# Top K Elements Pattern

The 'Top K Elements' pattern is a common heap/priority queue technique used to efficiently find the K largest, smallest, most frequent, or closest elements without sorting the entire input.

## Core Idea

- Use a Min Heap (size K) when you want to keep the K largest elements.  
- Use a Max Heap (size K) when you want to keep the K smallest elements.  
- Use a HashMap + Heap when you need frequency-based ordering.  
  
This reduces complexity to around O(N log K), better than O(N log N) sorting when K << N.

## Generic Approaches

### 1. K Largest Elements

import heapq  
  
def kLargest(nums, k):  
 minHeap = []  
 for num in nums:  
 heapq.heappush(minHeap, num)  
 if len(minHeap) > k:  
 heapq.heappop(minHeap)  
 return minHeap

### 2. K Smallest Elements

import heapq  
  
def kSmallest(nums, k):  
 maxHeap = []  
 for num in nums:  
 heapq.heappush(maxHeap, -num)  
 if len(maxHeap) > k:  
 heapq.heappop(maxHeap)  
 return [-x for x in maxHeap]

### 3. Top K Frequent Elements

import heapq  
from collections import Counter  
  
def topKFrequent(nums, k):  
 freq = Counter(nums)  
 minHeap = []  
 for num, count in freq.items():  
 heapq.heappush(minHeap, (count, num))  
 if len(minHeap) > k:  
 heapq.heappop(minHeap)  
 return [num for count, num in minHeap]

### 4. K Closest Numbers

import heapq  
  
def kClosest(nums, k, x):  
 maxHeap = []  
 for num in nums:  
 dist = abs(num - x)  
 heapq.heappush(maxHeap, (-dist, num))  
 if len(maxHeap) > k:  
 heapq.heappop(maxHeap)  
 return [num for dist, num in maxHeap]

## Common Problems Using This Pattern

- Top K Frequent Elements (Leetcode 347)  
- Kth Largest Element in an Array (Leetcode 215)  
- K Closest Points to Origin (Leetcode 973)  
- Kth Smallest Number in a Sorted Matrix (Leetcode 378)  
- K Pairs with Smallest Sums (Leetcode 373)

## Example Solutions

### Leetcode 451. Sort Characters By Frequency

import heapq  
from collections import Counter  
  
class Solution:  
 def frequencySort(self, s: str) -> str:  
 freq = Counter(s)  
 maxHeap = [(-count, char) for char, count in freq.items()]  
 heapq.heapify(maxHeap)  
   
 result = []  
 while maxHeap:  
 count, char = heapq.heappop(maxHeap)  
 result.append(char \* (-count))  
   
 return "".join(result)

### Leetcode 973. K Closest Points to Origin

import heapq  
  
class Solution:  
 def kClosest(self, points: list[list[int]], k: int) -> list[list[int]]:  
 maxHeap = []  
 for (x, y) in points:  
 dist = x\*x + y\*y  
 heapq.heappush(maxHeap, (-dist, x, y))  
 if len(maxHeap) > k:  
 heapq.heappop(maxHeap)  
 return [[x, y] for (\_, x, y) in maxHeap]

## Boilerplate Code for Top K Elements Pattern

import heapq  
from collections import Counter  
  
def topKElements(nums, k, mode="largest", target=None):  
 if mode == "largest":  
 minHeap = []  
 for num in nums:  
 heapq.heappush(minHeap, num)  
 if len(minHeap) > k:  
 heapq.heappop(minHeap)  
 return minHeap  
   
 elif mode == "smallest":  
 maxHeap = []  
 for num in nums:  
 heapq.heappush(maxHeap, -num)  
 if len(maxHeap) > k:  
 heapq.heappop(maxHeap)  
 return [-x for x in maxHeap]  
   
 elif mode == "frequent":  
 freq = Counter(nums)  
 minHeap = []  
 for num, count in freq.items():  
 heapq.heappush(minHeap, (count, num))  
 if len(minHeap) > k:  
 heapq.heappop(minHeap)  
 return [num for count, num in minHeap]  
   
 elif mode == "closest":  
 if target is None:  
 raise ValueError("Target value required for closest mode.")  
 maxHeap = []  
 for num in nums:  
 dist = abs(num - target)  
 heapq.heappush(maxHeap, (-dist, num))  
 if len(maxHeap) > k:  
 heapq.heappop(maxHeap)  
 return [num for \_, num in maxHeap]