Heaps & Priority Queues in DSA: Beginner to Advanced Guide

Subtitle: Learn how to efficiently manage dynamic datasets using heaps and priority queues for sorting and optimization problems.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Heaps are specialized binary trees used to maintain **max or min order**. They are the foundation for **priority queues, heap sort, and efficient selection problems**. This guide covers heap structures, operations, and practical applications.

Step 1: What is a Heap?

- **Binary Heap:** Complete binary tree with heap property
  - **Max Heap:** Parent ≥ children
  - **Min Heap:** Parent ≤ children
- Represented as **array** for efficient indexing
- **Applications:** Priority queue, heap sort, k-largest/smallest elements

Example: Array representation of max heap [90, 15, 10, 7, 12, 2]

Step 2: Heap Operations

1. **Insert:** Add element at end → bubble up to maintain heap property
2. **Extract (pop):** Remove root → replace with last element → bubble down
3. **Peek:** Return root element (max/min)
4. **Heapify:** Convert array into heap in O(n)

Example (Insert in Min Heap):

```
function insert(heap, val) {

 heap.push(val);

 let i = heap.length-1;

 while(i>0) {

  let parent = Math.floor((i-1)/2);

  if(heap[parent] > heap[i]) [heap[parent], heap[i]] = [heap[i], heap[parent]];
```

```
    else break;

    i = parent;

  }

}
```

Step 3: Common Heap Problems

1. **Heap Sort** → O(n log n)
2. **Kth Largest / Smallest Element**
3. **Merge K Sorted Arrays**
4. **Top K Frequent Elements**
5. **Median of Stream** → using two heaps

Step 4: Priority Queue

- Abstract data structure implemented using heap
- Supports: **enqueue(priority, value), dequeue()**
- Elements removed in order of **priority**
- Applications: Dijkstra's algorithm, task scheduling, event simulation

Example (JS Priority Queue with Min Heap):

```
class PriorityQueue {

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

  enqueue(val) { insert(this.heap, val); }

  dequeue() { return extractMin(this.heap); } // implement extractMin similar to insert

}
```

Step 5: Common Mistakes

- Forgetting to maintain heap property during insert/extract
- Confusing **array indices** for parent/children
- Using heap for unsorted array without heapify
- Not handling edge cases (empty heap)

Step 6: Practice Plan for Heaps

Day 1:

- Build min/max heap, insert, extract → 5 problems

Day 2:

- Heap sort → 5 problems

Day 3:

- Kth largest/smallest element → 5 problems

Day 4:

- Top K frequent elements → 5 problems

Day 5:

- Median of stream / merge K sorted arrays → 5 problems

Day 6:

- Mixed heap/priority queue problems → 10 problems

Mini Exercises

1. Implement max heap insert and extract
2. Find K smallest elements in an array
3. Merge multiple sorted linked lists using heap
4. Median of a dynamic stream of numbers

Key Takeaways

- Heaps maintain **partial order** for efficient retrieval
- Priority queues are implemented using **min/max heaps**
- Heaps are crucial for **sorting, selection, and optimization problems**
- Practice strengthens **indexing, bubbling, and heap operations intuition**

Visit haas.dev for more DSA heap guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app