Program:

import math

class MaxHeap:

def \_\_init\_\_(self):

self.heap = []

def insert(self, val):

self.heap.append(val)

self.\_bubble\_up(len(self.heap) - 1)

def delete\_max(self):

if not self.heap:

return None

if len(self.heap) == 1:

return self.heap.pop()

max\_val = self.heap[0]

# Move last element to root and heapify down

self.heap[0] = self.heap.pop()

self.\_bubble\_down(0)

return max\_val

def peek\_max(self):

return self.heap[0] if self.heap else None

def display(self):

if not self.heap:

print("Heap is empty")

return

n = len(self.heap)

levels = math.floor(math.log2(n)) + 1

max\_width = 2 \*\* levels

index = 0

for level in range(levels):

level\_nodes = 2 \*\* level

space\_between = max\_width // level\_nodes

# Print leading spaces

print(' ' \* (space\_between // 2), end='')

# Print nodes in this level

for i in range(level\_nodes):

if index >= n:

break

print(f"{self.heap[index]:2}", end=' ' \* space\_between)

index += 1

print('\n')

def \_bubble\_up(self, index):

parent = (index - 1) // 2

while index > 0 and self.heap[index] > self.heap[parent]:

self.heap[index], self.heap[parent] = self.heap[parent], self.heap[index]

index = parent

parent = (index - 1) // 2

def \_bubble\_down(self, index):

n = len(self.heap)

while True:

largest = index

left = 2 \* index + 1

right = 2 \* index + 2

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 != index:

self.heap[index], self.heap[largest] = self.heap[largest], self.heap[index]

index = largest

else:

break

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

h = MaxHeap()

h.insert(10)

h.insert(40)

h.insert(20)

print("Heap after insertions:")

h.display()

print("Peek max:", h.peek\_max())

print("Delete max:", h.delete\_max())

print("Heap after deleting max:")

h.display()

output:

Heap after insertions:

40

10 20

Peek max: 40

Delete max: 40

Heap after deleting max:

20

10