Heap Patterns Playbook

High-Impact Priority Queue Templates

Updated November 5, 2025

Curated from Blind 75, NeetCode 150, and top company sets

Contents

1 Heap Fundamentals

Master min-heaps, simulate max-heaps, and keep comparison keys tidy to spin up solution scaffolds quickly.

1.1 Min-Heap Quick Reference

Python's 'heapq' is a binary min-heap. Build helpers for clarity. **Complexity:** Push/pop $O(\log n)$, peek $O(1)$.

```python
import heapq

class MinHeap:
    def __init__(self, data=None):
        self.heap = list(data or [])
        heapq.heapify(self.heap)

    def push(self, val):
        heapq.heappush(self.heap, val)

    def pop(self):
        return heapq.heappop(self.heap)

    def peek(self):
        return self.heap[0]

    def __len__(self):
        return len(self.heap)
```

Example Problems:

- 703. Kth Largest Element in a Stream (Blind 75)
- 973. K Closest Points to Origin (NeetCode 150)
- 1046. Last Stone Weight (NeetCode 150 variant)
- 1130. Minimum Cost Tree From Leaf Values (Top Amazon)

1.2 Max-Heap Simulation

Negate values or wrap tuples to flip ordering.

```python
import heapq

class MaxHeap:
    def __init__(self, data=None):
        self.heap = [(-val, val) for val in (data or [])]
        heapq.heapify(self.heap)

    def push(self, val):
        heapq.heappush(self.heap, (-val, val))

    def pop(self):
        return heapq.heappop(self.heap)[1]
```

Example Problems:

- 215. Kth Largest Element in an Array (Blind 75)
- 621. Task Scheduler (Top Amazon)
- 767. Reorganize String (Top Google)
- 264. Ugly Number II (Top Meta)

1.3 Heapify from Iterable

‘heapq.heapify‘ in-place: linear-time build makes repeated pushes cheaper.

```python
import heapq

def heapify_array(nums):
    heapq.heapify(nums)
    return nums
```

Example Problems:

- 347. Top K Frequent Elements (Blind 75, NeetCode 150)
- 692. Top K Frequent Words (NeetCode 150)
- 502. IPO (Top Google)
- 218. The Skyline Problem (Top Amazon)

2 💎 Selection Patterns

Use heaps to isolate the smallest or largest k elements, or to rank items by custom scores.

2.1 Kth Element (Quickselect Backup)

Maintain a bounded heap for predictable $O(n \log k)$ when input is adversarial.

```python
import heapq

def kth_largest(nums, k):
    heap = nums[:k]
    heapq.heapify(heap)
    for num in nums[k:]:
        if num > heap[0]:
            heapq.heapreplace(heap, num)
    return heap[0]
```

Example Problems:

- 215. Kth Largest Element in an Array (Blind 75)
- 973. K Closest Points to Origin (NeetCode 150)
- 787. Cheapest Flights Within K Stops (Top Amazon)
- 719. Find K-th Smallest Pair Distance (Top Google)

2.2 Top-K by Frequency

Push frequency pairs; break ties lexicographically when needed.

```python
import heapq
from collections import Counter

def top_k_frequent(nums, k):
    counter = Counter(nums)
    heap = [(-freq, val) for val, freq in counter.items()]
    heapq.heapify(heap)
    return [heapq.heappop(heap)[1] for _ in range(k)]
```

Example Problems:

- 347. Top K Frequent Elements (Blind 75)
- 692. Top K Frequent Words (NeetCode 150)
- 451. Sort Characters By Frequency (NeetCode 150)
- 358. Rearrange String k Distance Apart (Top Amazon)

2.3 K-Way Merge of Sorted Sources

Keep next candidate from each list; add the successor after popping.

```python
import heapq

# ListNode definition provided by the online judge
def merge_k_lists(lists):
    heap = []
    for idx, head in enumerate(lists):
        if head:
            heapq.heappush(heap, (head.val, idx, head))

    dummy = tail = ListNode(0)
    while heap:
        _, idx, node = heapq.heappop(heap)
        tail.next = node
        tail = tail.next
        if node.next:
            heapq.heappush(heap, (node.next.val, idx, node.next))
    return dummy.next
```

Example Problems:

- 23. Merge k Sorted Lists (Blind 75)
- 373. Find K Pairs with Smallest Sums (NeetCode 150)
- 632. Smallest Range Covering Elements from K Lists (Top Google)
- 378. Kth Smallest Element in a Sorted Matrix (NeetCode 150)

3 💧 Streaming & Sliding Window

Two-heaps and lazy deletion keep order statistics online as streams evolve.

3.1 Running Median with Two Heaps

Balance max-heap (lower half) and min-heap (upper half).

```python
import heapq

class MedianStream:
    def __init__(self):
        self.low = []   # max-heap via negatives
        self.high = []  # min-heap

    def add(self, num):
        heapq.heappush(self.low, -num)
        if self.low and self.high and -self.low[0] > self.high[0]:
            heapq.heappush(self.high, -heapq.heappop(self.low))
        if len(self.low) > len(self.high) + 1:
            heapq.heappush(self.high, -heapq.heappop(self.low))
        if len(self.high) > len(self.low):
            heapq.heappush(self.low, -heapq.heappop(self.high))

    def median(self):
        if len(self.low) > len(self.high):
            return float(-self.low[0])
        return (-self.low[0] + self.high[0]) / 2
```

Example Problems:

- 295. Find Median from Data Stream (Blind 75)
- 480. Sliding Window Median (Hard, NeetCode 150)
- 2208. Minimum Operations to Halve Array Sum (Top Amazon)
- 502. IPO (variant balancing profits)

3.2 Kth Largest in Stream

Keep a size-k min-heap; pop when exceeding capacity.

```python
import heapq

class KthLargest:
    def __init__(self, k, nums):
        self.k = k
        self.heap = nums
        heapq.heapify(self.heap)
        while len(self.heap) > k:
            heapq.heappop(self.heap)

    def add(self, val):
        if len(self.heap) < self.k:
            heapq.heappush(self.heap, val)
        elif val > self.heap[0]:
            heapq.heapreplace(self.heap, val)
        return self.heap[0]
```

Example Problems:

- 703. Kth Largest Element in a Stream (Blind 75)
- 1046. Last Stone Weight (NeetCode 150)
- 215. Kth Largest Element in an Array (stream variant)
- 871. Minimum Number of Refueling Stops (Top Google)

3.3 Sliding Window Maximum via Lazy Deletion

Use max-heap storing ‘(-value, index)‘ and drop stale entries when they fall outside the window.

```python
import heapq

def max_sliding_window(nums, k):
    heap, result = [], []
    for i, val in enumerate(nums):
        heapq.heappush(heap, (-val, i))
        if i >= k - 1:
            while heap and heap[0][1] <= i - k:
                heapq.heappop(heap)
            result.append(-heap[0][0])
    return result
```

Example Problems:

- 239. Sliding Window Maximum (Blind 75)
- 480. Sliding Window Median (NeetCode 150)
- 1438. Longest Continuous Subarray With Absolute Diff $\leq$ Limit (Top Amazon)
- 295. Find Median from Data Stream (support routine)

4 ⏰ Scheduling & Greedy Decisions

Heaps prioritize upcoming work, track capacity, and optimize reward accumulation.

4.1 Interval Room Count

Sort starts; push end times to count concurrent meetings.

```python
import heapq

def min_meeting_rooms(intervals):
    intervals.sort()
    heap = []
    for start, end in intervals:
        if heap and heap[0] <= start:
            heapq.heapreplace(heap, end)
        else:
            heapq.heappush(heap, end)
    return len(heap)
```

Example Problems:

- 253. Meeting Rooms II (NeetCode 150)
- 1851. Minimum Interval to Include Each Query (Top Google)
- 1094. Car Pooling (Top Amazon)
- 1353. Maximum Number of Events That Can Be Attended (Top Facebook)

4.2 Greedy Pick with Profit and Capital

Two heaps: one for available projects by profit, one for locked projects by capital.

```python
import heapq

def find_maximized_capital(k, w, profits, capital):
    projects = sorted(zip(capital, profits))
    available, idx = [], 0
    for _ in range(k):
        while idx < len(projects) and projects[idx][0] <= w:
            heapq.heappush(available, -projects[idx][1])
            idx += 1
        if not available:
            break
        w -= heapq.heappop(available)
    return w
```

Example Problems:

- 502. IPO (NeetCode 150)
- 630. Course Schedule III (Top Google)
- 857. Minimum Cost to Hire K Workers (Top Amazon)
- 2542. Maximum Subsequence Score (Top Amazon)

4.3 Median Cost Scheduling

Track both halves to adjust costs as tasks arrive.

```python
import heapq

def min_total_distance(positions):
    left, right = [], []
    median = positions[0]
    cost = 0
    for pos in positions:
        if pos <= median:
            heapq.heappush(left, -pos)
        else:
            heapq.heappush(right, pos)
        # rebalance
        if len(left) > len(right) + 1:
            heapq.heappush(right, -heapq.heappop(left))
        if len(right) > len(left):
            heapq.heappush(left, -heapq.heappop(right))
        median = -left[0]
        cost += abs(pos - median)
    return cost
```

Example Problems:

- 846. Hand of Straights (NeetCode 150 helper)
- 1705. Maximum Number of Eaten Apples (Top Apple)
- 1882. Process Tasks Using Servers (Top Amazon)
- 2402. Meeting Rooms III (Top Meta)

5 🧭 Best-First Search

Heaps drive best-first traversal when you always expand the most promising state next.

5.1 Weighted Grid Path (Dijkstra Variant)

Standard template; tweak scoring function for probability or minimax goals.

```python
import heapq

def best_first(grid):
    rows, cols = len(grid), len(grid[0])
    heap = [(grid[0][0], 0, 0)]
    dist = {(0, 0): grid[0][0]}
    while heap:
        cost, r, c = heapq.heappop(heap)
        if (r, c) == (rows - 1, cols - 1):
            return cost
        for dr, dc in ((1, 0), (-1, 0), (0, 1), (0, -1)):
            nr, nc = r + dr, c + dc
            if 0 <= nr < rows and 0 <= nc < cols:
                new_cost = cost + grid[nr][nc]
                if new_cost < dist.get((nr, nc), float('inf')):
                    dist[(nr, nc)] = new_cost
                    heapq.heappush(heap, (new_cost, nr, nc))
```

Example Problems:

- 1631. Path With Minimum Effort (Blind 75)
- 1514. Path With Maximum Probability (NeetCode 150 – flip to max-heap)
- 505. The Maze II (Top Google)
- 778. Swim in Rising Water (Blind 75)

5.2 Best-First Search with Heuristic Key

Push `(priority, state_id, state)` to encode tie-breakers.

```python
import heapq
import itertools

def a_star(start_state, heuristic, expand, goal_check):
    counter = itertools.count()
    heap = [(heuristic(start_state), 0, next(counter), start_state)]
    seen = {start_state: 0}
    while heap:
        est_total, cost, _, state = heapq.heappop(heap)
        if goal_check(state):
            return cost
        if cost > seen[state]:
            continue
        for nxt, step_cost in expand(state):
            new_cost = cost + step_cost
            if new_cost < seen.get(nxt, float('inf')):
                seen[nxt] = new_cost
                priority = new_cost + heuristic(nxt)
                heapq.heappush(heap, (priority, new_cost, next(counter), nxt))
    return -1
```

Example Problems:

- 773. Sliding Puzzle (Top Google)
- 847. Shortest Path Visiting All Nodes (NeetCode 150)
- 1293. Shortest Path in a Grid with Obstacles Elimination (NeetCode 150)
- 2290. Minimum Obstacle Removal to Reach Corner (Top Amazon)

6 Interview Playbook

Rapid checks for when to reach for heaps and how to articulate trade-offs.

6.1 Decision Guide

- Need the best/current worst element repeatedly? Use min/max-heap.
- Streaming order statistics? Pair min/max heaps (median, percentile).
- K-way merge or multi-source lists? Push ‘(value, origin, data)‘ tuples.
- Greedy scheduling or picking top rewards? Sort by unlock time, heap by payoff.
- Weighted best-first traversal? Dijkstra/A* with ‘(distance, state)‘ heaps.
- Memory tight? Keep heap size bounded at k.
- Need deletions? Combine heap with hash map counts for lazy removal.

6.2 Rapid Practice Sets

- **Blind 75 Heap Core:** 215, 239, 295, 347, 23, 973, 1046.
- **NeetCode 150 Heap Tier:** 23, 253, 295, 347, 373, 378, 480, 502, 621, 692, 703, 767, 857, 973, 1046, 1405, 1675.
- **Top Company Favourites:** 218, 239, 253, 355, 502, 621, 632, 745, 778, 857, 871, 1882, 2187, 2296.
- **Advanced Hard Hitters:** 3015, 1675, 1834, 2208, 2542, 2599, 2654.