 2] = temp;
            index = index / 2;
        }
        return true;
    }


    public int poll() {
        if (size == 0) return -1;
        int result = queue[1];
        queue[1] = queue[size--];
        int index = 1;
        while (index * 2 <= size) {
            int maxIndex = index * 2;
            if (maxIndex + 1 <= size && queue[maxIndex] < queue[maxIndex + 1]) {
                maxIndex = maxIndex + 1;
            }
            if (queue[index] >= queue[maxIndex]) {
                break;
            }
            int temp = queue[index];
            queue[index] = queue[maxIndex];
            queue[maxIndex] = temp;
            index = maxIndex;
        }
        return result;
    }

    public static void main(String[] args) {
        MyPQ myPQ = new MyPQ();
        int[] ints = {3, 2, 1, 5, 5, 5, 6, 4};
        for (int anInt : ints) {
            myPQ.offer(anInt);
        }
        for (int i = 0; i < ints.length; i++) {
            System.out.print(myPQ.poll() + ",");
        }
    }
}
```

15.彩蛋

题目描述：给出一个二叉树的中序序列和后序序列，请推导出二叉树的前序序列，并将前序序列存储到数组中，求出数组元素和下标相乘的累加之和。（数组索引从0开始）

中序序列：

```
12749 294767 309290 437568 469954 564468 659504 745108 924150 967972 974917
1162298 1260765 1329167 1348254 1353244 1365330 1488797 1766350 1862820 2053516
2184231 2208547 2235597 2272075 2273706 2570269 2624658 2656382 2750683 2767632
2824965 2828978 2869946 2989524 3470900 3483131 3496401 3717154 3801252 4024949
4054201 4088280 4211850 4265340 4312774 4408554 4507379 4684979 4716875 4738930
4756732 4766454 4766498 4907755 4916636 4982365 5109920 5124409 5138897 5353629
5421185 5527253 5703586 5910322 5931879 6122034 6154381 6197242 6749378 6815578
6857759 6872983 6979524 7039892 7441409 7471374 7487109 7616136 7640405 7704030
7739330 7868756 7985183 8031754 8044472 8207834 8262199 8304205 8438581 8519367
8782153 9230452 9237105 9535473 9572567 9672291 9738691 9799622 9802046
```

后序序列：

```
294767 12749 437568 469954 309290 659504 564468 745108 967972 1329167 1260765
1162298 974917 1365330 1353244 1766350 2053516 1862820 1488797 2208547 2184231
1348254 2272075 2570269 2273706 2235597 2656382 2750683 2824965 2828978 2767632
2989524 2869946 2624658 924150 4024949 3801252 3717154 3496401 3483131 4312774
4507379 4408554 4265340 4716875 4756732 4766454 4738930 4684979 4211850 4088280
4054201 4907755 4982365 4916636 5109920 4766498 5353629 5138897 5910322 5703586
5527253 6122034 6749378 6815578 6197242 6154381 6872983 6979524 7039892 7471374
7441409 6857759 5931879 5421185 7704030 7739330 7640405 7616136 7487109 5124409
7868756 8031754 7985183 8262199 8207834 8519367 8438581 8782153 8304205 8044472
3470900 9535473 9237105 9672291 9799622 9738691 9802046 9572567 9230452
```