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

K lists.
Each of size N.

```
k lists of length n

Solution-0:
    Join all linked list into 1 list
    Sort the single linked list O(nk log nk)

Solution-1: Merge lists sequentially till left with 1 list [k-1 merge
s]
    O(2n) + O(3n) + ... O(kn)
     O(n) (2 + 3 + 4... k)
        O(n) k(k+1)/2 => O(n) ((k^2)/2 + k/2) => O(n) O(k^2) => O(n *
k^2)


             l1 l2 l3 l4 l5 l6 l7 l8
                12  3 4 5 6 7 8
                123  4 5 6 7 8
                   ....
                    12345678

Solution-2: Merge lists pairwise till left with 1 list [k-1 merges]
        O(k*n) * log k => O(nk log k)

               1 2 3 4 5 6 7 8
                12  34  56 78 -> n*k
                  1234   5678 -> n*k
                    12345678

Solution-3:
    Store firt pointer of each list in an array
    Find min from array (size k)
    Put min into result.
    Put min->next ptr back into the list
    O(N*k)
    N = Total elements in all lists
    k = Number of LL

Solution-4: Heap
    Store firt pointer of each list in heap
    Find min from heap (size k)
    Put min into result.
    Put min->next ptr back into the heap if it's not  NULL
    O(N * log k)
    N = Total elements in all lists
    k = Number of LL
```

```
struct Cmp {
    bool operator()(ListNode* p1, ListNode*p2) {
        return p1->val > p2->val;
    }
};


class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        // merge 2 sorted arrays: O(n+m)


        // build a min heap
        // with custom comparator to work on value of node pointer,
        priority_queue<ListNode*, vector<ListNode*>,  Cmp > heap;

        // push only those pointers which are not NULL
        for(auto it = lists.begin(); it != lists.end(); it++) {
            if (*it != NULL) {
                heap.push(*it);
            }
        }

        ListNode* res = NULL, *curr = NULL;
        while(!heap.empty()) {
            if (curr == NULL)
                curr = heap.top();
            else {
                curr->next = heap.top();
                curr = curr->next;
            }

            if (res == NULL) {
                res = curr;
            }

            if (curr->next) {
                heap.push(curr->next);
            }
            heap.pop();
        }

        return res;
    }
};
```

```
import heapq
class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optiona
l[ListNode]:

        ListNode.__lt__ = lambda x,y: x.val < y.val

        ## [[]]
        # [None]
        # vector {NULL}
        heap = []
        for ptr in lists:
            if ptr is not None:
                heapq.heappush(heap, ptr)

        res = None
        prev = None
        while (len(heap) != 0 ):

            curr = heapq.heappop(heap)
            if res is None:
                res = curr

            if curr.next != None:
                heapq.heappush(heap, curr.next)

            if prev is not None:
                prev.next = curr
            prev = curr

        return res
```

In [ ]:
```
1
```

In [ ]:
```
1
```

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]

C++

```
void sort_k_sorted(int arr[], int k) {}

void print(int arr[], int n) {
    for (int i = 0; i < n; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int input1[] = {6, 5, 3, 2, 8, 10, 9};

    sort_k_sorted(input1, 3);
    print(input1, 7);


    int input2[] = {1, 4, 5, 2, 3, 7, 8, 6, 10, 9};

    sort_k_sorted(input2, 3);
    print(input2, 10);
}
```

python

```
def sort_k_sorted(data, k):
    pass

input = [6, 5, 3, 2, 8, 10, 9]
sort_k_sorted(input, 3)
print(input)


input = [1, 4, 5, 2, 3, 7, 8, 6, 10, 9]
sort_k_sorted(input, 2)
print(input)
```

In [ ]:
```
1
```

In [ ]:
```
1
```

```
{6, 5, 3, 2, 8, 10, 9}, K = 3

p a b c

[6 5 3 2]

2 [6 5 3 8]

2 3 [6 5 8 10]

2 3 5 [6 8 10 9]

2 3 4 6 [8 10 9]
2 3 4 6 8 [10 9]
2 3 4 6 8 9 [10]
2 3 4 6 8 9 10 []

Use min heap when solving L->R
1. Push k + 1 elements in heap
2. while (heap ! empty)
    a. pop from heap -> push to result -> i
    b. if not reached end of array: push to heap -> j


Use max heap when solving R->L

TC: O(n log k)
SC: O(k)
```

Java

```
class Solution
{
    //Function to return the sorted array.
    ArrayList <Integer> nearlySorted(int arr[], int num, int k)
    {
```

```
C++
```C++
class Solution
{
    public:
    //Function to return the sorted array.
    vector <int> nearlySorted(int arr[], int num, int K){
    vector <int> ans;
    priority_queue<int,vector<int>,greater<int>> pq(arr,arr+K+1);
    int count=K+1;
    while(!pq.empty())
    {
        ans.push_back(pq.top());
        pq.pop();
        if(count<num)
            pq.push(arr[count]);
        count++;

    }

    return ans;
    }
};
```
```

Python

```
def sort_k_sorted(data, k):
    res = []
    a = []
    for i in range(k+1):
        heapq.heappush(a, data[i])
    i = k
    while a:
        res.append(heapq.heappop(a))
        if i < len(data):
            heapq.heappush(a, data[i])
            i += 1
    return res
```

```
class Solution
{
    public:
    vector <int> nearlySorted(int arr[], int num, int K){

        int correct_k = (num == k) ? k : k+1;
        priorty_queue<int, vector<int>, greater<int> > heap(arr, arr+
correct_k);

        int count = 0;

        for (int i = correct_k; i < n; i++) {
            arr[count++] = heap.top();
            heap.pop();
            heap.push(arr[i]);
        }

        while (heap.empty() == false) {
            arr[count++] = heap.top();
            pheapq.pop();
        }
    }
};
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Question Skyline

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

```
buildings = [[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
Output: [[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]


        // start=building[0][0] ?
        // end = find max building[i][1]?

        // map = [start.. end] = 0


        // for each building:
        //     for j=building[i][0] till building[i][1]:
        //       map[j] = max(map[j], building[i][2])

        // [2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]

        // [2,10,I] [9,10,D], [3,15, I], [7,15,D], [5,12,I][12,12,D],
[15,10, I],[20,10,D],[19,8, I], [24,8, D]

        // a[0] == b[0] -> a[1] == b[1] ? a[2] > b[2] : a[1] > b[1]
        // a[0] < b[0]

        // [2,10,I] [3,15, I] [5,12,I] [7,15,D][9,10,D] [12,12,D],[1
5,10,I] [19,8,I], [20,10,D],[24,8, D]

        // prev = 0
        // [2,10,I] -> (10) [2,10]
```

In [ ]:
```
1
```