

Solutions

7.1 The Heap Property

Reference example

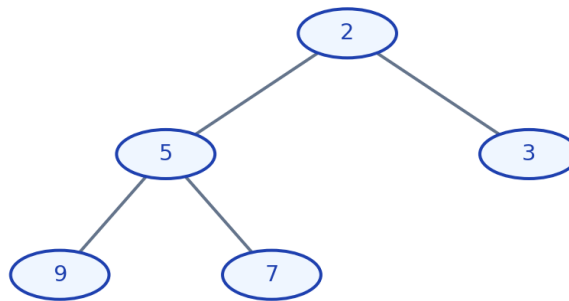
Min-heap array: [2, 5, 3, 9, 7]

Parent $\leq$ each child

Minimum: root

Siblings are not required to be sorted

A complete min-heap



Every parent is at most its children; sibling values need not be sorted.

7.1.1 - Explain

Is [1, 4, 3, 9, 8] a valid min-heap in zero-based array order?

Yes. The root 1 is at most 4 and 3, and parent 4 is at most 9 and 8. No sibling ordering is required.

7.1.2 - Explain

Why cannot a min-heap generally find an arbitrary target in logarithmic time by a BST-style search?

Its ordering is only between a parent and its children, not across entire left and right subtrees. An arbitrary target may require scanning many nodes.

Solutions

7.2 Array Representation

Reference example

Zero-based index i
Left child: $2*i + 1$
Right child: $2*i + 2$
Parent for $i > 0$: $(i - 1) / 2$ using integer division

7.2.1 - Explain

For index 3 in a zero-based heap, give the parent and possible child indices.

The parent is index 1. Child indices are 7 and 8, provided those positions exist.

7.2.2 - Write code

For a heap of size 6, print the existing child indices of node 2.

Check computed positions against the heap size.

```
int size = 6, i = 2;  
int left = 2*i+1, right = 2*i+2;  
if (left < size) System.out.println(left);  
if (right < size) System.out.println(right);
```

Expected output

5

Solutions

7.3 Insertion and Extraction

Reference example

```
PriorityQueue<Integer> q = new PriorityQueue<Integer>();  
q.add(5); q.add(2); q.add(7); q.add(1);  
System.out.println(q.remove());  
System.out.println(q.remove());
```

7.3.1 - Explain

Insert 1 into min-heap [2, 5, 3]. Give one resulting array after standard sift-up.

Append 1 at index 3, swap with 5, then with 2. The resulting array is [1, 2, 3, 5].

7.3.2 - Write code

Use a min-priority queue to print 4, 1, and 2 in priority order. Explain why direct iteration is not a substitute.

Repeated removal exposes the minimum each time; iteration does not promise sorted priority order.

```
PriorityQueue<Integer> queue = new PriorityQueue<Integer>();  
queue.add(4); queue.add(1); queue.add(2);  
while (!queue.isEmpty()) System.out.print(queue.remove() + " ");
```

Expected output

1 2 4

Solutions

7.4 Building and Heap Sort

Reference example

Bottom-up build: sift down from last internal node
Most nodes are near the leaves
Build-heap total: $O(n)$
Repeated extraction in heap sort: $O(n \log n)$

7.4.1 - Explain

Compare building a heap by n separate insertions with bottom-up build in worst-case time.

Repeated insertion has an $O(n \log n)$ worst-case bound. Bottom-up build is $O(n)$.

7.4.2 - Explain

What are the standard in-place heap sort time and auxiliary-space bounds?

It runs in $O(n \log n)$ time and uses $O(1)$ auxiliary space with iterative sift operations. It is generally not stable.