

Day – 12 Heaps

Problem Statement: Min Heap & Max Heap

1. Min Heap

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

    def parent(self, i):
        return (i - 1) // 2

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

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

    def swap(self, i, j):
        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

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

    def extract_min(self):
        if len(self.heap) == 0:
            return None

        min_item = self.heap[0]
        self.swap(0, len(self.heap) - 1)
        self.heap.pop()
```

```
self.heapify_down(0)
return min_item
```

```
def heapify_up(self, i):
    while i > 0 and self.heap[i] < self.heap[self.parent(i)]:
        self.swap(i, self.parent(i))
        i = self.parent(i)
```

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

```
    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 != i:
        self.swap(i, smallest)
        self.heapify_down(smallest)
```

```
heap = MinHeap()
```

```
heap.insert(5)
```

```
heap.insert(3)
```

```
heap.insert(8)
```

```
heap.insert(1)
```

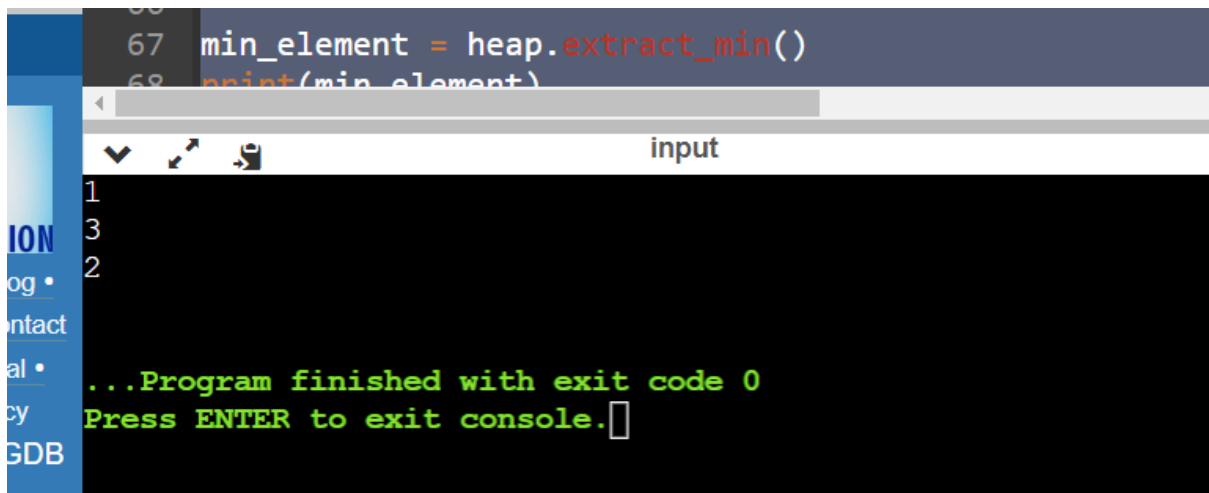
```
heap.insert(10)
```

```
min_element = heap.extract_min()
print(min_element)
```

```
min_element = heap.extract_min()
print(min_element)
```

```
heap.insert(2)
```

```
min_element = heap.extract_min()
print(min_element)
```



```
67 min_element = heap.extract_min()
68 print(min_element)
input
1
3
2
...Program finished with exit code 0
Press ENTER to exit console.
```

2. Max Heap

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

    def parent(self, i):
        return (i - 1) // 2

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

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

    def swap(self, i, j):
```

```

        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

def insert(self, value):
    self.heap.append(value)
    current = len(self.heap) - 1
    while (
        current > 0
        and self.heap[current] > self.heap[self.parent(current)]
    ):
        self.swap(current, self.parent(current))
        current = self.parent(current)

def heapify(self, n, i):
    largest = i
    left = self.left_child(i)
    right = self.right_child(i)

    if left < n and self.heap[left] > self.heap[largest]:
        largest = left

    if right < n and self.heap[right] > self.heap[largest]:
        largest = right

    if largest != i:
        self.swap(i, largest)
        self.heapify(n, largest)

def build_heap(self, arr):
    n = len(arr)
    self.heap = arr
    for i in range(n // 2 - 1, -1, -1):
        self.heapify(n, i)

def extract_max(self):
    if len(self.heap) == 0:
        return None

    max_value = self.heap[0]
    self.heap[0] = self.heap[-1]
    self.heap.pop()
    self.heapify(len(self.heap), 0)
    return max_value

# Create a new max heap
heap = MaxHeap()

heap.insert(5)
heap.insert(10)

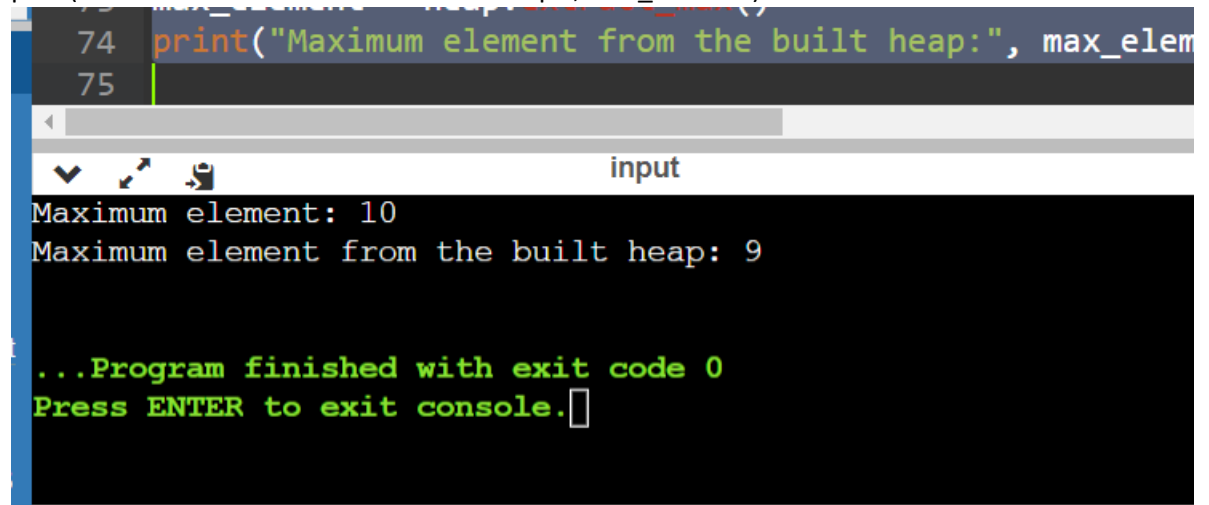
```

```
heap.insert(3)
heap.insert(8)
heap.insert(1)
```

```
max_element = heap.extract_max()
print("Maximum element:", max_element)
```

```
arr = [7, 2, 9, 4, 6]
heap.build_heap(arr)
```

```
max_element = heap.extract_max()
print("Maximum element from the built heap:", max_element)
```

The image shows a code editor window with a dark background. Line 74 contains the code `print("Maximum element from the built heap:", max_element)`. Below the editor is a terminal window titled "input". The terminal shows the output of the program: "Maximum element: 10" and "Maximum element from the built heap: 9". At the bottom of the terminal, it says "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor.

```
74 print("Maximum element from the built heap:", max_element)
75
```

input

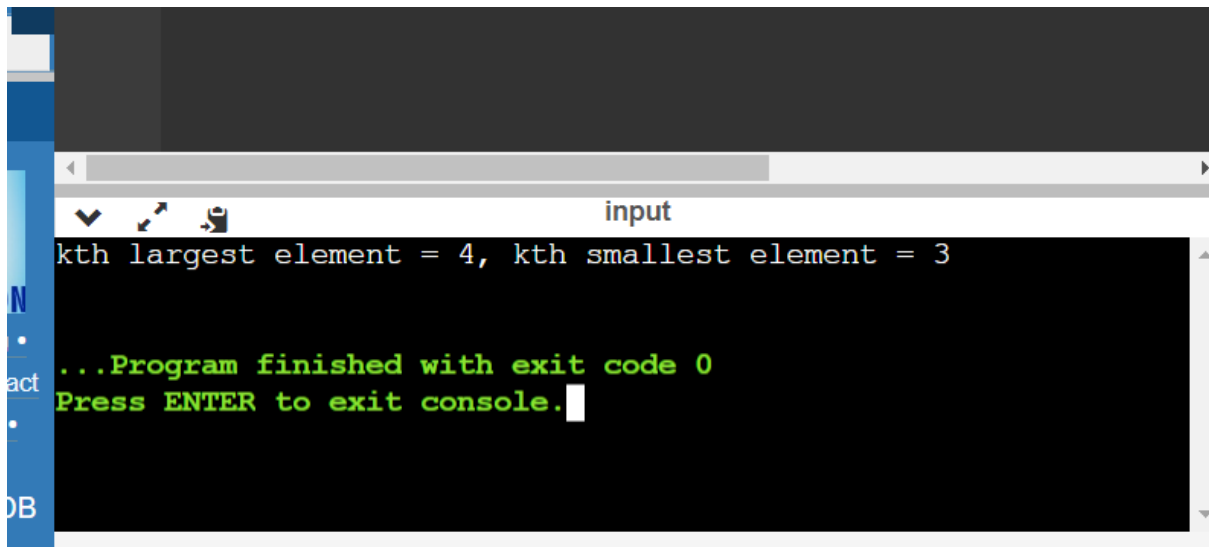
Maximum element: 10
Maximum element from the built heap: 9

...Program finished with exit code 0
Press ENTER to exit console.

Problem Statement: Given an unsorted array, print Kth Largest and Smallest Element from an unsorted array.

```
def find_kth_largest_smallest(array, k):
    array.sort()
    kth_smallest = array[k - 1]
    kth_largest = array[len(array) - k]
    print(f"kth largest element = {kth_largest}, kth smallest element = {kth_smallest}")

array = [1, 2, 6, 4, 5, 3]
k = 3
find_kth_largest_smallest(array, k)
```

A screenshot of a terminal window. The title bar at the top says 'input'. The terminal content shows 'kth largest element = 4, kth smallest element = 3' in white text. Below that, in green text, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.' with a white cursor. The terminal has a dark background and standard window controls on the left.

```
input
kth largest element = 4, kth smallest element = 3
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given two equally sized 1-D arrays **A**, **B** containing **N** integers each.

A **sum combination** is made by adding one element from array **A** and another element of array **B**.

Return the **maximum C valid sum combinations** from all the possible sum combinations.

```
def find_max_sum_combinations(A, B, C):
    combinations = []
    for num_a in A:
        for num_b in B:
            combinations.append(num_a + num_b)
    combinations.sort(reverse=True)
    return combinations[:C]
```

```
A1 = [3, 2]
```

```
B1 = [1, 4]
```

```
C1 = 2
```

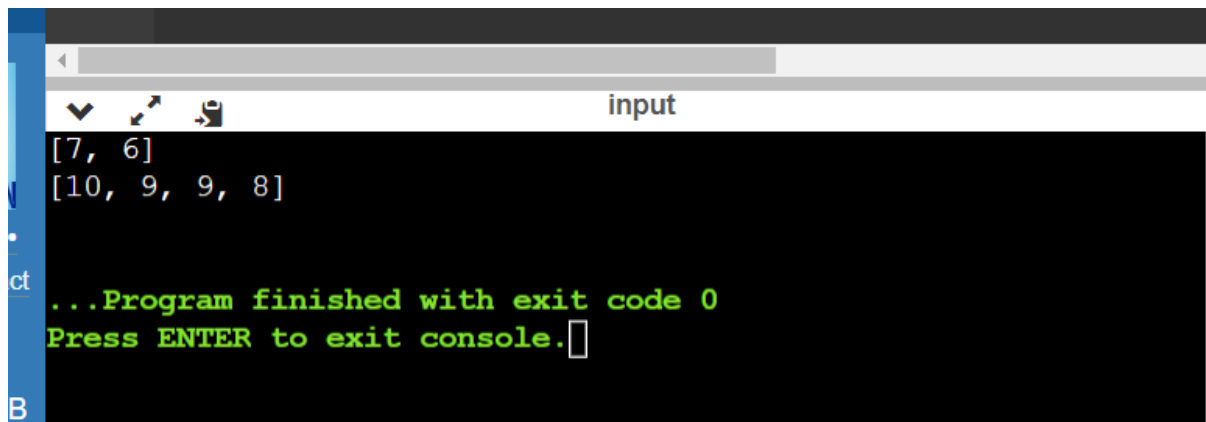
```
print(find_max_sum_combinations(A1, B1, C1))
```

```
A2 = [1, 4, 2, 3]
```

```
B2 = [2, 5, 1, 6]
```

C2 = 4

```
print(find_max_sum_combinations(A2, B2, C2))
```

A screenshot of a terminal window with a dark background. The title bar at the top says 'input'. The terminal shows two lines of input: '[7, 6]' and '[10, 9, 9, 8]'. Below the input, green text indicates the program has finished with exit code 0 and prompts the user to press ENTER to exit the console. On the left side of the terminal, there is a vertical blue bar with some text partially visible: 'ct' and 'B'.

Problem Statement: The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values.

```
import heapq
```

```
class MedianFinder:
```

```
    def __init__(self):
```

```
        self.max_heap = []
```

```
        self.min_heap = []
```

```
    def addNum(self, num: int) -> None:
```

```
        heapq.heappush(self.max_heap, -num)
```

```
        heapq.heappush(self.min_heap, -heapq.heappop(self.max_heap))
```

```
        if len(self.min_heap) > len(self.max_heap):
```

```
            heapq.heappush(self.max_heap, -heapq.heappop(self.min_heap))
```

```
    def findMedian(self) -> float:
```

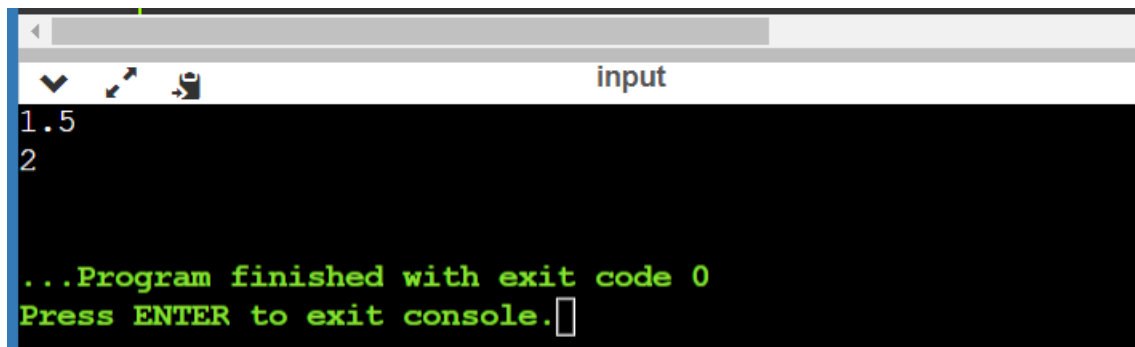
```
        if len(self.max_heap) == len(self.min_heap):
```

```
            return (-self.max_heap[0] + self.min_heap[0]) / 2
```

```
        else:
```

```
            return -self.max_heap[0]
```

```
medianFinder = MedianFinder()
medianFinder.addNum(1)
medianFinder.addNum(2)
print(medianFinder.findMedian())
medianFinder.addNum(3)
print(medianFinder.findMedian())
```

A screenshot of a terminal window with a dark background. The title bar at the top says "input". The terminal shows the numbers "1.5" and "2" on separate lines. Below these, a green message reads "...Program finished with exit code 0" followed by "Press ENTER to exit console." and a cursor icon.

```
1.5
2

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Merge k sort arrays.

```
import heapq
```

```
def merge_k_sorted_arrays(arrays):
    result = []
    heap = []

    for i, arr in enumerate(arrays):
        if len(arr) > 0:
            heapq.heappush(heap, (arr[0], i, 0))

    while heap:
        val, arr_idx, idx = heapq.heappop(heap)
```



```

result.append(val)

if idx + 1 < len(arrays[arr_idx]):

    heapq.heappush(heap, (arrays[arr_idx][idx + 1], arr_idx, idx + 1))

return result

```

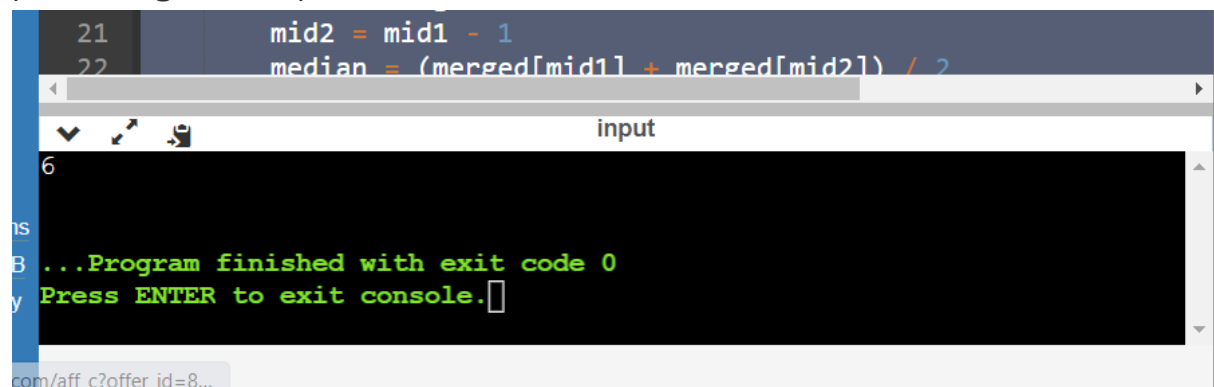
```

arrays = [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

merged_array = merge_k_sorted_arrays(arrays)

print(merged_array)

```



The screenshot shows a code editor with a dark theme. The code being edited is a Python function for finding the median of two sorted arrays. The code is as follows:

```

21     mid2 = mid1 - 1
22     median = (merged[mid1] + merged[mid2]) / 2

```

Below the code editor, there is a terminal window titled "input". The terminal shows the output of the program:

```

6
...Program finished with exit code 0
Press ENTER to exit console.

```

The terminal window also shows a URL at the bottom: com/aff_c?offer_id=8...

Problem Statement: Merge k sorted array

```

def find_kth_element(array1, array2, k):

    m, n = len(array1), len(array2)

    i, j = 0, 0

    count = 0

    while i < m and j < n:

        if array1[i] <= array2[j]:

            current_element = array1[i]

            i += 1

        else:

```

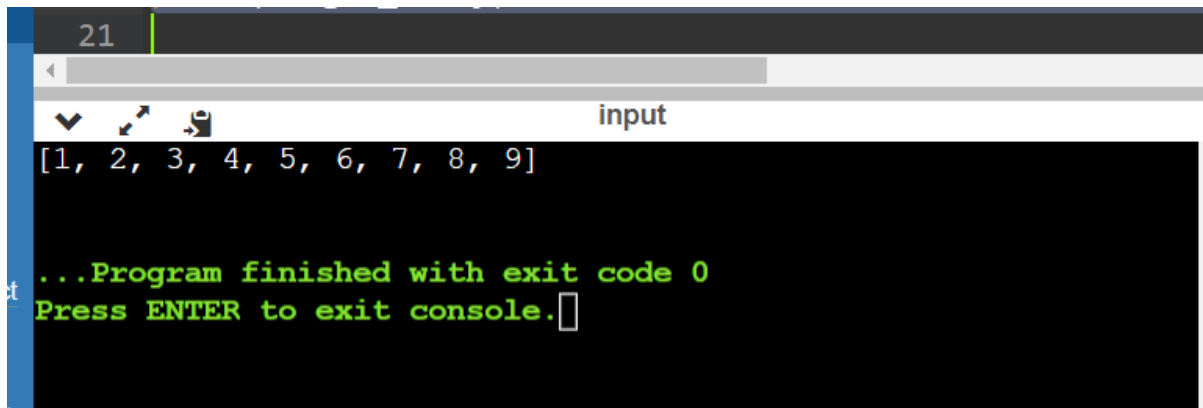
```
        current_element = array2[j]
        j += 1

    count += 1
    if count == k:
        return current_element

while i < m:
    count += 1
    if count == k:
        return array1[i]
    i += 1
while j < n:
    count += 1
    if count == k:
        return array2[j]
    j += 1

return "Error: k exceeds the total number of elements."
```

```
array1 = [2, 3, 6, 7, 9]
array2 = [1, 4, 8, 10]
k = 5
result = find_kth_element(array1, array2, k)
print(result)
```



A screenshot of a terminal window. The title bar shows '21' and 'input'. The terminal content displays a list of numbers: `[1, 2, 3, 4, 5, 6, 7, 8, 9]`. Below this, a green message states: `...Program finished with exit code 0`. The final line is `Press ENTER to exit console.` followed by a cursor.

Problem Statement : k most frequent emement

```
def find_k_most_frequent_elements(nums, k):
```

```
    frequency_map = {}
```

```
    for num in nums:
```

```
        if num in frequency_map:
```

```
            frequency_map[num] += 1
```

```
        else:
```

```
            frequency_map[num] = 1
```

```
    sorted_elements = sorted(frequency_map.keys(), key=lambda x: frequency_map[x], reverse=True)
```

```
    return sorted_elements[:k]
```

```
numbers = [1, 2, 3, 2, 1, 3, 4, 5, 4, 2, 5, 5]
```

```
k = 3
```

```
k_most_frequent = find_k_most_frequent_elements(numbers, k)
```

```
print(f"The {k} most frequent elements are: {k_most_frequent}")
```

