

PromptScratchpadOur Solution(s)Video Explanation

Run Code

Solution 1

```
31     handler.rebalanceHeaps()
32     handler.updateMedian()
33 }
34
35 func (handler *ContinuousMedianHandler) rebalanceHeaps() {
36     if handler.lower.Length()-handler.greater.Length() == 2 {
37         handler.greater.Insert(handler.lower.Remove())
38     } else if handler.greater.Length()-handler.lower.Length() == 2 {
39         handler.lower.Insert(handler.greater.Remove())
40     }
41 }
42
43 func (handler *ContinuousMedianHandler) updateMedian() {
44     if handler.lower.Length() == handler.greater.Length() {
45         sum := (handler.lower.Peek() + handler.greater.Peek())
46         handler.Median = float64(sum) / 2
47     } else if handler.lower.Length() > handler.greater.Length() {
48         handler.Median = float64(handler.lower.Peek())
49     } else {
50         handler.Median = float64(handler.greater.Peek())
51     }
52 }
53
54 type Heap struct {
55     comp    HeapFunc
56     values []int
57 }
58
59 type HeapFunc func(int, int) bool
60
61 var MinHeapFunc = func(a, b int) bool { return a < b }
62 var MaxHeapFunc = func(a, b int) bool { return a > b }
63
64 func NewHeap(fn HeapFunc) *Heap {
65     return &Heap{
66         comp:  fn,
67         values: []int{},
68     }
69 }
70
71 func (h *Heap) Length() int {
72     return len(h.values)
73 }
74
75 func (h *Heap) Peek() int {
76     if len(h.values) == 0 {
77         return -1
78     }
79     return h.values[0]
80 }
81
82 func (h *Heap) Insert(value int) {
83     h.values = append(h.values, value)
84     h.siftUp()
85 }
86
87 func (h *Heap) Remove() int {
88     l := h.Length()
89     h.swap(0, l-1)
90     peeked := h.values[l-1]
91     h.values = h.values[0 : l-1]
92     h.siftDown()
93     return peeked
94 }
95
96 func (h *Heap) siftUp() {
97     currentIndex := h.Length() - 1
98     parentIndex := (currentIndex - 1) / 2
99     for currentIndex > 0 {
100         current, parent := h.values[currentIndex], h.values[parentIndex]
101         if h.comp(current, parent) {
102             h.swap(currentIndex, parentIndex)
103             currentIndex = parentIndex
104             parentIndex = (currentIndex - 1) / 2
105         } else {
106             return
107         }
108     }
109 }
110
111 func (h *Heap) siftDown() {
112     currentIndex := 0
113     endIndex := h.Length() - 1
114     childOneIdx := currentIndex*2 + 1
115     for childOneIdx <= endIndex {
116         childTwoIdx := -1
```