# ⏱️ Intervals Interview Quick-Check Pattern

*A visual strategy guide for solving interval-based problems in coding interviews.*

---

## 1. 🧠 When to Think in Intervals

Ask yourself:

- ❓ Is input a list of **[start, end]** pairs?
- ❓ Do you need to **merge**, **insert**, or **find overlaps**?
- ❓ Are you comparing **gaps**, **coverage**, or **conflicts**?
- ❓ Are you asked for **maximum usage**, **free time**, or **minimum resources**?

> 💡 Interval problems are all about **sorting**, **sweeping**, and **merging**.

---

## 2. 🔍 Most Common Interval Patterns

| Problem Type | Strategy to Use |
|---|---|
| Merge overlapping intervals | Sort + Merge |
| Insert interval | Scan + Merge |
| Meeting rooms / overlaps | Min-heap or timeline sweep |
| Interval intersections | Two pointers |
| Free time between intervals | Complement intervals or sweep line |
| Maximum overlap | Timeline sweep or min heap |

---

## 3. 🔧 Core Templates

---

### ✅ Merge Intervals

```
function merge(intervals) {
  if (!intervals.length) return [];
  intervals.sort((a, b) => a[0] - b[0]);

  const res = [intervals[0]];
  for (let i = 1; i < intervals.length; i++) {
    const last = res[res.length - 1];
    const [start, end] = intervals[i];
    if (start <= last[1]) {
      last[1] = Math.max(last[1], end);
    } else {
      res.push([start, end]);
    }
  }
  return res;
}
```

---

### ✅ Insert Interval

```
function insert(intervals, newInterval) {
  const res = [], [newStart, newEnd] = newInterval;
  let i = 0;

  // Add all intervals before newInterval
  while (i < intervals.length && intervals[i][1] < newStart) {
    res.push(intervals[i++]);
  }

  // Merge all overlapping
  while (i < intervals.length && intervals[i][0] <= newEnd) {
    newInterval[0] = Math.min(newStart, intervals[i][0]);
    newInterval[1] = Math.max(newEnd, intervals[i][1]);
    i++;
```

```
  }

  res.push(newInterval);

  // Add rest
  while (i < intervals.length) res.push(intervals[i++]);

  return res;
}
```

---

### ✅ Minimum Meeting Rooms (Heap)

```
function minMeetingRooms(intervals) {
  if (!intervals.length) return 0;
  intervals.sort((a, b) => a[0] - b[0]);

  const heap = []; // stores end times
  heap.push(intervals[0][1]);

  for (let i = 1; i < intervals.length; i++) {
    if (intervals[i][0] >= heap[0]) heap.shift(); // room reused
    heap.push(intervals[i][1]);
    heap.sort((a, b) => a - b);
  }

  return heap.length;
}
```

---

### ✅ Interval Intersection (Two Pointers)

```
function intervalIntersection(firstList, secondList) {
  let i = 0, j = 0, res = [];

  while (i < firstList.length && j < secondList.length) {
    const [a1, a2] = firstList[i];
    const [b1, b2] = secondList[j];
    const start = Math.max(a1, b1);
    const end = Math.min(a2, b2);
```

```
    if (start <= end) res.push([start, end]);

    if (a2 < b2) i++;
    else j++;
  }

  return res;
}
```

---

## 4. 🧱 Edge Cases to Watch For

- Overlapping vs touching (`[1,3]` and `[3,5]`)
- Empty input
- Fully nested intervals
- One interval is much larger than others
- Negative intervals or zero length
- Intervals crossing midnight/zero (time zones)

> 🧠 Always draw overlapping timelines to visualize edge cases!

---

## 5. 🧠 Mental Model for Interval Problems

| Question Pattern | Matching Strategy |
|---|---|
| “Merge overlapping intervals” | Sort by start, merge if overlap |
| “Insert one interval” | Linear scan + merge |
| “How many rooms/resources needed?” | Min Heap of end times |
| “Where do overlaps occur?” | Two pointers |
| “Free time / no overlap window” | Invert merged intervals |

| | |
|---|---|
| “What’s the max usage?” | Sweep line algorithm |

---

## 🔁 Problem Solving Loop

1. 🔍 Should I **sort by start or end**?
2. 🔧 Do I **merge or track usage**?
3. ⌚ Is this about **time windows**, **conflicts**, or **availability**?
4. 📦 Can I use a **heap**, **scan**, or **prefix diff**?
5. 🧠 Have I drawn sample timelines to visualize the question?

---

## ✅ Final Interview Checklist

- Did I **sort** intervals before processing?
- Am I **merging correctly** (start ≤ last end)?
- Am I counting **overlaps or gaps**?
- Have I handled **edge cases** like exact-touching intervals?
- Did I choose the **right pattern**: scan, heap, pointers, or sweep?