Heap

Created by	j jubink
Created time	@February 26, 2025 10:27 PM
Tags	

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

    def parent(self,index):
        return(index-1)//2 if index > 0 else None

    def left_child(self,index):
        return 2 * index + 1

    def right_child(self,index):
        return 2 * index + 2

    def insert(self,value):
        self.heap.append(value)
        self.heapify_up(len(self.heap)-1)

    def heapify_up(self,index):
        parent = self.parent(index)
        while parent is not None and self.heap[parent] > self.heap[index]:
            self.heap[parent], self.heap[index] = self.heap[index], self.heap[parent]
            index = parent
            parent = self.parent(index)

    def extract_min(self):
        if not self.heap:
            return None
        if len(self.heap) == 1:
            return self.heap.pop()
        # get the minimum element
        root_value = self.heap[0]
        # move lasst element to the root
        self.heap[0] = self.heap.pop()
        # restore heap property
        self.heapify_down(0)
        return root_value

    def heapify_down(self,index):
        smallest = index
        left = self.left_child(index)
        right = self.right_child(index)

        if left < len(self.heap) and self.heap[left] < self.heap[smallest]:
            smallest = left
        if right < len(self.heap) and self.heap[right] < self.heap[smallest]:
            smallest = right
        if smallest != index:
            self.heap[index], self.heap[smallest] = self.heap[smallest],self.heap[index
            self.heapify_down(smallest)

    def delete(self,value):
        try:
            index = self.heap.index(value)
            self.heap[index] = self.heap.pop()
            self.heapify_down(index)
        except ValueError:
            print("value not found in the heap")

    def heapify(self,list1):
        self.heap = list1[:]
        for i in range(len(self.heap) // 2, -1, -1):
            self.heapify_down(i)
    def heap_sort(self):
        sorted_list = []
        while self.heap:
            sorted_list.append(self.extract_min())
        return sorted_list

    def display(self):
        print(self.heap)


heap = BinaryHeap()
nums = [10,1,5,20,3]

heap.heapify(nums)
print("Heapified array:",heap.heap)

heap.insert(2)
heap.insert(8)
heap.display()

print("Extracted min:",heap.extract_min())
heap.display()

heap.delete(5)
heap.display()


sorted_array  = heap.heap_sort()
print("Sorted array:",sorted_array)
```

🔥 Heap Data Structure - Practice Questions

A **heap** is a **complete binary tree** used for priority-based operations, mainly in **Heap Sort, Priority Queues, and Dijkstra's Algorithm**.

- **Min Heap**: Parent ≤ Children
- **Max Heap**: Parent ≥ Children

Each level contains:

✅ **Step-by-step algorithm**

✅ **Python implementation**

🟢 Beginner Level (Heap Fundamentals)

1. Insert an element into a Heap

📌 **Concept**:

- Insert at the end and **heapify up** to maintain heap order.

Algorithm (Min Heap):

1. Insert the new element at the **end**.
2. Compare with **parent** and **swap** if smaller.
3. Repeat until the heap property is restored.

Python Code

```
python
CopyEdit
import heapq

min_heap = []
heapq.heappush(min_heap, 10)
heapq.heappush(min_heap, 5)
heapq.heappush(min_heap, 20)
```

```
print("Heap after insertion:", min_heap)  # Output: [5, 10, 20]
```

2. Delete the Root from a Heap

📌 **Concept**:

- Replace root with the **last element**, then **heapify down**.

Algorithm (Min Heap):

1. Swap **root** with **last node** and remove it.
2. Compare **new root** with children.
3. Swap with the **smallest child** if needed.
4. Repeat until heap property is restored.

Python Code

```
python
CopyEdit
heapq.heappop(min_heap)  # Removes the smallest element
print("Heap after deletion:", min_heap)  # Output: [10, 20]
```

3. Find the Minimum or Maximum Element in a Heap

📌 **Concept**:

- **Min Heap** → Root is the **minimum**.
- **Max Heap** → Root is the **maximum**.

Python Code (Min Heap)

```
python
CopyEdit
```

```
print("Minimum element:", min_heap[0])  # Output: 10
```

4. Convert an Array into a Heap

📌 **Concept**:

- Use **heapify()** to **convert an unordered array into a valid heap**.

Python Code

```
python
CopyEdit
arr = [3, 1, 4, 1, 5, 9]
heapq.heapify(arr)
print("Heap from array:", arr)  # Output: [1, 1, 4, 3, 5, 9]
```

🟡 Intermediate Level (Heap Operations & Applications)

5. Implement Heap Sort

📌 **Concept**:

- Convert array → **Heapify** → **Extract Min** repeatedly.

Python Code

```
python
CopyEdit
def heap_sort(arr):
    heapq.heapify(arr)
    return [heapq.heappop(arr) for _ in range(len(arr))]

arr = [5, 3, 8, 4, 2]
```

```
print("Sorted array:", heap_sort(arr))  # Output: [2, 3, 4, 5, 8]
```

6. Merge K Sorted Lists Using a Heap

📌 **Concept**:

- Insert first element of **each list** into a heap.
- Extract **smallest**, insert next element from that list.

Python Code

```
python
CopyEdit
lists = [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
merged_list = list(heapq.merge(*lists))
print("Merged list:", merged_list)  # Output: [1,2,3,4,5,6,7,8,9]
```

7. Find Kth Smallest or Kth Largest Element

📌 **Concept**:

- **Min Heap** → Extract `K` times for Kth smallest.
- **Max Heap** → Store negative values to simulate max heap.

Python Code

```
python
CopyEdit
arr = [7, 10, 4, 3, 20, 15]
heapq.heapify(arr)
k = 3
for _ in range(k - 1):
    heapq.heappop(arr)
```

```
print(f"{k}th smallest element:", arr[0])  # Output: 7
```

🔴 Advanced Level (Heap-Based Algorithms & Variants)

8. Implement a Max Heap using a Min Heap

📌 **Concept**:

- Store **negative values** in a Min Heap to **simulate Max Heap**.

Python Code

```
python
CopyEdit
max_heap = []
heapq.heappush(max_heap, -10)
heapq.heappush(max_heap, -5)
heapq.heappush(max_heap, -20)

print("Max element:", -heapq.heappop(max_heap))  # Output: 20
```

9. Implement a Priority Queue Using a Heap

📌 **Concept**:

- Priority Queue maintains **priority-based ordering**.

Python Code

```
python
CopyEdit
tasks = [(2, "low priority"), (1, "high priority"), (3, "medium priority")]
heapq.heapify(tasks)
```

```
while tasks:
    print(heapq.heappop(tasks))  # Processes high → medium → low priority
```

---

10. Find the Median of a Stream Using Two Heaps

📌 **Concept**:

- Maintain **two heaps**:
    - **Max Heap (left half)**
    - **Min Heap (right half)**
- **Balance heaps** to keep medians accurate.

Python Code

```
python
CopyEdit
import heapq

left_half = []  # Max Heap (negative values)
right_half = []  # Min Heap

def insert(num):
    if not left_half or num <= -left_half[0]:
        heapq.heappush(left_half, -num)
    else:
        heapq.heappush(right_half, num)

    if len(left_half) > len(right_half) + 1:
        heapq.heappush(right_half, -heapq.heappop(left_half))
    elif len(right_half) > len(left_half):
        heapq.heappush(left_half, -heapq.heappop(right_half))

def find_median():
    if len(left_half) == len(right_half):
        return (-left_half[0] + right_half[0]) / 2
    return -left_half[0]

# Usage
insert(10)
insert(20)
insert(15)
print("Median:", find_median())  # Output: 15
```

📌 Heap Sort Practice Questions (Beginner to Advanced)

Heap Sort is a comparison-based sorting algorithm that uses a binary heap data structure. It works in **O(n log n)** time complexity and is widely used for in-place sorting.

🌟 Beginner Level (Basic Heap Sort Concepts)

1. **Sort an array using Heap Sort.**
2. **Sort an array in descending order using Heap Sort.**
3. **Find the k-th smallest element using Heap Sort.**
4. **Find the k-th largest element using Heap Sort.**
5. **Check if an array is sorted after applying Heap Sort.**
6. **Find the second smallest element using Heap Sort.**
7. **Find the second largest element using Heap Sort.**
8. **Sort an array containing duplicate values using Heap Sort.**
9. **Sort an array containing negative numbers using Heap Sort.**
10. **Sort an already nearly sorted array using Heap Sort efficiently.**

✅ **Example: Sorting an array using Heap Sort**

```
python
CopyEdit
def heap_sort(arr):
    heap = BinaryHeap()
    heap.heapify(arr)
    return heap.heap_sort()

arr = [4, 10, 3, 5, 1]
sorted_arr = heap_sort(arr)
print("Sorted Array:", sorted_arr)  # Output: [1, 3, 4, 5, 10]
```

🌟 Intermediate Level (Heap Sort Variants & Applications)

1. **Sort an array of strings using Heap Sort (lexicographically).**
2. **Sort an array of tuples based on the second element using Heap Sort.**
3. **Find the median of an unsorted array using Heap Sort.**
4. **Sort a linked list using Heap Sort.**
5. **Sort an array of floating-point numbers using Heap Sort.**
6. **Sort an array of large numbers (BigInteger) using Heap Sort.**
7. **Sort an array of intervals based on start times using Heap Sort.**
8. **Find the largest `k` elements from an array using Heap Sort.**
9. **Find the smallest `k` elements from an array using Heap Sort.**
10. **Sort an array where each element is at most `k` positions away from its sorted position (nearly sorted array).**

✅ **Example: Sorting an array of tuples**

```
python
CopyEdit
def sort_tuples(arr):
    heap = BinaryHeap()
    heap.heapify(arr)
    return heap.heap_sort()
```

```
arr = [(1, 'b'), (3, 'a'), (2, 'c')]
sorted_arr = sort_tuples(arr)
print("Sorted Tuples:", sorted_arr)  # Output: [(1, 'b'), (2, 'c'), (3, 'a')]
```

🌟 Advanced Level (Heap Sort Applications & Performance Analysis)

1. **Implement an in-place Heap Sort (without extra space).**
2. **Find the k most frequent elements using Heap Sort.**
3. **Sort an array where each element appears at most twice using Heap Sort.**
4. **Sort an array of job deadlines and profits using Heap Sort.**
5. **Sort a very large dataset using external Heap Sort (disk-based sorting).**
6. **Find the largest sum contiguous subarray using Heap Sort.**
7. **Sort an array of complex numbers based on magnitude using Heap Sort.**
8. **Sort an array of people based on height and weight using Heap Sort.**
9. **Sort an array based on frequency of elements using Heap Sort.**
10. **Implement a multi-threaded Heap Sort.**

✅ **Example: Finding the `k` most frequent elements**

```
python
CopyEdit
from collections import Counter
import heapq

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

```
arr = [1, 1, 1, 2, 2, 3]
k = 2
print("Top K Frequent Elements:", k_most_frequent(arr, k))  # Output: [1, 2]
```