🛠️ Heap Interview Quick-Check Pattern

A tactical cheat sheet to solve heap-based problems with confidence and clarity.

---

1. 🧠 When to Use a Heap

Ask yourself:

- ❓ Do I need to constantly track the **min** or **max** value?
- ❓ Do I need to always get the **top K smallest/largest** elements?
- ❓ Is the data **dynamic** and growing/shrinking?
- ❓ Is the problem related to **priority**, **frequency**, or **distance**?

> 💡 Heaps shine when you need **efficient access to the smallest/largest elements on the fly**.

---

2. 🔍 Common Use Cases for Heaps

Problem Type	Use Heap To…
Top K Frequent Elements	Use Max Heap or Min Heap with size K
Merge K Sorted Lists	Min Heap based on current node values
Kth Largest / Smallest Element	Min/Max Heap of size K
Sliding Window Maximum	Max Heap with index management or deque
Median of Data Stream	Min Heap + Max Heap to balance halves
Schedule/Meeting Problems	Min Heap for end times
Huffman Encoding	Build tree with Min Heap

---

3. 📦 JavaScript Heap: How to Implement?

JavaScript doesn't have a native heap, so use a class:

```
class MinHeap {
  constructor() {
    this.heap = [];
  }

  insert(val) {
    this.heap.push(val);
    this.bubbleUp();
  }

  pop() {
    const top = this.heap[0];
    const end = this.heap.pop();
    if (this.heap.length > 0) {
      this.heap[0] = end;
      this.bubbleDown();
    }
    return top;
  }

  bubbleUp() {
    let i = this.heap.length - 1;
    while (i > 0) {
      let p = Math.floor((i - 1) / 2);
      if (this.heap[i] >= this.heap[p]) break;
      [this.heap[i], this.heap[p]] = [this.heap[p], this.heap[i]];
      i = p;
    }
  }

  bubbleDown() {
    let i = 0;
    const len = this.heap.length;
    while (true) {
      let left = 2 * i + 1, right = 2 * i + 2;
      let smallest = i;
```

```
      if (left < len && this.heap[left] < this.heap[smallest])
smallest = left;
      if (right < len && this.heap[right] < this.heap[smallest])
smallest = right;

      if (i === smallest) break;
      [this.heap[i], this.heap[smallest]] = [this.heap[smallest],
this.heap[i]];
      i = smallest;
    }
  }
}
```

> ✨ You can create a MaxHeap by inverting the sign (`-val`) when inserting and popping.

---

4. 🧪 Core Templates & Patterns

✅ Top K Frequent Elements

```
function topKFrequent(nums, k) {
  const freqMap = {};
  for (let n of nums) freqMap[n] = (freqMap[n] || 0) + 1;

  const heap = [];
  for (let key in freqMap) {
    heap.push([freqMap[key], key]);
  }

  heap.sort((a, b) => b[0] - a[0]); // max heap via sort
  return heap.slice(0, k).map(([_, val]) => val);
}
```

✅ Kth Largest in Stream

```
class KthLargest {
  constructor(k, nums) {
    this.k = k;
    this.heap = [];
    for (let n of nums) this.add(n);
  }

  add(val) {
    this.heap.push(val);
    this.heap.sort((a, b) => a - b); // simulate min-heap
    if (this.heap.length > this.k) this.heap.shift();
    return this.heap[0];
  }
}
```

✅ Merge K Sorted Lists (Min Heap)

```
function mergeKLists(lists) {
  const heap = [];
  for (let node of lists) {
    if (node) heap.push(node);
  }

  heap.sort((a, b) => a.val - b.val);
  const dummy = new ListNode(0);
  let curr = dummy;

  while (heap.length) {
    const node = heap.shift();
    curr.next = node;
    curr = curr.next;
    if (node.next) {
      heap.push(node.next);
      heap.sort((a, b) => a.val - b.val);
    }
  }
```

```
  return dummy.next;
}
```

---

5. 🧱 Edge Cases & Gotchas

- Heap of objects → ensure correct **comparator function**
- Pop from empty heap
- JavaScript has **no native Heap** — implement or use sort as workaround
- Maintain **heap size K** — don't grow indefinitely
- Duplicates in input — may affect frequency counts
- For `topK`, always check if it's **Kth largest or K largest values**

🧠 Think: *Do I need to keep all data? Or only the "top" K elements?*

---

6. 🧠 Mental Model for Heap Problems

Trigger words for Heap:

Problem asks for...	You probably need...
"Top K"	Min/Max Heap of size K
"Keep max/min while streaming"	Min/Max Heap
"Merge multiple sorted things"	Min Heap for smallest element
"Continuous comparison of size"	Dual Heaps (min + max)
"Track end times or priorities"	Min Heap based on time/value

🔁 Problem Solving Loop

1. 🎯 Is this a **top K / min K / Kth value** problem?
2. 💼 Do I need to **track values dynamically**?

3. 📦 Can I use a **fixed-size heap** to save memory?
4. 🧪 Do I need a **custom comparator**?
5. 🔍 Can sorting solve it or is heap more efficient?

✅ Final Interview Checklist

- Is this a Min Heap or Max Heap problem?
- Do I need to implement the heap or simulate with sorting?
- Can I use a **heap of size K** instead of storing all values?
- Is the comparison based on value, frequency, or custom priority?
- Any edge cases like empty input, large data, or duplicates?