Heaps & Priority Queues: Comprehensive Notes with Diagrams, Code (Python & C++), and LeetCode Examples

1. Introduction to Heaps & Priority Queues

A **heap** is a special complete binary tree-based data structure that satisfies the *heap property*:

- **Max Heap**: Every parent node is **greater than or equal** to its children.
- **Min Heap**: Every parent node is **less than or equal** to its children.

A **priority queue** is an abstract data type where each element has a "priority" associated with it. Elements are served based on priority (highest or lowest), not just insertion order. Heaps are the most efficient way to implement priority queues[1].

2. Heap Structure & Diagrams

Max Heap Example

```
       90
      /  \
    15    10
   / \   / \
  7  12 2   8
```

- Every parent is ≥ its children.

Min Heap Example

```
       10
      /  \
    15    20
   / \   / \
  30 40 50 100
```

- Every parent is ≤ its children.

Array Representation (for both):

- For node at index `i`: left child at `2i+1`, right child at `2i+2`, parent at `(i-1)//2`.

3. Heap Operations

A. Insertion

Algorithm:

1. Insert the new element at the end (bottom right).
2. "Bubble up" (heapify up): Compare with parent and swap if heap property is violated.
3. Repeat until heap property is restored[2] [3] [4].

Python (Min Heap)

```python
import heapq
heap = []
heapq.heappush(heap, value)
```

C++ (Max Heap)

```cpp
#include <queue>
std::priority_queue<int> maxHeap;
maxHeap.push(value);
```

Manual Implementation (Python)

```python
def insert(heap, value):
    heap.append(value)
    i = len(heap) - 1
    while i > 0 and heap[(i-1)//2] < heap[i]:  # For max heap
        heap[(i-1)//2], heap[i] = heap[i], heap[(i-1)//2]
        i = (i-1)//2
```

B. Deletion (Extract Min/Max)

Algorithm:

1. Remove the root (min or max).
2. Replace root with last element.
3. "Bubble down" (heapify down): Swap with the larger (max heap) or smaller (min heap) child if heap property is violated.
4. Repeat until heap property is restored[3] [4].

Python (Min Heap)

```
min_elem = heapq.heappop(heap)
```

C++ (Max Heap)

```
int max_elem = maxHeap.top();
maxHeap.pop();
```

Manual Implementation (Python)

```
def extract_max(heap):
    max_val = heap[^0]
    heap[^0] = heap.pop()
    i = 0
    while True:
        left = 2*i + 1
        right = 2*i + 2
        largest = i
        if left < len(heap) and heap[left] > heap[largest]:
            largest = left
        if right < len(heap) and heap[right] > heap[largest]:
            largest = right
        if largest == i:
            break
        heap[i], heap[largest] = heap[largest], heap[i]
        i = largest
    return max_val
```

C. Peek (Get Min/Max)

- Just return `heap` (root).

4. Heap Sort

Algorithm:

1. Build a max heap from the array.
2. Swap the root (max) with the last item, reduce heap size by 1.
3. Heapify the root.
4. Repeat until heap size is 1.

Time Complexity: O(n log n)
Space Complexity: O(1) (in-place)[5] [6] [7] [8] [9].

Python Implementation

```
def heapify(arr, n, i):
    largest = i
    l = 2*i + 1
    r = 2*i + 2
    if l < n and arr[l] > arr[largest]:
        largest = l
    if r < n and arr[r] > arr[largest]:
        largest = r
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)
    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n-1, 0, -1):
        arr[i], arr[^0] = arr[^0], arr[i]
        heapify(arr, i, 0)
```

C++ Implementation

```
void heapify(vector<int>& arr, int n, int i) {
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && arr[l] > arr[largest]) largest = l;
    if (r < n && arr[r] > arr[largest]) largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(vector<int>& arr) {
    int n = arr.size();
    for (int i = n/2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n-1; i > 0; i--) {
        swap(arr[^0], arr[i]);
        heapify(arr, i, 0);
    }
}
```

Diagram: Heap Sort Steps

```
Initial: [8, 5, 3, 9, 1]
Build Max Heap: [9, 5, 8, 3, 1]
Extract Max: [1, 5, 8, 3, 9]
Extract Max: [1, 3, 8, 5, 9]
Extract Max: [1, 3, 5, 8, 9]
Sorted: [1, 3, 5, 8, 9]
```

5. K Largest/Smallest Elements

Kth Largest Element

Approach 1: Max Heap

- Build a max heap and pop k times; kth pop is the answer[10] [11] [12].

Approach 2: Min Heap of Size K

- For Kth largest, maintain a min heap of size K.
- Insert elements; if size > K, pop the smallest.
- The root is the Kth largest.

Python Example (Kth Largest)

```
import heapq
def kth_largest(nums, k):
    return heapq.nlargest(k, nums)[-1]
```

C++ Example (Kth Largest)

```
#include <queue>
int kthLargest(vector<int>& nums, int k) {
    priority_queue<int, vector<int>, greater<int>> minHeap;
    for (int n : nums) {
        minHeap.push(n);
        if (minHeap.size() > k)
            minHeap.pop();
    }
    return minHeap.top();
}
```

Kth Smallest Element

Python Example (Kth Smallest)

```
import heapq
def kth_smallest(nums, k):
    return heapq.nsmallest(k, nums)[-1]
```

C++ Example (Kth Smallest)

```
#include <queue>
int kthSmallest(vector<int>& nums, int k) {
    priority_queue<int> maxHeap;
    for (int n : nums) {
        maxHeap.push(n);
        if (maxHeap.size() > k)
            maxHeap.pop();
    }
    return maxHeap.top();
}
```

6. Median in Data Stream

Problem:

Maintain the median as numbers arrive in a stream.

Optimal Solution: Two Heaps Approach

- **Max Heap** for lower half
- **Min Heap** for upper half
- Balance sizes so their difference is at most 1
- Median is:
  - If odd: root of larger heap
  - If even: average of roots[13] [14] [15] [16] [17]

Python Implementation

```
import heapq

class MedianFinder:
    def __init__(self):
        self.small = []  # Max heap (invert sign)
        self.large = []  # Min heap

    def addNum(self, num):
        heapq.heappush(self.small, -num)
        heapq.heappush(self.large, -heapq.heappop(self.small))
        if len(self.large) > len(self.small):
            heapq.heappush(self.small, -heapq.heappop(self.large))

    def findMedian(self):
        if len(self.small) > len(self.large):
            return -self.small[^0]
        return (-self.small[^0] + self.large[^0]) / 2.0
```

C++ Implementation

```
#include <queue>
class MedianFinder {
    priority_queue<int> maxHeap; // lower half
    priority_queue<int, vector<int>, greater<int>> minHeap; // upper half
public:
    void addNum(int num) {
        if (maxHeap.empty() || num <= maxHeap.top())
            maxHeap.push(num);
        else
            minHeap.push(num);
        if (maxHeap.size() < minHeap.size())
            maxHeap.push(minHeap.top()), minHeap.pop();
        else if (maxHeap.size() > minHeap.size() + 1)
            minHeap.push(maxHeap.top()), maxHeap.pop();
    }
    double findMedian() {
        if (maxHeap.size() == minHeap.size())
            return (maxHeap.top() + minHeap.top()) / 2.0;
        return maxHeap.top();
    }
};
```

Diagram:

```
Stream: [2, 3, 4]
Heaps after each insert:
Insert 2: maxHeap=[^2], minHeap=[]
Insert 3: maxHeap=[^2], minHeap=[^3]
Insert 4: maxHeap=[3,2], minHeap=[^4]
Median: 3
```

7. LeetCode Examples

A. 215. Kth Largest Element in an Array

Question:
Given an integer array nums and an integer k, return the kth largest element in the array.

Python Solution

```
import heapq
def findKthLargest(nums, k):
    return heapq.nlargest(k, nums)[-1]
```

C++ Solution

```
#include <queue>
int findKthLargest(vector<int>& nums, int k) {
    priority_queue<int, vector<int>, greater<int>> minHeap;
    for (int n : nums) {
        minHeap.push(n);
        if (minHeap.size() > k)
            minHeap.pop();
    }
    return minHeap.top();
}
```

B. 295. Find Median from Data Stream

Question:
The median is the middle value in an ordered integer list. If the size of the list is even, the median is the mean of the two middle values. Implement the MedianFinder class:

- `MedianFinder()` initializes the MedianFinder object.
- `void addNum(int num)` adds the integer num from the data stream to the data structure.
- `double findMedian()` returns the median of all elements so far.

Python Solution

```
import heapq
class MedianFinder:
    def __init__(self):
        self.small = []
        self.large = []

    def addNum(self, num):
        heapq.heappush(self.small, -num)
        heapq.heappush(self.large, -heapq.heappop(self.small))
        if len(self.large) > len(self.small):
            heapq.heappush(self.small, -heapq.heappop(self.large))

    def findMedian(self):
        if len(self.small) > len(self.large):
            return -self.small[^0]
        return (-self.small[^0] + self.large[^0]) / 2.0
```

C++ Solution

```
#include <queue>
class MedianFinder {
    priority_queue<int> maxHeap;
    priority_queue<int, vector<int>, greater<int>> minHeap;
public:
    void addNum(int num) {
        if (maxHeap.empty() || num <= maxHeap.top())
            maxHeap.push(num);
        else
            minHeap.push(num);
        if (maxHeap.size() < minHeap.size())
            maxHeap.push(minHeap.top()), minHeap.pop();
        else if (maxHeap.size() > minHeap.size() + 1)
            minHeap.push(maxHeap.top()), maxHeap.pop();
    }
    double findMedian() {
        if (maxHeap.size() == minHeap.size())
            return (maxHeap.top() + minHeap.top()) / 2.0;
        return maxHeap.top();
    }
};
```

8. Heap Variants

- **Min-Max Heap:** Supports both find-min and find-max in O(1) time[18].
- **d-ary Heap:** Each node has d children (used in some priority queues for performance tuning).
- **Pairing Heap, Fibonacci Heap:** Used for advanced algorithms (e.g., Dijkstra's).

9. Applications

- **Priority Scheduling** (OS, Networking)
- **Dijkstra's Shortest Path**
- **Median Maintenance**
- **Heap Sort**
- **Event Simulation**

10. Complexity Table

Operation	Min/Max Heap	Heap Sort
Insert	O(log n)	-
Delete Root	O(log n)	-
Build Heap	O(n)	O(n)
Heapify	O(log n)	O(log n)
Heap Sort	-	O(n log n)

11. Summary Table: Min Heap vs Max Heap

Feature	Min Heap	Max Heap
Root Value	Minimum	Maximum
Use Case	Kth Smallest, Median	Kth Largest, HeapSort
Python API	heapq (default)	Use negatives
C++ API	priority_queue + greater<>	priority_queue (default)

References:

- [DigitalOcean Min Heap][2]
- [BTechSmartClass Max Heap][3]
- [takeUforward Kth Element][10]
- [LeetCode 215][A]
- [LeetCode 295][B]
- [EnjoyAlgorithms Kth Smallest][11]
- [Baeldung Median Stream][13]
- [Wikipedia Heapsort][9]

This guide covers heap fundamentals, operations, sorting, Kth element, median maintenance, and real LeetCode problems with detailed code in Python and C++ for interviews and real-world applications.

⁂

1. https://www.shiksha.com/online-courses/articles/priority-queue-all-that-you-need-to-know/
2. https://www.digitalocean.com/community/tutorials/min-heap-binary-tree
3. http://www.btechsmartclass.com/data_structures/max-heap.html
4. https://www.programiz.com/dsa/heap-data-structure
5. https://www.tutorialspoint.com/data_structures_algorithms/heap_sort_algorithm.htm
6. https://www.sanfoundry.com/python-program-implement-heapsort/
7. https://www.scaler.com/topics/heap-sort-cpp/
8. https://www.hackerearth.com/practice/algorithms/sorting/heap-sort/tutorial/
9. https://en.wikipedia.org/wiki/Heapsort
10. https://takeuforward.org/data-structure/kth-largest-smallest-element-in-an-array/
11. https://www.enjoyalgorithms.com/blog/find-the-kth-smallest-element-in-an-array/
12. https://www.w3resource.com/python-exercises/heap-queue-algorithm/python-heapq-exercise-7.php
13. https://www.baeldung.com/java-stream-integers-median-using-heap
14. https://algo.monster/liteproblems/295
15. https://blog.heycoach.in/find-median-from-data-stream-solution-in-cpp/

16. https://www.youtube.com/watch?v=itmhHWaHupI
17. https://leetcode.com/problems/find-median-from-data-stream/
18. https://en.wikipedia.org/wiki/Min-max_heap