In [ ]:

In [4]:
```python
import heapq

class MinHeap:
    def __init__(self):
        self.heap = []

    def insert(self, element):
        heapq.heappush(self.heap, element)

    def delete(self, element):
        try:
            self.heap.remove(element)
            heapq.heapify(self.heap)
        except ValueError:
            print(f"Element {element} not found in the heap.")

    def get_min(self):
        if self.heap:
            return self.heap[0]
        raise IndexError("get_min from an empty heap")

class MaxHeap:
    def __init__(self):
        self.heap = []

    def insert(self, element):
        heapq.heappush(self.heap, -element)

    def delete(self, element):
        try:
            self.heap.remove(-element)
            heapq.heapify(self.heap)
        except ValueError:
            print(f"Element {element} not found in the heap.")

    def get_max(self):
        if self.heap:
            return -self.heap[0]
        raise IndexError("get_max from an empty heap")
```

In [ ]:

In [6]:

In [8]:
```python
# MinHeap
min_heap = MinHeap()
min_heap.insert(5)
min_heap.insert(3)
min_heap.insert(8)
print("Min:", min_heap.get_min())
min_heap.delete(3)
print("Min after deletion:", min_heap.get_min())

# MaxHeap
max_heap = MaxHeap()
max_heap.insert(5)
max_heap.insert(3)
max_heap.insert(8)
print("Max:", max_heap.get_max())
max_heap.delete(8)
print("Max after deletion:", max_heap.get_max())
```

```
Min: 3
Min after deletion: 5
Max: 8
Max after deletion: 5
```

In [10]:
```python
import heapq
import random
import time

def k_largest_elements(nums, k):
    if k <= 0:
        return []
    return heapq.nlargest(k, nums)

# Brute-force
def k_largest_elements_brute_force(nums, k):
    if k <= 0:
```

```
        return []
    return sorted(nums, reverse=True)[:k]


def test_k_largest_functions():

    large_dataset = [random.randint(1, 1000000) for _ in range(100000)]
    k = 10


    start_time = time.time()
    heap_result = k_largest_elements(large_dataset, k)
    heap_time = time.time() - start_time


    start_time = time.time()
    brute_force_result = k_largest_elements_brute_force(large_dataset, k)
    brute_force_time = time.time() - start_time


    print(f"K largest elements using heap: {heap_result}")
    print(f"Time taken using heap: {heap_time:.6f} seconds")

    print(f"K largest elements using brute-force: {brute_force_result}")
    print(f"Time taken using brute-force: {brute_force_time:.6f} seconds")


test_k_largest_functions()
```

```
K largest elements using heap: [999999, 999995, 999989, 999987, 999984, 999984, 999979, 999977, 999954, 999932]
Time taken using heap: 0.001569 seconds
K largest elements using brute-force: [999999, 999995, 999989, 999987, 999984, 999984, 999979, 999977, 999954, 9
99932]
Time taken using brute-force: 0.020637 seconds
```