

Task 1: Advanced Data Structures

Here is your task

Your task is to implement a novel data structure - your project lead is calling it a power of two max heap. The rest of your team is doing their best to come up with a better name. The requirements of the data structure are as follows:

- The heap must satisfy the heap property.
- Every parent node in the heap must have 2^x children.
- The value of x must be a parameter of the heap's constructor.
- The heap must implement an insert method.
- The heap must implement a pop max method.
- The heap must be implemented in Java.
- The heap must be performant
 - You must use a more descriptive variable name than x in your implementation.

Think carefully about how you implement each method, and manage the underlying data. Performance is critical, so keep cycles and memory usage to a minimum. Be sure to test your heap with very small and very large values of x . As always, keep a weather eye out for sneaky edge cases.

Solution

```
1 - import java.util.ArrayList;
2 import java.util.List;
3 // Your actual solution class
4 class PowerOfTwoMaxHeap {
5     private final int childrenPower; // the "x" from  $2^x$  children
6     private final int childrenCount; // actual number of children
7     private final List<Integer> heap; // underlying storage
8     // Constructor
9     public PowerOfTwoMaxHeap(int childrenPower) {
10         if (childrenPower < 0) {
11             throw new IllegalArgumentException("childrenPower must be >= 0");
12         }
13         this.childrenPower = childrenPower;
14         this.childrenCount = 1 << childrenPower; //  $2^x$ 
15         this.heap = new ArrayList<>();
16     }
17     // Public method: Insert value into heap
18     public void insert(int value) {
19         heap.add(value);
20         int index = heap.size() - 1;
21
22         while (index > 0) {
23             int parent = parentIndex(index);
24
25             if (heap.get(index) <= heap.get(parent)) {
26                 break;
27             }
28
29             swap(index, parent);
30             index = parent;
31         }
32     }
33     // Public method: Remove and return max value
34     public int popMax() {
35         if (heap.isEmpty()) {
36             throw new IllegalStateException("Heap is empty");
37         }
38
39         int maxValue = heap.get(0);
40         int lastValue = heap.remove(heap.size() - 1);
41         if (!heap.isEmpty()) {
42             heap.set(0, lastValue);
43             heapifyDown(0);
44         }
45         return maxValue;
46     }
47     // Private helper: swap two elements
48     private void swap(int i, int j) {
49         int temp = heap.get(i);
50         heap.set(i, heap.get(j));
51         heap.set(j, temp);
52     }
53     // Private helper: heapify down after popMax
54     private void heapifyDown(int index) {
55         while (true) {
56             int largest = index;
57             int firstChild = firstChildIndex(index);
58
59             for (int i = 0; i < childrenCount; i++) {
60                 int childIndex = firstChild + i;
61
62                 if (childIndex >= heap.size()) {
63                     break;
64                 }
65
66                 if (heap.get(childIndex) > heap.get(largest)) {
67                     largest = childIndex;
68                 }
69             }
70
71             if (largest == index) {
72                 break;
73             }
74         }
75         swap(index, largest);
76     }
77 }
```

```
58
59         int maxValue = heap.get(0);
60         int lastValue = heap.remove(heap.size() - 1);
61         if (!heap.isEmpty()) {
62             heap.set(0, lastValue);
63             heapifyDown(0);
64         }
65         return maxValue;
66     }
67     // Private helper: swap two elements
68     private void swap(int i, int j) {
69         int temp = heap.get(i);
70         heap.set(i, heap.get(j));
71         heap.set(j, temp);
72     }
73     // Private helper: heapify down after popMax
74     private void heapifyDown(int index) {
75         while (true) {
76             int largest = index;
77             int firstChild = firstChildIndex(index);
78
79             for (int i = 0; i < childrenCount; i++) {
80                 int childIndex = firstChild + i;
81
82                 if (childIndex >= heap.size()) {
83                     break;
84                 }
85
86                 if (heap.get(childIndex) > heap.get(largest)) {
87                     largest = childIndex;
88                 }
89             }
90
91             if (largest == index) {
92                 break;
93             }
94         }
95         swap(index, largest);
96     }
97 }
```

```
67         largest = childIndex;
68     }
69 }
70 if (largest == index) {
71     break;
72 }
73 }
74 swap(index, largest);
75 index = largest;
76 }
77 }
78 }
79 // Private helper: calculate parent index
80 private int parentIndex(int index) {
81     return (index - 1) / childrenCount;
82 }
83
84 // Private helper: calculate first child index
85 private int firstChildIndex(int index) {
86     return index * childrenCount + 1;
87 }
88 }
89
90 // Optional class for testing (not required by Walmart)
91 public class Main {
92     public static void main(String[] args) {
93         PowerOfTwoMaxHeap heap = new PowerOfTwoMaxHeap(2); // 2^2 = 4 children
94         heap.insert(10);
95         heap.insert(50);
96         heap.insert(30);
97         heap.insert(40);
98
99         System.out.println(heap.popMax()); // 50
100        System.out.println(heap.popMax()); // 40
101    }
102 }
103 }
```