Priority Queue

- ADT: Abstract data type
- A queue where each object has an associated priority attached
- Operation
  - Insert: with a priority value
  - Remove: max/min
  - Get/Peek: max/min value

Can be implemented using simple sorted array, heap or using balanced BSTs

simple sorted array

- Insert O(N)
- Remove O(1)
- Peek O(1)

HEAP

- Insert O(log N)
- Remove O(log N)
- Peek O(1)

Balanced BSTs

- Insert O(log N)
- Remove O(log N)
- Peek O(log N)

Practically Heap is the prefered implementation over BST as Heap uses an array which gives better locality of reference (caching) Hence better practical performance even though complexity is same.

In [ ]:

In [ ]:

Priority Queue in different languages

In [ ]:

Python

In [1]:
```
import heapq

data = [1,2,3,4] # list/array
heapq.heapify(data)
print(data)

# min heap max heap? min heap
print()
data = [4,3,2,1]
heapq.heapify(data)
print(data)


print()
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
```

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

[1, 3, 2, 4]

pop 1 [2, 3, 4]
pop 2 [3, 4]
pop 3 [4]
pop 4 []
```

In [ ]:
```

```

In [2]:
```
data = [4,3,2,1]
heapq.heapify(data)
print(data)



print()
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
```

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

pop 1 [2, 3, 4]
pop 2 [3, 4]
pop 3 [4]
pop 4 []
```

In [ ]:

In [3]:
```
h = []
heapq.heappush(h, 10)
print(h)
heapq.heappush(h, 1)
print(h)
print(h[0])
```

```
[10]
[1, 10]
1
```

In [ ]:

In [5]:
```
# Heapify before doing heappop
data = [5,3,4,1]
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)


print()
data = [5,3,4,1]
heapq.heapify(data)
print(data)
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
```

```
pop 5 [1, 3, 4]
pop 1 [3, 4]
pop 3 [4]
pop 4 []

[1, 3, 4, 5]
pop 1 [3, 5, 4]
pop 3 [4, 5]
pop 4 [5]
pop 5 []
```

In [ ]:

Strings and heap

In [ ]:
```
# min heap
# data  1 4 3 6 5 8 7
# idx   0 1 2 3 4 5 6
# p-idx   0 0 1 1 2 2


      1
    4   3
   6 5  8 7
```

In [ ]:
```

```

In [6]:
```
data = ["a", "c", "d", "b",  "e"]
heapq.heapify(data)
print(data)
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
# a b d c e
# 0 1 2 3 4
#   0 0 1 1

#         a
#       b    d
#     c   e
```

```
['a', 'b', 'd', 'c', 'e']
pop a ['b', 'c', 'd', 'e']
pop b ['c', 'e', 'd']
pop c ['d', 'e']
pop d ['e']
```

In [2]:
```
"a" < "b"
```

Out[2]:
```
True
```

In [ ]:
```

```

In [ ]:
```
data = ["a", "c", "d", "b",  "e"]
heapq.heapify(data)
print(data)

for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
```

In [ ]:
```

```

Custom Objects and Heap

In [7]:
```
# min heap example using age for priority
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age =  age

    def __lt__(self, other): # less than
        return self.age < other.age

    def __str__(self):
        return f"({self.name}, {self.age})"

    def __repr__(self):
        return self.__str__()


data = [Person("gaurav",1), Person("abhishek",7), Person("manu",2), Person("Ni

heapq.heapify(data)
print(data)
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
```

```
[(gaurav, 1), (Niraj, 4), (manu, 2), (abhishek, 7)]
pop (gaurav, 1) [(manu, 2), (Niraj, 4), (abhishek, 7)]
pop (manu, 2) [(Niraj, 4), (abhishek, 7)]
pop (Niraj, 4) [(abhishek, 7)]
pop (abhishek, 7) []
```

In [ ]:

Python heap as max heap

In [8]:
```python
# max heap example using age for priority
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age =  age

    def __lt__(self, other): # less than
        return self.age > other.age

    def __str__(self):
        return f"({self.name}, {self.age})"

    def __repr__(self):
        return self.__str__()

# use age for priority
data = [Person("gaurav",1), Person("abhishek",7), Person("manu",2), Person("Ni

heapq.heapify(data)
print(data)
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
```

```
[(abhishek, 7), (Niraj, 4), (manu, 2), (gaurav, 1)]
pop (abhishek, 7) [(Niraj, 4), (gaurav, 1), (manu, 2)]
pop (Niraj, 4) [(manu, 2), (gaurav, 1)]
pop (manu, 2) [(gaurav, 1)]
pop (gaurav, 1) []
```

In [31]:
```
# min heap example using name for priority
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age =  age

    def __lt__(self, other): # less than
        return self.name < other.name

    def __str__(self):
        return f"({self.name}, {self.age})"

    def __repr__(self):
        return self.__str__()

# use age for priority
data = [Person("gaurav",1), Person("abhishek",7), Person("manu",2), Person("ni

heapq.heapify(data)
print(data)
for _ in range(4): # i = 0, 1, 2 3
    r = heapq.heappop(data)
    print("pop", r, data)
```

```
[(abhishek, 7), (gaurav, 1), (manu, 2), (niraj, 4)]
pop (abhishek, 7) [(gaurav, 1), (niraj, 4), (manu, 2)]
pop (gaurav, 1) [(manu, 2), (niraj, 4)]
pop (manu, 2) [(niraj, 4)]
pop (niraj, 4) []
```

In [32]:
```
import heapq
heap=[]
heapq.heappush(heap,11)
heapq.heappush(heap,2)
heapq.heappush(heap,12)
heapq.heappush(heap,6)
heapq._heapify_max(heap)
print(heap)
```

```
[12, 11, 2, 6]
```

In [ ]:
```
```

JAVA

default min heap

```
PriorityQueue<Integer> pq=new PriorityQueue<>();

for(int i=5; i>= 1;i--)
{
  pq.add(i);
}

while(!pq.isEmpty())
{
  System.out.println(pq.poll());
}
```

default min heap

```
int data[]={1,2,3,5,4};

PriorityQueue<Integer> pq=new PriorityQueue<>();

// copy data
for(int i=0;i< data.length;i++)
{
  pq.add(data[i]);
}

while(!pq.isEmpty())
{
  System.out.println(pq.poll());
}
```

max heap using comparator

```
int data[]={1,2,3,5,4};

PriorityQueue<Integer> pq=new PriorityQueue<Integer>((o1,o2)->  o2
-o1);
for(int i=0;i< data.length;i++)
{
  pq.add(data[i]);
}
```

In [10]:

```
def do(data, f):
    for d in data:
        print(f(d))

def mul(x,y):
    return x*y

do([1,-2,3], lambda x: x*x)
print()
do([1,-2,3], lambda x: -x)
print()
do([1,-2,3], mul)
```

```
1
4
9

-1
2
-3


---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 12
     10 do([1,-2,3], lambda x: -x)
     11 print()
---> 12 do([1,-2,3], mul)

Cell In[10], line 3, in do(data, f)
      1 def do(data, f):
      2     for d in data:
----> 3         print(f(d))

TypeError: mul() missing 1 required positional argument: 'y'
```

In [ ]:

C++

```
#include <iostream>
#include <queue>
using namespace std;

int main()
{
      int arr[]={1,2,3,4,5};

    priority_queue<int> pq(arr, arr+5);
    cout<<"Max priority queue: ";
      while(!pq.empty()){
      cout<<pq.top()<<endl;
      pq.pop();
    }


    priority_queue <int, vector<int>, greater<int> > pq1(arr,arr+5);
    cout<<"Min priority queue: ";
    while(!pq1.empty()){
    cout<<pq1.top()<<endl;
    pq1.pop();
    }

}
```

C#

min heap

```
PriorityQueue<string, int> queue = new PriorityQueue<string, int>();
queue.Enqueue("Item A", 1);
queue.Enqueue("Item B", 2);
queue.Enqueue("Item C", 3);
queue.Enqueue("Item D", 5);
queue.Enqueue("Item E", 4);

while (queue.TryDequeue(out string item, out int priority))
{
    Console.WriteLine($"Popped Item : {item}. Priority Was : {priorit
y}");
}
```

Javascript

In [ ]:

- heapify -> convert a sequence/array to satisfy heap property: O(N)
- heappop: assumes data is in heap format/heapified: O(log N)
- heappush: assumes data is in heap format/heapified: O(log N)
- peek: O(1)

In [ ]:

https://leetcode.com/problems/kth-largest-element-in-an-array/ (https://leetcode.com/problems/kth-largest-element-in-an-array/)

Solution-1:

- sort data : O(n log n)
- get kth element from end: O(1)

TC:O(n log n)
SC: O(1)

Solution-2:

- Max heap of entire data: O(N)
- Heappop(k-1 times) : O(k log n)
- Top: O(1)

TC: O(N) + O(k log n)
if n >> k O(N)
if k ~ n O(N log N)
SC: O(N)

Solution-3:

- Make a min heap of first k elements: O(k)
- For each remaining element pop if current current element > heap.top() and push(current element): (N-k) log k
- get top of heap O(1)

TC: O(n log k): O(n log n) if k ~n SC: O(k)

In [ ]:

In [ ]:

https://leetcode.com/problems/merge-k-sorted-lists/ (https://leetcode.com/problems/merge-k-sorted-lists/)

K lists.
Each of size N.

Solution-1

- Merge all lists pair wise till we have only one list left

TC: O(nk log k) SC: O(k)

Solution-2

- Join all list: O(nk)
- Sort all data(size is now nk): O(nk log nk)

TC: (nk log nk)

Solution-3: based on merging 2 sorted arrays: pick the smallest element from k lists

- Maintain a list of current index for all lists
- Pick the smallest element from all lists. Increment the pointer/index for the list where min element is picked O(k)
- repeat above step for all data elements/ till all lists are exhausted.

TC: O(n * k^2)

Solution-4

- use a heap to maintain the minimum of value from all the k lists. Similar to solution-3

TC: O(nk log k) SC: O(k)

In [ ]:
```
# Solution
# Use the logic to merge 2 sorted lists pairwise

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next

# n
# n O(n) + O(2N) + O(3n) + ... O(kn) => n (O(1) + O(2) + ... O(k)) => O(n*k(k+
# n O(3n)
# n O(4n)
# n..
# ..
# n O(kn)

class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNod
        if len(lists) == 0:
            return None

        while len(lists) > 1:
            l1 = lists.pop()
            l2 = lists.pop()
            lists.append(self.merge2List(l1, l2))

        return lists[0]

    def merge2List(self, l1, l2):
        head = None
        curr = None

        if l1 is None:
            return l2
        if l2 is None:
            return l1

        while l1 is not None and l2 is not None:
            temp = None
            if l1.val < l2.val:
                temp = l1
                l1  = l1.next
            else:
                temp = l2
                l2= l2.next

            if head is not None:
                curr.next = temp
                curr  = temp
            else:
                curr = temp
                head = temp

        if l1 is not None:
            curr.next = l1
```

```
        if l2 is not None:
            curr.next = l2

        return head
```

In [ ]:

```
// Solution-3
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {

        if (lists.size() == 0) {
            return NULL;
        }

        ListNode *head  = NULL;
        ListNode *curr  = NULL;

        while (true) {

            // Find the list which has minimum first element
            int minIdx = -1;
            for (int i = 0; i < lists.size(); i++) {
                if (lists[i] == NULL) {
                    continue;
                }

                if (minIdx == -1) {
                    minIdx = i;
                } else if (lists[minIdx]->val > lists[i]->val) {
                    minIdx = i;
                }
            }

            // No list with min found => all lists are exhausted
            if (minIdx == -1) {
                break;
            }

            if (head == NULL) {
                head = lists[minIdx];
                curr = lists[minIdx];
            } else {
                curr->next = lists[minIdx];
                curr = lists[minIdx];
            }
            // advance the ponter for the list which contains min element
            lists[minIdx] = lists[minIdx]->next;

        }

        return head;
```

```
    }
};
```

In [ ]:
```
NULL
NULL
7->9

[NULL,NULL,NULL]
 0 1 2

2


head -> 1
curr -> 1->2->2->3->4->5->5->7->9
```

In [ ]:

Sort nearly k sorted array

Given a k–sorted array that is almost sorted such that each of the n elements may be misplaced by no more than k positions from the correct sorted order. Find a space-and-time efficient algorithm to sort the array.

For example,

Input:

arr = [1, 4, 5, 2, 3, 7, 8, 6, 10, 9] k = 2

Output:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Solution-1: using O(n log n) sorting algorithm

Solution-2: use the common n^2 sorting algorithms

- Bubble: O(n^2) X
- Insertion: O(n*k)
- Selection: O(n^2) X

Solution-3: Use a min heap

- put first k+1 elements in the heap
- pop the min element O(log k)
- add the next element into the heap O(log k)
  TC: O(n log k)
  SC: O(k)

In [ ]:

In [ ]:

Skyline: https://leetcode.com/problems/the-skyline-problem/ (https://leetcode.com/problems/the-skyline-problem/)

https://leetcode.com/problems/sliding-window-maximum/description/ (https://leetcode.com/problems/sliding-window-maximum/description/)