Python Exercises- Heap queue

<https://www.w3resource.com/python-exercises/heap-queue-algorithm/index.php>

'''

1. Write a Python program to find the three largest integers from

a given list of numbers using the heap queue algorithm

'''

from heapq import nlargest

lst1=[1,2,3,4,5,8,3,13,11]

print(nlargest(3,lst1))

'''

2. Write a Python program to find the three smallest integers from a

given list of numbers using the heap queue algorithm.

'''

from heapq import nsmallest

lst1=[4,6,2,7,9,10,2,8]

print(nsmallest(3,lst1)) #[2,2,4]

'''

3. Write a Python program to implement heapsort by pushing all values

onto a heap and then popping off the smallest values one at a time

'''

import heapq as hq

lst1=[1,3,4,2,7,8,9,10,16,14]

hq.heapify(lst1)

print(lst1) # [1, 2, 4, 3, 7, 8, 9, 10, 16, 14]

for i in range(10):

hq.heappop(lst1)

print(lst1)

'''

[2, 3, 4, 10, 7, 8, 9, 14, 16]

[3, 7, 4, 10, 16, 8, 9, 14]

[4, 7, 8, 10, 16, 14, 9]

[7, 9, 8, 10, 16, 14]

[8, 9, 14, 10, 16]

[9, 10, 14, 16]

[10, 16, 14]

[14, 16]

[16]

[]

'''

#Textbook

def heapsort(iterable):

h = []

for value in iterable:

hq.heappush(h, value)

return [hq.heappop(h) for i in range(len(h))]

print(heapsort([1, 3, 5, 7, 9, 2, 4, 6, 8, 0]))

#heappush(h,value): add 'value' into 'h'

#heappop: pull out smallest by sequence

#warmup

lst2=[6,8,12,37,4,1,9,22,53,11,40]

h=[]

for val in lst2:

hq.heappush(h,val)

print(h)

print([hq.heappop(h) for i in range(len(h))])

#自動排序 耗用最少資源 但要先用heappush準備好heap

#heapify (or heappush into a new heap) can both do well ,anyway we need to prepare an iterable with ‘heap property’!

lst3=[6,8,12,37,4,1,9,22,53,11,40]

hq.heapify(lst3)

print([hq.heappop(lst3) for i in range(11)])

'''from chatGPT:

which one is faster? sorted() or heapsort?

For small lists, sorted() is likely to be faster since it uses an adaptive algorithm that takes advantage of pre-existing order in the data.(data可能本來就partially sorted了) Additionally, sorted() is implemented in C and highly optimized, making it very efficient for small to medium-sized lists.

On the other hand, for large lists, heapsort() may be faster because it has a lower memory overhead than sorted(). heapsort() only requires O(1) additional space, while sorted() requires O(n) additional space to store the sorted list.

各種sorting algorithms:

https://www.geeksforgeeks.org/comparison-among-bubble-sort-selection-sort-and-insertion-sort/

'''

'''

4. Write a Python function that accepts an arbitrary list and converts

it to a heap using the heap queue algorithm

'''

import heapq as hq

def cth(lsts):

hq.heapify(lsts)

return lsts

print(cth([4,7,2,3,8,16,9]))

'''

5. Write a Python program that deletes the smallest element from a heap

and then inserts a new item

'''

import heapq as hq

h=[2,8,5,6,7,9,12,15]

hq.heapify(h)

print(h)

hq.heappop(h)

hq.heappush(h,13)

print(h)

#Textbook

heap = [25, 44, 68, 21, 39, 23, 89]

hq.heapify(heap)

print("heap: ", heap)

hq.heapreplace(heap, 21)

print("heapreplace(heap, 21): ", heap)

hq.heapreplace(heap, 110)

print("heapreplace(heap, 110): ", heap)

#heapreplace(heap, item): replace the smallest value and replace it with 'item'

# more efficient than heappop + heappush

'''

6. Write a Python program to sort a given list of elements in ascending

order using the heap queue algorithm

'''

import heapq as hq

lst1=[5,7,2,8,1,9,4]

print(hq.nsmallest(7,lst1))

#Textbook

nums\_list = [18, 14, 10, 9, 8, 7, 9, 3, 2, 4, 1]

print("Original list:")

print(nums\_list)

hq.heapify(nums\_list)

s\_result = [hq.heappop(nums\_list) for i in range(len(nums\_list))]

print("\nSorted list:")

print(s\_result)

'''

7. Write a Python program to find the kth (1 <= k <= array's length)

largest element in an unsorted array using the heap queue algorithm

'''

import heapq as hq

def kl(lsts,k):

hq.heapify(lsts)

print(hq.nlargest(k,lsts)[-1])

kl([3,6,1,5,9,7,8],3) #7

#Textbook

import heapq

class Solution(object):

def find\_Kth\_Largest(self, nums, k):

"""

:type nums: List[int]

:type of k: int

:return value type: int

"""

h = []

for e in nums:

heapq.heappush(h, (-e, e))

for i in range(k):

w, e = heapq.heappop(h)

if i == k - 1:

return e

arr\_nums = [12, 14, 9, 50, 61, 41]

s = Solution()

result = s.find\_Kth\_Largest(arr\_nums, 3)

print("Third largest element:",result)

result = s.find\_Kth\_Largest(arr\_nums, 2)

print("\nSecond largest element:",result)

result = s.find\_Kth\_Largest(arr\_nums, 5)

print("\nFifth largest element:",result)

'''

8. Write a Python program to compute the maximum product of three numbers

in a given array of integers using the heap queue algorithm.

'''

import heapq as hq

def mpo3(lsts):

hq.heapify(lsts)

res=hq.nlargest(3,lsts)

return res[0]\*res[1]\*res[2]

print(mpo3([3,6,5,1,9,7,2]))

#Textbook

def maximumProduct(nums):

import heapq

a, b = heapq.nlargest(3, nums), heapq.nsmallest(2, nums)

return max(a[0] \* a[1] \* a[2], a[0] \* b[0] \* b[1])

arr\_nums = [12, 74, 9, 50, 61, 41]

print("Original array elements:")

print(arr\_nums)

print("Maximum product of three numbers of the said array:")

print(maximumProduct(arr\_nums))

'''

9. Write a Python program to find the top k integers that occur

the most frequently from a given list of sorted and distinct integers

using the heap queue algorithm.

'''

import heapq as hq

from collections import Counter

def tkoc(lsts,k):

h=[]

for i in lsts:

hq.heappush(h,i)

# print(h)

lsts1=[hq.heappop(h) for j in range(len(h))]

# print(lst2)

c=Counter(lsts1)

return [n for n,f in c.most\_common(k)]

lst1=[5,2,8,1,4,5,5,6,9,4,4,5,6]

print(tkoc(lst1,3))

#Textbook

def func(nums, k):

import collections

d = collections.defaultdict(int)

for row in nums:

for i in row:

d[i] += 1

temp = []

import heapq

for key, v in d.items():

if len(temp) < k:

temp.append((v, key))

if len(temp) == k:

heapq.heapify(temp)

else:

if v > temp[0][0]:

heapq.heappop(temp)

heapq.heappush(temp, (v, key))

result = []

while temp:

v, key = heapq.heappop(temp)

result.append(key)

return result

nums = [

[1, 2, 6],

[1, 3, 4, 5, 7, 8],

[1, 3, 5, 6, 8, 9],

[2, 5, 7, 11],

[1, 4, 7, 8, 12]

]

k = 3

print("Original lists:")

print(nums)

print("\nTop 3 integers that occur the most frequently in the said lists:")

print(func(nums, k))

'''

10. Write a Python program to get the most expensive and least expensive

items from a given dataset using the heap queue algorithm

'''

import heapq as hq

price=[245,756,112,34,899,47]

hq.heapify(price)

print(hq.nlargest(1,price))

print(hq.nsmallest(1,price))

#Textbook

import heapq

from pprint import pprint

items = [

{'name': 'Item-1', 'price': 101.1},

{'name': 'Item-2', 'price': 555.22},

{'name': 'Item-3', 'price': 45.09},

{'name': 'Item-4', 'price': 22.75},

{'name': 'Item-5', 'price': 16.30},

{'name': 'Item-6', 'price': 110.65}

]

cheap = heapq.nsmallest(3, items, key=lambda s: s['price'])

expensive = heapq.nlargest(3, items, key=lambda s: s['price'])

print("Original datasets:")

pprint(items)

print("\nFirst 3 expensive items:")

pprint(expensive)

print("\nFirst 3 cheap items:")

pprint(cheap)

'''

11. Write a Python program that merges multiple sorted inputs

into a single sorted iterator (over the sorted values)

using the heap queue algorithm

'''

#Textbook

import heapq

num1 = [25, 24, 15, 4, 5, 29, 110]

num2 = [19, 20, 11, 56, 25, 233, 154]

num3 = [24, 26, 54, 48]

num1 = sorted(num1)

num2 = sorted(num2)

num3 = sorted(num3)

result = heapq.merge(num1, num2, num3)

print(list(result))

num4 = [25, 24, 15, 4, 5, 29, 110]

num5 = [19, 20, 11, 56, 25, 233, 154]

num6 = [24, 26, 54, 48]

result1=heapq.merge(num4,num5,num6)

print(list(result1))

# [19, 20, 11, 24, 25, 24, 15, 4, 5, 26, 29, 54, 48, 56, 25, 110, 233, 154]

# unwanted result since num4,num5,num6 not sorted priorly!

# if sorted in one go:

num7=num4+num5+num6

print(sorted(num7))

'''

12. Given a n x n matrix where each of the rows and columns is sorted

in ascending order, write a Python program to find the

kth smallest element in the matrix using the heap queue algorithm.

'''

import numpy as np

import heapq as hq

arr33=np.array([[1,2,3],[4,5,6],[7,8,9]])

arr33f=arr33.flatten()

l33=list(arr33f)

hq.heapify(l33)

k=int(input('Input k:\n'))

print(hq.nsmallest(k,l33))

#Textbook

import heapq

class Solution(object):

def find\_Kth\_Smallest(self, matrix, k):

"""

:type matrix: List[List[int]]

:type k: int

:rtype: int

"""

m, n = len(matrix), len(matrix[0])

temp = [[False] \* n for \_ in range(m)]

q = [(matrix[0][0], 0, 0)]

ans = None

temp[0][0] = True

for \_ in range(k):

ans, i, j = heapq.heappop(q)

if i + 1 < m and not temp[i + 1][j]:

temp[i + 1][j] = True

heapq.heappush(q, (matrix[i + 1][j], i + 1, j))

if j + 1 < n and not temp[i][j + 1]:

temp[i][j + 1] = True

heapq.heappush(q, (matrix[i][j + 1], i, j + 1))

return ans

matrix = [

[0, 5, 9],

[11, 12, 13],

[15, 17, 19]

]

k = 1

s = Solution()

result = s.find\_Kth\_Smallest(matrix, k)

print("First smallest element:",result)

k = 2

s = Solution()

result = s.find\_Kth\_Smallest(matrix, k)

print("\nSecond smallest element:",result)

k = 3

s = Solution()

result = s.find\_Kth\_Smallest(matrix, k)

print("\nThird smallest element:",result)

print()

#warmup- several functions in one class (if 同質性高)

class MathUtils:

def add(self, x, y):

return x + y

def subtract(self, x, y):

return x - y

def multiply(self, x, y):

return x \* y

def divide(self, x, y):

if y == 0:

raise ValueError("Cannot divide by zero")

return x / y

calculator=MathUtils()

print(calculator.add(4,8))

print(calculator.divide(7.41978,4.5567))

print(calculator.divide(4,0))

'''

13. Write a Python program to find the nth super ugly number

from a given prime list of size k using the heap queue algorithm.

\*Ugly number: prime number僅含 2,3,5 ;

ex: 6 =2\*3; 8 =2\*2\*2 are ugly numbers

ex: 1 本身即被視為 ugly number

#Super ugly number: A super ugly number is a positive integer that has only prime factors that are in a given list of prime numbers

For example, if the given list of prime numbers is {2, 7, 13, 19}, the first few super ugly numbers are:

1 (by convention)

2 (only prime factor is 2)

7 (only prime factor is 7)

13 (only prime factor is 13)

14 (prime factors are 2 and 7)

19 (only prime factor is 19)

26 (prime factors are 2 and 13)

28 (prime factors are 2, 2, and 7)

38 (prime factors are 2, 19)

49 (prime factors are 7, 7)

Another example, [1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32] is the sequence of the first 12 super ugly numbers given primes = [2, 7, 13, 19] of size 4.

'''

def is\_super\_ugly(pnl,n):

for p in pnl:

while n%p==0:

n//=p

return n==1

print(is\_super\_ugly([2,7,13,19],14)) #True

print(is\_super\_ugly([2,7,13,19],76)) #True

print(is\_super\_ugly([2,7,13,19],37)) #False

def super\_ugly\_list(pnl1, first\_n):

i=1

res=[1]

while len(res)<first\_n:

i+=1

if is\_super\_ugly(pnl1,i):

res.append(i)

print('the first %d numbers from the super\_ugly\_list is:' % first\_n)

return res

print(super\_ugly\_list([2,7,13,19],12))

print(super\_ugly\_list([2,3,5],20))

#Textbook

import heapq

#Ref.: https://bit.ly/32c9P3A

def nth\_Super\_Ugly\_Number(n, primes):

uglies = [1]

def gen(prime):

for ugly in uglies:

yield ugly \* prime

merged = heapq.merge(\*map(gen, primes))

#initially, uglies:[1,2] because gen(1)=2; then [1,2,4]...

while len(uglies) < n:

ugly = next(merged)

if ugly != uglies[-1]:

uglies.append(ugly)

return uglies[-1]

#---end of def---

n = 12

primes = [2,7,13,19]

print(nth\_Super\_Ugly\_Number(n, primes)) #32

### Ugly numbers

An ugly number is a number that can be represented exactly in binary (i.e., with no rounding errors) but has a non-terminating decimal expansion. One example of an ugly number is 0.1, which in binary is represented as:

0.0001100110011001100110011001100110011001100110011...

In the case of 0.1, the binary fraction representation is a repeating sequence because the decimal fraction 0.1 cannot be represented exactly in binary. This is because 0.1 is not a multiple of any power of 2, and so its binary representation involves a repeating pattern of digits after the binary point.

To convert the decimal number 0.1 to binary, we can use the following method:

Multiply the decimal number by 2. 0.1 \* 2 = 0.2

Write down the integer part of the result. 0.2 has an integer part of 0, so we write down 0 as the first binary digit.

Take the fractional part of the result and repeat the process. The fractional part of 0.2 is 0.2, so we multiply it by 2. 0.2 \* 2 = 0.4

Again, write down the integer part of the result. 0.4 has an integer part of 0, so we write down 0 as the second binary digit.

Repeat the process, taking the fractional part and multiplying by 2. The fractional part of 0.4 is 0.4, so we multiply it by 2. 0.4 \* 2 = 0.8

Write down the integer part of the result. 0.8 has an integer part of 0, so we write down 0 as the third binary digit.

Repeat the process again. The fractional part of 0.8 is 0.8, so we multiply it by 2. 0.8 \* 2 = 1.6

Write down the integer part of the result. 1.6 has an integer part of 1, so we write down 1 as the fourth binary digit.

Continue the process until the desired level of accuracy is reached. If we continue this process, we see that the decimal representation of the binary number converges to 0.0001100110011..., with the 0's and 1's repeating in a pattern.

Therefore, the binary representation of the decimal number 0.1 is 0.0001100110011..., with the 0's and 1's repeating in a pattern.

窗体顶端

窗体底端

可解釋 為何 0.5 的 binary是 0.1 : 0.5+0.5=1 ==> (binary) 0.1 +0.1 = 1

所有數字 的 decimal face.value = binary face.value 只有 1

'''

14. Write a Python program to get the k most frequent elements

from a given non-empty list of words using the heap queue algorithm

'''

#Textbook

import heapq

from collections import Counter

class Solution:

def top\_K\_Frequent(self, words, k):

"""

:type words: List[str]

:type k: int

:return type: List[str]

"""

ctr = Counter(words)

heap = [(-ctr[word], word) for word in ctr]

heapq.heapify(heap)

return [heapq.heappop(heap)[1] for \_ in range(k)]

if \_\_name\_\_ == '\_\_main\_\_':

words = ["a", "abc", "abcdef", "a", "abcd", "abcd", "abc", "abcdefg"]

k = 3

s = Solution()

print("3 most frequent elements:")

print(s.top\_K\_Frequent(words, k))

'''

15. Write a Python program to check if the letters in a given string

can be rearranged. This is to make sure that two characters that are

adjacent to each other are different using the heap queue algorithm

(explain more in detail) reorganizes the characters in it such that no two adjacent characters are the same, and returns the resulting string

'''

import heapq

from collections import Counter

def reorganizeString(S):

ctr = Counter(S)

heap = [(-value, key) for key, value in ctr.items()]

heapq.heapify(heap)

if (-heap[0][0]) \* 2 > len(S) + 1:

return ""

#by checking if the negative count of the most common character times 2 is greater than the length of the input string plus one. If it is, there are not enough characters to form a string without adjacent characters being the same

ans = []

while len(heap) >= 2:

nct1, char1 = heapq.heappop(heap)

nct2, char2 = heapq.heappop(heap)

ans.extend([char1, char2])

if nct1 + 1: heapq.heappush(heap, (nct1 + 1, char1))

if nct2 + 1: heapq.heappush(heap, (nct2 + 1, char2))

return "".join(ans) + (heap[0][1] if heap else "")

print(reorganizeString("aab")) #aba

print(reorganizeString("abc")) #abc

print(reorganizeString("aabb")) #abab

print(reorganizeString("abccdd")) #cdabcd

'''

16. Write a Python program that adds integer numbers from

the data stream to a heapq and computes the median of all elements.

Use the heap queue algorithm

'''

import heapq as hq

from statistics import median

#warmup

h=[]

hq.heappush(h,2)

print(h) #1

hq.heappush(h,1)

print(h) #[1,2]

hq.heappush(h,4)

print(h) #[1,2,4] 自動堆好了!

print(median(h)) #2

print(sum(h)) #7

def anm(ds):

h=[]

for i in ds:

hq.heappush(h,i)

return 'sum:',sum(h), 'median',median(h)

print(anm([6,9,12,1,4,86,48]))

print(anm([i for i in range(1,101)])) #median is 50.5 since totally even numbers (100)

#Textbook

import heapq

class Median\_Finder:

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

#initialize data structure

self.max\_heap = []

self.min\_heap = []

def add\_Number(self, num):

#type num: int, rtype: void

if not self.max\_heap and not self.min\_heap:

heapq.heappush(self.min\_heap, num)

return

if not self.max\_heap:

if num > self.min\_heap[0]:

heapq.heappush(self.max\_heap, -heapq.heappop(self.min\_heap))

heapq.heappush(self.min\_heap, num)

else:

heapq.heappush(self.max\_heap, -num)

return

if len(self.max\_heap) == len(self.min\_heap):

if num < -self.max\_heap[0]:

heapq.heappush(self.max\_heap, -num)

else:

heapq.heappush(self.min\_heap, num)

elif len(self.max\_heap) > len(self.min\_heap):

if num < -self.max\_heap[0]:

heapq.heappush(self.min\_heap, -heapq.heappop(self.max\_heap))

heapq.heappush(self.max\_heap, -num)

else:

heapq.heappush(self.min\_heap, num)

else:

if num > self.min\_heap[0]:

heapq.heappush(self.max\_heap, -heapq.heappop(self.min\_heap))

heapq.heappush(self.min\_heap, num)

else:

heapq.heappush(self.max\_heap, -num)

def find\_Median(self):

#rtype: float

if len(self.max\_heap) == len(self.min\_heap):

return (-self.max\_heap[0] + self.min\_heap[0]) / 2

elif len(self.max\_heap) > len(self.min\_heap):

return -self.max\_heap[0]

else:

return self.min\_heap[0]

S = Median\_Finder()

S.add\_Number(1)

S.add\_Number(2)

result = S.find\_Median()

print(result)

S.add\_Number(3)

result = S.find\_Median()

print(result)

S.add\_Number(4)

S.add\_Number(5)

result = S.find\_Median()

print(result)

'''

17. You have two integer arrays sorted in ascending order and

an integer k. Write a Python program to find k number of pairs (u, v)

which consist of one element from the first array and one element

from the second array using the heap queue algorithm

'''

#Textbook

import heapq

def k\_Smallest\_Pairs(nums1, nums2, k):

queue = []

def push(i, j):

if i < len(nums1) and j < len(nums2):

heapq.heappush(queue, [nums1[i] + nums2[j], i, j])

push(0, 0)

pairs = []

while queue and len(pairs) < k:

\_, i, j = heapq.heappop(queue)

pairs.append([nums1[i], nums2[j]])

push(i, j + 1)

if j == 0:

push(i + 1, 0)

return pairs

nums1 = [1,3,7]

nums2 = [2,4,6]

k = 2

result = k\_Smallest\_Pairs(nums1, nums2, k)

print(result)

'''

18. Write a Python program to find the nth ugly number using the

heap queue algorithm

'''

from heapq import merge

#my code does not use it

uglies=[1]

def fnun(n):

i=1

while len(uglies)!=n:

i+=1

if i%2==0 or i%3==0 or i%5==0:

uglies.append(i)

return uglies[-1]

print(fnun(10)) #12 (1,2,3,4,5,6,8,9,10,12)

#Textbook

import heapq

# class Solution(object):

# #:type n: integer

# #:return type: integer

# def nth\_Ugly\_Number(self, n):

# ugly\_num = 0

# heap = []

# heapq.heappush(heap, 1)

# for \_ in range(n):

# ugly\_num = heapq.heappop(heap)

# if ugly\_num % 2 == 0:

# heapq.heappush(heap, ugly\_num \* 2)

# elif ugly\_num % 3 == 0:

# heapq.heappush(heap, ugly\_num \* 2)

# heapq.heappush(heap, ugly\_num \* 3)

# else:

# heapq.heappush(heap, ugly\_num \* 2)

# heapq.heappush(heap, ugly\_num \* 3)

# heapq.heappush(heap, ugly\_num \* 5)

# return ugly\_num

# n = 7

# S = Solution()

# result = S.nth\_Ugly\_Number(n)

# print("7th Ugly number:")

# print(result)

# n = 10

# result = S.nth\_Ugly\_Number(n)

# print("\n10th Ugly number:")

# print(result)

# wrong result 1 !

#correct provided by chatGPT

import heapq

class Solution(object):

def nth\_Ugly\_Number(self, n):

if n <= 0:

return 0

ugly\_num = 0

heap = []

heapq.heappush(heap, 1)

for \_ in range(n):

ugly\_num = heapq.heappop(heap)

if ugly\_num % 2 == 0:

heapq.heappush(heap, ugly\_num \* 2)

if ugly\_num % 3 == 0:

heapq.heappush(heap, ugly\_num \* 3)

if ugly\_num % 5 == 0:

heapq.heappush(heap, ugly\_num \* 5)

return ugly\_num

n = 7

S = Solution()

result = S.nth\_Ugly\_Number(n)

print("7th Ugly number:")

print(result)

n = 10

result = S.nth\_Ugly\_Number(n)

print("\n10th Ugly number:")

print(result)

#still error (with indes out of range) TBD

'''

19. Write a Python program to print a heap as a tree-like data structure

'''

import heapq as hq

from math import floor,log2

lst1=[5,7,2,4,9,1,8]

hq.heapify(lst1)

print(lst1)

#tree should be: L1-1,L2=2,L3-4,L4-8.... 2^n

print(floor(log2(7)))

#Textbook

import math

from io import StringIO

#source https://bit.ly/38HXSoU

def show\_tree(tree, total\_width=60, fill=' '):

"""Pretty-print a tree.

total\_width depends on your input size"""

output = StringIO()

last\_row = -1

for i, n in enumerate(tree):

if i:

row = int(math.floor(math.log(i+1, 2)))

else:

row = 0

if row != last\_row:

output.write('\n')

columns = 2\*\*row

col\_width = int(math.floor((total\_width \* 1.0) / columns))

output.write(str(n).center(col\_width, fill))

last\_row = row

print (output.getvalue())

print ('-' \* total\_width)

return

#test

import heapq

heap = []

heapq.heappush(heap, 1)

heapq.heappush(heap, 2)

heapq.heappush(heap, 3)

heapq.heappush(heap, 4)

heapq.heappush(heap, 7)

heapq.heappush(heap, 9)

heapq.heappush(heap, 10)

heapq.heappush(heap, 8)

heapq.heappush(heap, 16)

heapq.heappush(heap, 14)

show\_tree(heap)

'''

20. Write a Python program to combine two sorted lists

using the heapq module.

Sample Output:

Original sorted lists:

[1, 3, 5, 7, 9, 11]

[0, 2, 4, 6, 8, 10]

After merging the said two sorted lists:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

'''

import heapq as hq

def c2sl(lst1,lst2):

hq.heapify(lst1)

hq.heapify(lst2)

lst3=lst1+lst2

return hq.nsmallest(len(lst3),lst3)

print(c2sl([1,3,5,7,9,11],[0,2,4,6,8,10]))

#Textbook

from heapq import merge

nums1 = [1, 3, 5, 7, 9, 11]

nums2 = [0, 2, 4, 6, 8, 10]

print("Original sorted lists:")

print(nums1)

print(nums2)

print("\nAfter merging the said two sorted lists:")

print(list(merge(nums1, nums2)))

'''

21. Write a Python program to push three items into the heap and

print the items from the heap.

Sample Output:

('V', 1)

('V', 2)

('V', 3)

'''

from heapq import \*

h=[]

heappush(h,('V', 1))

heappush(h,('V', 2))

heappush(h,('V', 3))

for i in h:

print(i)

'''

22. Write a Python program to push three items into a heap and

return the smallest item from the heap.

Also, pop and return the smallest item from the heap.

Sample Output:

Items in the heap:

('V', 1)

('V', 3)

('V', 2)

----------------------

The smallest item in the heap:

('V', 1)

----------------------

Pop the smallest item in the heap:

('V', 2)

('V', 3)

'''

import heapq as hq

h=[]

hq.heappush(h,('V', 1))

hq.heappush(h,('V', 3))

hq.heappush(h,('V', 2))

print(h)

print('the smallest item in the heap:\n',hq.nsmallest(1,h))

print('--------------------------')

hq.heappop(h)

print('pop the smallest item in the heap:\n',h)

print()

#Textbook

import heapq

heap = []

heapq.heappush(heap, ('V', 3))

heapq.heappush(heap, ('V', 2))

heapq.heappush(heap, ('V', 1))

print("Items in the heap:")

for a in heap:

print(a)

print("----------------------")

print("The smallest item in the heap:")

print(heap[0])

print("----------------------")

print("Pop the smallest item in the heap:")

heapq.heappop(heap)

for a in heap:

print(a)

'''

23. Write a Python program to push an item on the heap,

then pop and return the smallest item from the heap.

Sample Output:

Items in the heap:

('V', 1)

('V', 3)

('V', 2)

----------------------

Using heappushpop push item on the heap and return the smallest item.

('V', 2)

('V', 3)

('V', 6)

'''

import heapq as hq

h=[]

hq.heappush(h,('V', 1))

hq.heappush(h,('V', 3))

hq.heappush(h,('V', 2))

#heappushpop: push and pop smallest

queqe=[('V', 2),('V', 3),('V', 6)]

for i in queqe:

smallest=hq.heappushpop(h,i)

print(smallest)

print(h) #[('V', 3), ('V', 3), ('V', 6)]

print()

#Textbook

import heapq

heap = []

heapq.heappush(heap, ('V', 3))

heapq.heappush(heap, ('V', 2))

heapq.heappush(heap, ('V', 1))

print("Items in the heap:")

for a in heap:

print(a) #sequence: 1,3,2

print("----------------------")

print("Using heappushpop push item on the heap and return the smallest item.")

heapq.heappushpop(heap, ('V', 6)) # 去1, 3,2,6重排

for a in heap:

print(a) #output: 2,3,6

'''

24. Write a Python program to create a heapsort, pushing all values

onto a heap and then popping off the smallest values one at a time.

Sample Output:

[10, 20, 20, 40, 50, 50, 60, 70, 80, 90, 100]

'''

import heapq as hq

h=[]

def heapsort(iterable):

for i in iterable:

hq.heappush(h,i)

return [hq.heappop(h) for j in range(len(h))]

print(heapsort([60,40,30,20,10,90,80,70,100,50]))

'''

25. Write a Python program to get the two largest and

three smallest items from a dataset.

Sample Output:

[100, 90] [10, 20, 20]

'''

import heapq as hq

lst1=[10,20,30,20,40,50,60,70,80,90,100]

hq.heapify(lst1)

print(hq.nlargest(2,lst1),hq.nsmallest(3,lst1))

#Textbook- no need to heapify first

import heapq

h = [10, 20, 50, 70, 90, 20, 50, 40, 60, 80, 100]

print(heapq.nlargest(2,h))

print(heapq.nsmallest(3,h))

'''

26. Write a Python program to create a queue and display all the members and the size of the queue.

Sample Output:

Members of the queue:

0 1 2 3

Size of the queue:

4

'''

import heapq as hq

lst1=[10,90,20,60,30,20,40,50,60,70,80,90,100]

hq.heapify(lst1)

print(\*lst1)

print(len(lst1))

import queue

q = queue.Queue() #q is a class of queue

for x in range(4):

q.put(x)

print("Members of the queue:")

# y=z=q.qsize()

for n in q.queue:

print(n, end=" ")

print("\nSize of the queue:")

print(q.qsize())

#chatGPT

'''queue provides a variety of thread-safe data structures, including a priority queue, while heapq provides functions for working with heaps, which can be used to implement a priority queue but do not provide thread-safety guarantees.

what is 'thread-safety'?

Thread-safety is a concept in computer programming that refers to the ability of a program or data structure to be used safely by multiple threads simultaneously without causing errors or unexpected behavior. In a multi-threaded program, multiple threads can access the same resources, such as memory or data structures, at the same time. Thread-safety ensures that these resources are accessed in a way that prevents conflicts and ensures that each thread behaves properly and fulfills its design specifications without unintended interaction

A thread-safe program or data structure is designed to handle concurrent access by multiple threads in a way that avoids data races, which are situations in which data are set to either correct or incorrect values, depending upon the order in which multiple threads access them. Achieving thread-safety often involves the use of synchronization mechanisms, such as locks or semaphores, to ensure that only one thread can access a shared resource at a time.

why heapq cannot provide 'thread-safe- structures'?

The heapq module in Python does not provide thread-safe structures because none of the heap operations are intrinsically thread-safe1. This means that if multiple threads access the same heap structure simultaneously, it can lead to race conditions and other synchronization issues. Therefore, a user would need to put locks around the calls or isolate them to a single thread to ensure thread-safety1.

In contrast, the PriorityQueue class in the queue module provides a thread-safe implementation of a priority queue2. The PriorityQueue class is essentially a heapq implementation that is wrapped in a thread-safe interface provided by the queue module3. The PriorityQueue class uses a lock to coordinate access to the underlying heap structure, ensuring that only one thread can access the heap at a time

'''

'''

27. Write a Python program to find out whether a queue is empty or not.

Sample Output:

True

False

'''

import queue

q1=queue.Queue()

q2=queue.Queue()

for i in range(3):

q2.put(i)

print(q1.queue,len(q1.queue)==0)

print(q2.queue,len(q2.queue)==0)

#Textbook

import queue

p = queue.Queue()

q = queue.Queue()

for x in range(4):

q.put(x)

print(p.empty())

print(q.empty())

'''

28. Write a Python program to create a FIFO queue.

Sample Output:

0 1 2 3

'''

import queue

lst1=[2,1,3,0]

q=queue.Queue()

for i in lst1:

q.put(i)

for j in q.queue:

print(j,end=' ') #2 1 3 0 未排!

print()

while not q.empty():

print(q.get(), end=" ") # still 2 1 3 0 未排!

#Textbook

import queue

q = queue.Queue()

#insert items at the end of the queue

for x in range(4):

q.put(str(x))

#remove items from the head of the queue

while not q.empty():

print(q.get(), end=" ")

print("\n")

#PriorityQueue can fix this!

lst1 = [2, 1, 3, 0]

q = queue.PriorityQueue()

# Put items in the queue

for i in lst1:

q.put(i)

# Print sorted items

while not q.empty():

print(q.get(), end=' ')

print()

'''

29. Write a Python program to create a LIFO queue.

Sample Output:

3 2 1 0

'''

import queue

q=queue.Queue()

for i in range(4):

q.put(i)

lst1=list(q.queue)

lst1.reverse()

for j in lst1:

print(j)

#Textbook

import queue

q = queue.LifoQueue()

#insert items at the head of the queue

for x in range(4):

q.put(str(x))

#remove items from the head of the queue

while not q.empty():

print(q.get(), end=" ")

print()