Max Heap in Python - GeeksforGeeks

Source: https://www.geeksforgeeks.org/max-heap-in-python/

Courses Tutorials Practice Jobs Python Tutorial Data Types Interview Questions Examples Quizzes DSA Python Data Science NumPy Pandas Practice Django Flask Technical Scripter 2026 Explore Python Fundamentals Python Introduction Input and Output in Python Python Variables Python Operators Python Keywords Python Data Types Conditional Statements in Python Loops in Python - For, While and Nested Loops Python Functions Recursion in Python Python Lambda Functions Python Data Structures Python String Python Lists Python Tuples Python Dictionary Python Sets Python Arrays List Comprehension in Python Advanced Python Python OOP Concepts Python Exception Handling File Handling in Python Python Database Tutorial Python MongoDB Tutorial Python MySQL Python Packages Python Modules Python DSA Libraries List of Python GUI Library and Packages Data Science with Python NumPy Tutorial - Python Library Pandas Tutorial Matplotlib Tutorial Python Seaborn Tutorial StatsModel Library - Tutorial Learning Model Building in Scikit-learn TensorFlow Tutorial PyTorch Tutorial Web Development with Python Flask Tutorial Django Tutorial | Learn Django Framework Django ORM - Inserting, Updating & Deleting Data Templating With Jinja2 in Flask Django Templates Build a REST API using Flask - Python Building a Simple API with Django REST Framework Python Practice Python Quiz Python Coding Practice Python Interview Questions and Answers Data Science Course 90% Refund Max Heap in Python Last Updated : 12 Jul, 2025 A Max-Heap is a Data Structure with the following properties: It is a Complete Binary Tree . The value of the root node must be the largest among all its descendant nodes, and the same thing must be done for its left and right sub-tree also. How is Max Heap represented? A max Heap is a Complete Binary Tree. A max heap is typically represented as an array. The root element will be at Arr[0]. Below table shows indexes of other nodes for the ith node, i.e., Arr[i]: Arr[(i-1)/2] Returns the parent node. Arr[(2*i)+1] Returns the left child node. Arr[(2*i)+2] Returns the right child node. Operations on Max Heap: getMax() : It returns the root element of Max Heap. Time Complexity of this operation is O(1) . extractMax() : Removes the maximum element from MaxHeap. Time Complexity of this Operation is O(log n) as this operation needs to maintain the heap property (by calling heapify()) after removing the root. insert() : Inserting a new key takes O(log n) time. We add a new key at the end of the tree. If the new key is smaller than its parent, then we don&#x2019t need to do anything. Otherwise, we need to traverse up to fix the violated heap property. Note: In the below implementation, we do indexing from index 1 to simplify the implementation. Python import sys class MaxHeap : def __init__ (self , cap): self . cap = cap self . n = 0 self . a = [0] * (cap + 1) self . a [0] = sys . maxsize self . root = 1 def parent (self , i): return i // 2 def left (self , i): return 2 * i def right (self , i): return 2 * i + 1 def isLeaf (self , i): return i > (self . n // 2) and i <= self . n def swap (self , i , j): self . a [i], self . a [j] = self . a [j], self . a [i] def maxHeapify (self , i): if not self . isLeaf (i): largest = i if self . left (i) <= self . n and self . a [i] < self . a [self . left (i)]: largest = self . left (i) if self . right (i) <= self . n and self . a [largest] < self . a [self . right (i)]: largest = self . right (i) if largest != i : self . swap (i , largest) self . maxHeapify (largest) def insert (self , val): if self . n >= self . cap : return self . n += 1 self . a [self . n] = val i = self . n while self . a [i] > self . a [self . parent (i)]: self . swap (i , self . parent (i)) i = self . parent (i) def extractMax (self): if self . n == 0 : return None max_val = self . a [self . root] self . a [self . root] = self . a [self . n] self . n -= 1 self . maxHeapify (self . root) return max_val def printHeap (self): for i in range (1 , (self . n // 2) + 1): print (f "PARENT: { self . a [i] } " , end = " ") if self . left (i) <= self . n : print (f "LEFT: { self . a [self . left (i)] } " , end = " ") if self . right (i) <= self . n : print (f "RIGHT: { self . a [self . right (i)] } " , end = " ") print () # Example if __name__ == "__main__" : print ("The maxHeap is:") h = MaxHeap (15) vals = [5 , 3 , 17 , 10 , 84 , 19 , 6 , 22 , 9] for val in vals : h . insert (val) h . printHeap () print ("The Max val is" , h . extractMax ()) Output: The maxHeap is PARENT : 84 LEFT CHILD : 22 RIGHT CHILD : 19 PARENT : 22 LEFT CHILD : 17 RIGHT CHILD : 10 PARENT : 19 LEFT CHILD : 5 RIGHT CHILD : 6 PARENT : 17 LEFT CHILD : 3 RIGHT CHILD : 9 The Max val is 84 Using Library functions: We use heapq class to implement Heap in Python. By default Min Heap is implemented by this class. But we multiply each value by -1 so that we can use it as MaxHeap. Python # Python3 program to demonstrate heapq (Max Heap) from heapq import heappop , heappush , heapify # Create an empty heap h = [] heapify (h) # Add elements (multiplying by -1 to simulate Max Heap) heappush (h , - 10) heappush (h , - 30) heappush (h , - 20) heappush (h , - 400) # Print max element print ("Max:" , - h [

0]) # Print heap elements print ("Heap:" , [- i for i in h]) # Pop max element heappop (h) # Print heap after removal print ("Heap after pop:" , [- i for i in h]) Output Max: 400 Heap: [400, 30, 20, 10] Heap after pop: [30, 10, 20] Using Library functions with dunder method for Numbers, Strings, Tuples, Objects etc We use heapq class to implement Heaps in Python. By default Min Heap is implemented by this class. To implement MaxHeap not limiting to only numbers but any type of object(String, Tuple, Object etc) we should Create a Wrapper class for the item in the list. Override the __lt__ dunder method to give inverse result. Following is the implementation of the method mentioned here. Python """ Python3 program to implement MaxHeap using heapq for Strings, Numbers, and Objects """ from functools import total_ordering import heapq @total_ordering class Wrap : def __init__ (self , v): self . v = v def __lt__ (self , o): return self . v > o . v # Reverse for Max Heap def __eq__ (self , o): return self . v == o . v # Max Heap for numbers h = [10 , 20 , 400 , 30] wh = list (map (Wrap , h)) heapq . heapify (wh) print ("Max:" , heapq . heappop (wh) . v) # Max Heap for strings h = ["this" , "code" , "is" , "wonderful"] wh = list (map (Wrap , h)) heapq . heapify (wh) print ("Heap:" , end = " ") while wh : print (heapq . heappop (wh) . v , end = " ") Output Max: 400 Heap: wonderful this is code Using internal functions used in the heapq library This is by far the most simple and convenient way to apply max heap in python. DISCLAIMER - In Python, there's no strict concept of private identifiers like in C++ or Java. Python trusts developers and allows access to so-called "private" identifiers. However, since these identifiers are intended for internal use within the module, they are not officially part of the public API and may change or be removed in the future. Following is the implementation. Python from heapq import _heapify_max , _heappop_max , _siftdown_max # Implementing heappush for max heap def hpush (h , v): h . append (v) _siftdown_max (h , 0 , len (h) - 1) def maxh (a): c = a . copy () # Copy for later use _heapify_max (a) # Convert to max heap while a : print (_heappop_max (a)) # Pop elements a = c # Restore array h = [] for v in a : hpush (h , v) # Insert elements back into heap print ("Max Heap Ready!") while h : print (_heappop_max (h)) # Pop elements # Example a = [6 , 8 , 9 , 2 , 1 , 5] maxh (a) Output 9 8 6 5 2 1 Max Heap Ready! 9 8 6 5 2 1 Using Priority Queue Python from queue import PriorityQueue q = PriorityQueue () # Insert elements into the queue (negate values to simulate Max Heap) q . put (- 10) q . put (- 20) q . put (- 5) # Remove and return the highest priority item (convert back to positive) print (- q . get ()) # 20 (highest value) print (- q . get ()) # 10 # Check queue size print ('Items in queue:' , q . qsize ()) # Check if queue is empty print ('Is queue empty:' , q . empty ()) # Check if queue is full print ('Is queue full:' , q . full ()) Output 20 10 Items in queue: 1 Is queue empty: False Is queue full: False Comment Article Tags: Article Tags: Python Technical Scripter 2019 Python-DSA