<https://xiaomi.f.mioffice.cn/docs/dock4GoOXYCJVY9jV8G2MJwUZcq>

**第一个目标：找到性能异常点**

现有文件（链接在下面)中包含过去十年每秒的打点数据共315532803（3652\*24\*60\*60+3）个整型数字。现在以1小时（3600个数字）为滑动窗口，每个窗口的90%分位数代表该窗口的延迟。请找出延迟最大的窗口。

文件链接：二选一即可，每行一个打点

​[https://pan.mioffice.cn:443/link/6A75A7B467F1DF9EF4F5902677F263BD](https://pan.mioffice.cn:443/link/6A75A7B467F1DF9EF4F5902677F263BD" \t "/home/doqin/Documents\\x/_blank)​

​[http://cnbj1-fds.api.xiaomi.net/1024data/315532803txt](http://cnbj1-fds.api.xiaomi.net/1024data/315532803txt" \t "/home/doqin/Documents\\x/_blank)​

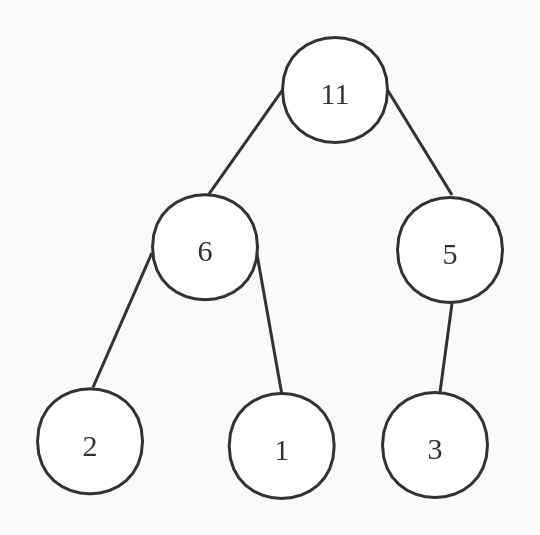
**思路：**

以打点数组[2,3,11,6,1,5,12,4,10,9,13,8,7,14,16]为例。滑动窗口大小为10，寻找第60%分位数。

第60%分位数：将窗口的10个数按照升序排列之后，第10\*0.6大的数即为目标分位数。

【1】：创建两个初始堆。第一个堆为大根堆，大小为10\*0.6=6。第二个堆为小根堆，大小为10\*(1-0.6)=4，保证小根堆的任意一个数大于大根堆的任意一个数。这样做的目的是保证在o(1)时间内获取当前滑动窗口的第6大的数（大根堆的堆顶）和第7大的数（小根堆的堆顶）。算法如下：

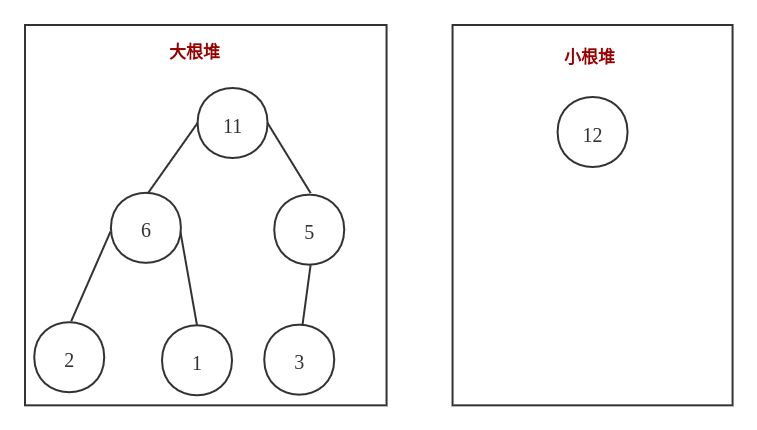
1. 先读取6个数，构成大根堆：

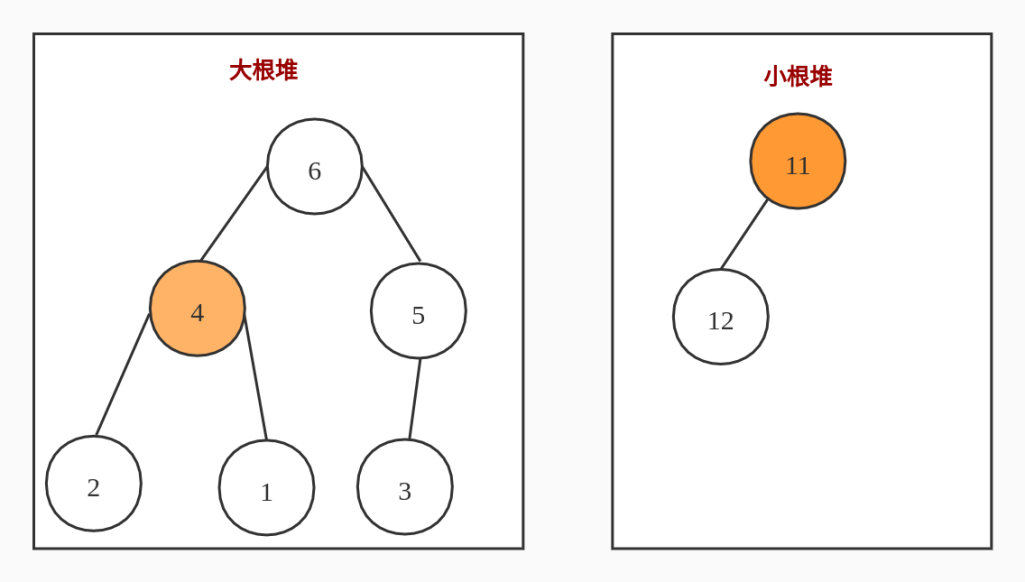


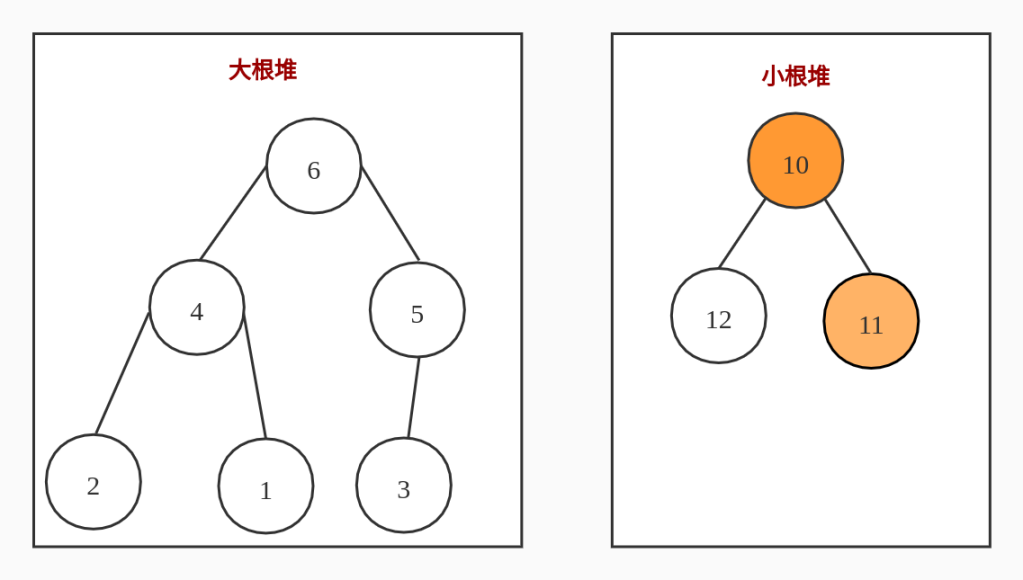
1. 再依次加入4个数。每加入一个数时，需要跟大根堆堆顶进行比较。

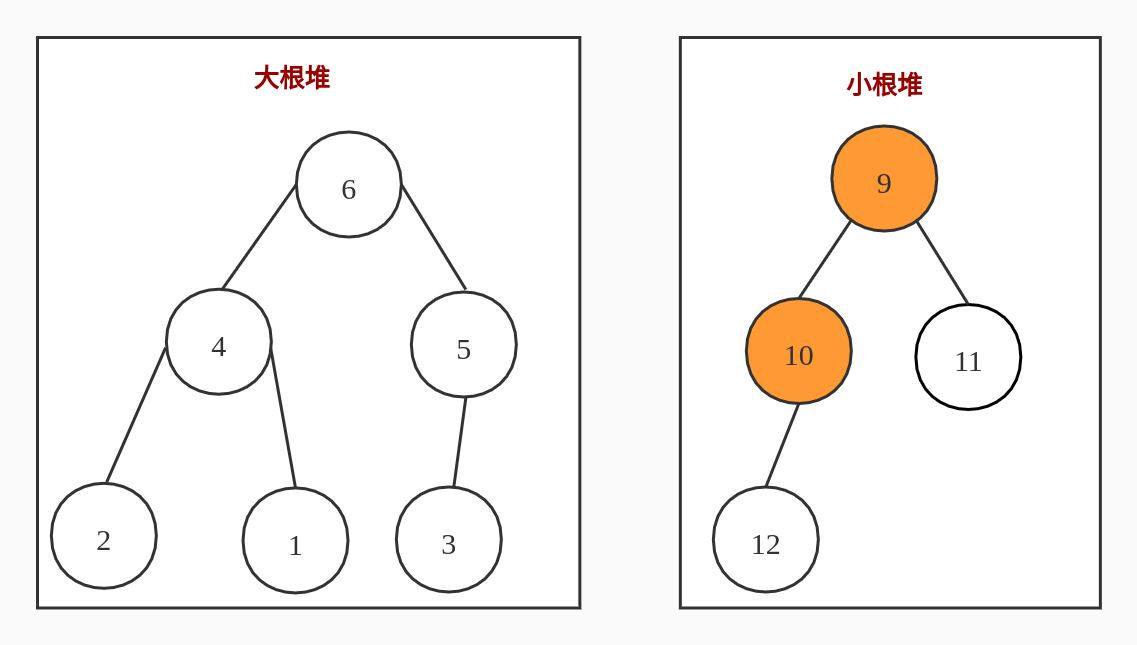
如果 当前元素 >= 大根堆，直接将当前元素加入小根堆

当前元素 < 大根堆，将大根堆堆顶加入小根堆。将当前元素加入大根堆并调整。









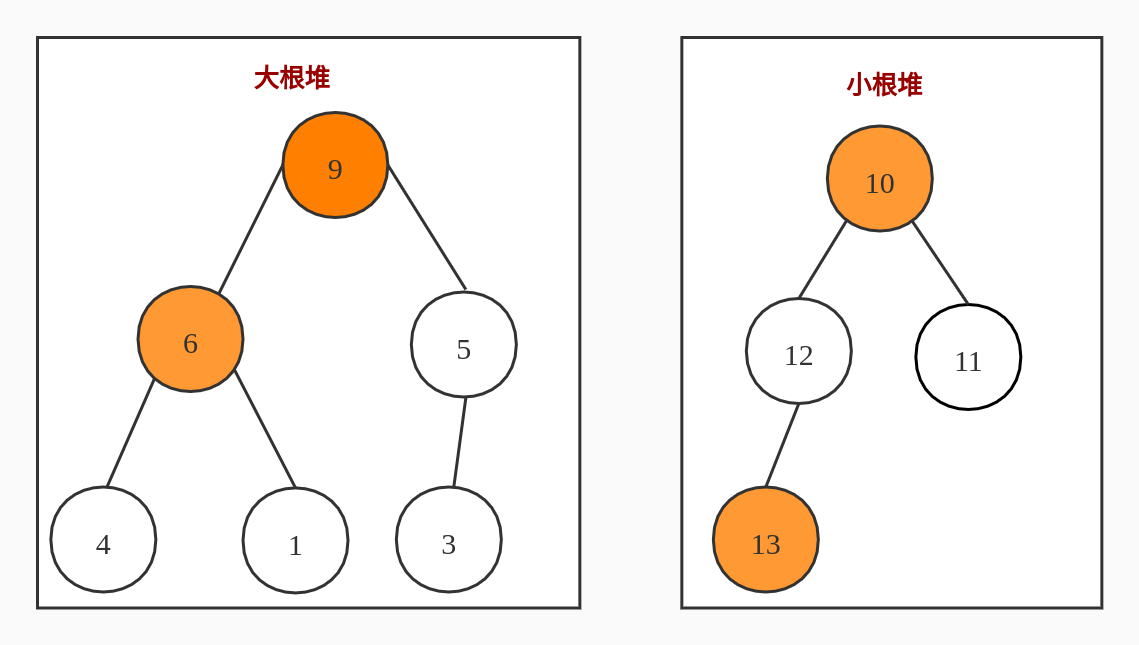
当前窗口的第60%分位数就是：大根堆堆顶--->6

【2】：移动窗口，比较历史最大的第60%分位数。

**\* 如果要移除的元素在大根堆**  
\* 首先移除窗口最左端的元素，然后 比较 a[index -k] 与 大根堆堆顶元素  
\* (1) a[index] <= maxHeap.peek()  
\* 说明应该将a[index]加入大根堆，直接加入大根堆即可,大根堆总数不变  
\* (2) maxHeap.peek() < a[index] <= minHeap.peek()  
\* 由于大根堆移除了一个元素，因此，应该将a[index]加入大根堆,维持大根堆总数不变  
\* (3) a[index] > minHeap.peek()  
\* 应该将 minHeap.peek() 加入大根堆，将 a[index]加入小根堆  
**\* 综上所述： 当 a[index] <= minHeap.peek(), 将a[index]加入大根堆  
\* 当 a[index] > minHeap.peek(), 将 minHeap.peek() 加入大根堆，将 a[index]加入小根堆**  
\*  
**\* 如果要移除的元素在小根堆**  
\* 首先移除窗口最左端的元素，然后 比较 a[index] 与 大根堆堆顶元素  
 \* (1) a[index] <= maxHeap.peek()  
 \* 说明应该将a[index]加入大根堆，将 maxHeap.peek()加入小根堆，小根堆总数不变  
 \* (2) maxHeap.peek() < a[index] <= minHeap.peek()  
 \* 由于小根堆移除了一个元素，因此，应该将a[index]加入小根堆,维持小根堆总数不变  
 \* (3) a[index] > minHeap.peek()  
 \* 应该将 minHeap.peek() 加入小根堆  
 **\* 综上所述： 当 a[index] >= maxHeap.peek(), 将a[index]加入小根堆  
 \* 当 a[index] < maxHeap.peek(), 将 maxHeap.peek() 加入小根堆，将 a[index]加入大根堆**

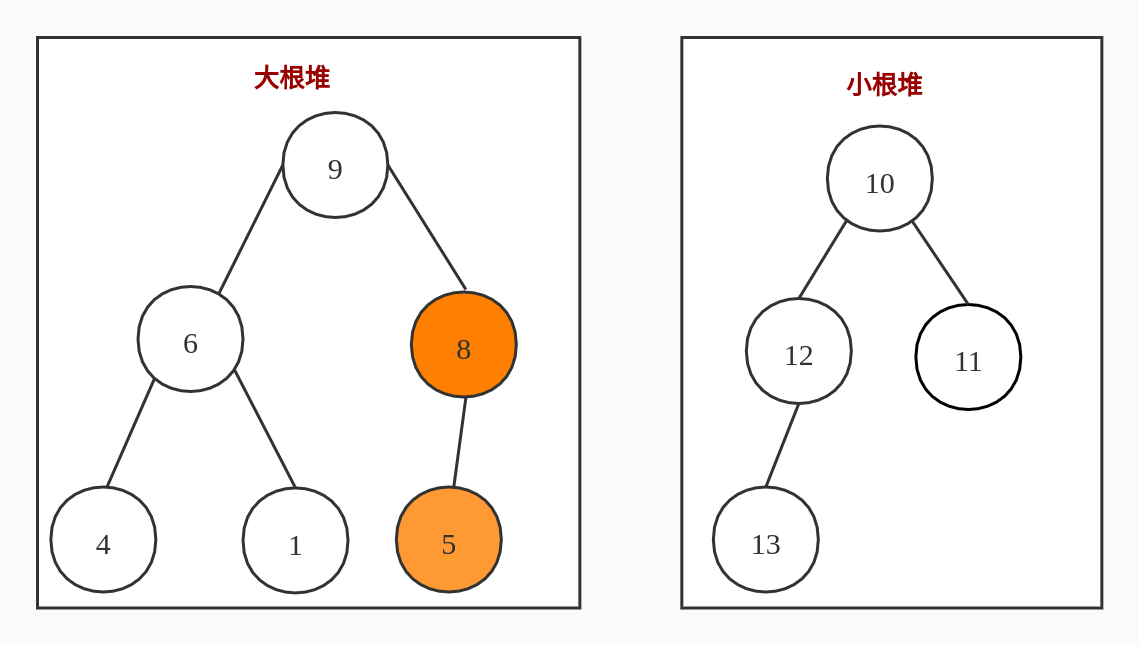
移除的元素：2 移入的元素：13

调整后的第60%分位数是9 > 6 (上一个窗口的分位数)，res = 9



移除的元素：3 移入的元素：8

调整后的第60%分位数是9 = 9 (上一个窗口的分位数)，res = 9



依次类推：

**我的实现如下：**

|  |
| --- |
| **public class** Test2 {  **public void initArray(int[] a)** **throws** IOException {  BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(  **new** FileInputStream(**"/home/doqin/Downloads/315532803txt"**)));  **int** i = 0;  String line = reader.readLine();  **while** (line != **null** && StringUtils.*isNotBlank*(line)) {  a[i++] = Integer.*parseInt*(line);  line = reader.readLine();  }  }   **public void initHeap (int[] a, int k, int maxLen, PriorityQueue<Integer> maxHeap, PriorityQueue<Integer> minHeap)** {  *//先初始化 maxLen 个元素的大根堆* **int** i = 0;  **for** (;i < maxLen; i++) {  maxHeap.add(a[i]);  }  *//生成 k-maxLen个元素的小根堆，并保证小根堆堆顶元素 >= 大根堆堆顶元素* **for** (;i< k; i++) {  *//a[i] >= 大根堆堆顶的元素，直接将a[i]加入小根堆* **if** (a[i] >= maxHeap.peek()) {  minHeap.add(a[i]);  }  *//否则将大根堆堆顶加入小根堆，将a[i]加入大根堆并调整* **else** {  **int** target = maxHeap.poll();  maxHeap.add(a[i]);  minHeap.add(target);  }  }  }   **public int search (int[] tmp, int[] a, int k , int index, PriorityQueue<Integer> maxHeap, PriorityQueue<Integer> minHeap)** {  *//当窗口滑动的时侯，首先移除最左边的元素 a[left]。判断该元素在大根堆还是在小根堆  /\*  \* 如果要移除的元素在大根堆  \* 首先移除窗口最左端的元素，然后 比较 a[index -k] 与 大根堆堆顶元素  \* (1) a[index] <= maxHeap.peek()  \* 说明应该将a[index]加入大根堆，直接加入大根堆即可,大根堆总数不变  \* (2) maxHeap.peek() < a[index] <= minHeap.peek()  \* 由于大根堆移除了一个元素，因此，应该将a[index]加入大根堆,维持大根堆总数不变  \* (3) a[index] > minHeap.peek()  \* 应该将 minHeap.peek() 加入大根堆，将 a[index]加入小根堆  \* 综上所述： 当 a[index] <= minHeap.peek(), 将a[index]加入大根堆  \* 当 a[index] > minHeap.peek(), 将 minHeap.peek() 加入大根堆，将 a[index]加入小根堆  \*  \* 如果要移除的元素在小根堆  \* 首先移除窗口最左端的元素，然后 比较 a[index] 与 大根堆堆顶元素  \* (1) a[index] <= maxHeap.peek()  \* 说明应该将a[index]加入大根堆，将 maxHeap.peek()加入小根堆，小根堆总数不变  \* (2) maxHeap.peek() < a[index] <= minHeap.peek()  \* 由于小根堆移除了一个元素，因此，应该将a[index]加入小根堆,维持小根堆总数不变  \* (3) a[index] > minHeap.peek()  \* 应该将 minHeap.peek() 加入小根堆  \* 综上所述： 当 a[index] >= maxHeap.peek(), 将a[index]加入小根堆  \* 当 a[index] < maxHeap.peek(), 将 maxHeap.peek() 加入小根堆，将 a[index]加入大根堆 \*/* **int** res = maxHeap.peek();**for** (**int** i = index; i< a.**length**; i++) {  *//窗口的最左端值也就是要移除的元素* **int** left = a[i-k];  *//如果要移除的元素在大根堆* **if** (left <= maxHeap.peek()) {  *//移除该元素* maxHeap.remove(left);  **if** (a[i] <= minHeap.peek()) {  maxHeap.add(a[i]);  } **else** {  maxHeap.add(minHeap.poll());  minHeap.add(a[i]);  }  }  *//如果要移除的元素在小根堆* **else** {  minHeap.remove(left);  **if** (a[i] >= maxHeap.peek()) {  minHeap.add(a[i]);  } **else** {  minHeap.add(maxHeap.poll());  maxHeap.add(a[i]);  }  }   *//调整完毕后，比对当前窗口的第60%中位数（大根堆堆顶），保存最大60%的中位数* **if**(maxHeap.peek() > res) {  res = maxHeap.peek();  *//拷贝a[i],,,,,,a[i+k] 到tmp中* System.*arraycopy*(a, i, tmp, 0, k);  }  *//System.out.println("res = " + res + " , maxHeap.peek() = " + maxHeap.peek() + ", minHeap.peek() = " + minHeap.peek());* }   **return** res;   }   **public static void** main(String[] args) **throws** IOException {  **long** start = System.*currentTimeMillis*();  Test2 m = **new** Test2();  **int** k = 3600;  **double** ratio = 0.9;  **int** maxLen = (**int**) (k \* ratio);  **int** [] a = **new int** [315532803];  **int**[] tmp = **new int**[3600];*//存放最大延迟窗口* m.initArray(a);  *//int[] a = new int [] {2,3,11,6,1,5,12,4,10,9,13,8,7,14,16};  //生成一个大根堆。大根堆的堆顶元素就是窗口为10的第60%的分位数* PriorityQueue<Integer> maxHeap = **new** PriorityQueue<>(6, **new** Comparator<Integer>() {  @Override  **public int** compare(Integer o1, Integer o2) {  **return** o2-o1;  }  });  *//生成一个小根堆（默认）* PriorityQueue<Integer> minHeap = **new** PriorityQueue<>(4);  m.initHeap(a, k, maxLen, maxHeap, minHeap);  **int** res = m.search(tmp, a, k, k, maxHeap,minHeap);  System.***out***.println(**"res = "** + res + **" , maxHeap.peek() = "** + maxHeap.peek() + **", minHeap.peek() = "** + minHeap.peek());  System.***out***.println(**"time = "** + (System.*currentTimeMillis*() - start));  } } |

**袁杰实现如下：**

|  |
| --- |
| **public class** Test3 {   **public static** Queue<Integer> *queue* = **new** LinkedList<Integer>();;  **public static** PriorityQueue<Integer> *minHeap* = **new** PriorityQueue<Integer>(); *//小顶堆，默认容量为11* **public static** PriorityQueue<Integer> *maxHeap* = **new** PriorityQueue<Integer>(Collections.*reverseOrder*());   **public static void** main(String[] args) **throws** IOException {  **long** start = System.*currentTimeMillis*();  String filePath = **"/home/doqin/Downloads/315532803txt"**;  File file = **new** File(filePath);  InputStreamReader read = **new** InputStreamReader(  **new** FileInputStream(file));*// 考虑到编码格式* BufferedReader bufferedReader = **new** BufferedReader(read);  String lineTxt = **null**;  **int** count = 0;  Integer fenweishu = -1;  Queue<Integer> tmp = **null**;  **while** ((lineTxt = bufferedReader.readLine()) != **null**){  count++;  **if** (count % 1000000 == 0) System.***out***.println(count);  Integer value = Integer.*parseInt*(lineTxt);  *queue*.add(value);   **if** (*maxHeap*.size() + *minHeap*.size() < 3600){  *maxHeap*.add(value);  } **else** {  Integer queueHead = *queue*.poll();  Integer peek1 = *maxHeap*.peek();  **if** (value <= peek1){  *maxHeap*.add(value);  } **else** {  *minHeap*.add(value);  }   **if** (queueHead <= peek1){  *maxHeap*.remove(queueHead);  } **else** {  *minHeap*.remove(queueHead);  }  }  *addjust*(*maxHeap*,*minHeap*);  **if** (*maxHeap*.size() + *minHeap*.size() == 3600){  **if** (*maxHeap*.peek() > fenweishu){  fenweishu = *maxHeap*.peek();  tmp = **new** LinkedList<Integer>(*queue*);  }  }  }   System.***out***.println(fenweishu);  List<Integer> collect = tmp.stream().collect(Collectors.*toList*());  Collections.*sort*(collect);  System.***out***.println(tmp.stream().collect(Collectors.*toList*()));  bufferedReader.close();  read.close();  System.***out***.println(**"time = "** + (System.*currentTimeMillis*() - start));  }   **public static void** addjust(PriorityQueue<Integer> maxHeap,PriorityQueue<Integer> minHeap){  **while** (maxHeap.size() > 3240 || minHeap.size() > 360){  **if** (maxHeap.size() > 3240){  Integer poll = maxHeap.poll();  minHeap.add(poll);  }  **if** (minHeap.size() > 360){  Integer poll = minHeap.poll();  maxHeap.add(poll);  }  }  } } |

**第二个目标：找到性能异常点**

<https://xiaomi.f.mioffice.cn/docs/dock4r5niJBL1NXbLND8Ix8mY77>

现在我们已经找到了延迟最高的时间段，需要分析一下该段时间内发生的事情如重启。已知该服务被设计成任意连续时间段的延迟累加大于 1s就会自动重启。由于重启打点丢失，请倒推上一任务中找到的时间段中会导致重启的最短间隔。

（打点数字以ns为单位， 1s=10亿ns）

**类似问题：和大于S的最短子序列(**给定长度为n的整数数列：a0,a1,..,an-1，以及整数S。这个数列会有连续的子序列的整数总和大于S的，求这些数列中，最小的长度)****

给定数组 [2,3,1,2,4,3] 和 s = 7, 子数组 [4,3] 是该条件下的最小长度子数组。

|  |
| --- |
| **public static int** shortestArrar (**int**[] a, **int** s) {  **int** fast = 0;  **int** slow = 0;  **int** curSum = 0;  **int** len = a.**length**;  **while** (fast < a.**length**) {  curSum = curSum + a[fast];  fast ++;  **if** (curSum >= s) {  len = Math.*min*(len, fast - slow);  **while** (curSum >= s && slow <=fast) {  len = Math.*min*(len, fast - slow);  curSum = curSum - a[slow];  slow ++;  }  }  }  **return** len;  }   **public static void** main(String[] args) {  **int**[] a = **new int**[] {2,3,1,2,4,3};  **int** s = 7;  **int** len = *shortestArrar* (a, s);  System.***out***.println(len);  } } |