# Sort Complexity

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Algorithm | Complexity | Worst Case | How the partial result looks like |
| Selection | O(n^2) | O(n^2) | Always the minimum is in partial result |
| Insertion | O(n^2) | O(n^2) | Partial result is always sorted, but minimum is not necessary in partial result. |
| Bubble | O(n^2) | O(n^2) | In every iteration, the larger number is exchange one level up. |
| Merge sort | O(nlog(n)) | O(nlog(n)) | You first get a sorted pair of two, then group of 4, then group of 8… |
| Quick Sort | O(nlog(n)) | O(n^2) | The first number in the list will be put in middle, with left part less than it, right part greater than it, slowly you are building a BST. |
| Heap sort | O(nlog(n)) | O(nlog(n)) | close to quick sort, but only the left part build up the BST.  Please remember if you have a small M groups of sorted group, heap sort can give you a O(n) solution. |
| Binary Tree sort | O(nlog(n)) | O(n^2) | Almost same as quick sort |
| Bucket sort | O(n)\*O(mlog(m)) | O(n)\*O(mlog(m)) | bucket sort is normally used for a large data which should be sorted externally. |

In the above list, you need to pay attention to **merge sort, heap sort, tree sort** and **bucket sort**

# Heap Sort

When we discuss sort problem, no need to say the heap sort is the top one pattern. We put all the related data in heap and poll them one by one. We can use either TreeMap or priority queue, just watch that the TreeMap (map or set) is single key based, if you have multiple key with same value consider either priority queue or multi\_map (multi\_set). By default the priority queue is from high to low, if you want to do it in reverse, make the negative.

## 683. K Empty Slots

Hard

You have N bulbs in a row numbered from 1 to N. Initially, all the bulbs are turned off. We turn on exactly one bulb everyday until all bulbs are on after N days.

You are given an array bulbs of length N where bulbs[i] = x means that on the (i+1)th day, we will turn on the bulb at position x where i is 0-indexed and x is 1-indexed.

Given an integer K, find out the **minimum day number** such that there exists two **turned on** bulbs that have **exactly** K bulbs between them that are **all turned off**.

If there isn't such day, return -1.

**Example 1:**

**Input:**

bulbs: [1,3,2]

K: 1

**Output:** 2

**Explanation:**

On the first day: bulbs[0] = 1, first bulb is turned on: [1,0,0]

On the second day: bulbs[1] = 3, third bulb is turned on: [1,0,1]

On the third day: bulbs[2] = 2, second bulb is turned on: [1,1,1]

We return 2 because on the second day, there were two on bulbs with one off bulb between them.

**Example 2:**

**Input:**

bulbs: [1,2,3]

K: 1

**Output:** -1

**Note:**

1. 1 <= N <= 20000
2. 1 <= bulbs[i] <= N
3. bulbs is a permutation of numbers from 1 to N.
4. 0 <= K <= 20000

### Analysis:

You simulate the bulb which opens every day by putting the open flower index in a heap and look back and forward until you see exactly K empty slot.

/// <summary>

/// Leet code #683. K Empty Slots

///

/// There is a garden with N slots. In each slot, there is a flower. The

/// N flowers will bloom one by one in N days. In each day, there will be

/// exactly one flower blooming and it will be in the status of blooming

/// since then.

///

/// Given an array flowers consists of number from 1 to N. Each number in

/// the array represents the place where the flower will open in that day.

///

/// For example, flowers[i] = x means that the unique flower that blooms

/// at day i will be at position x, where i and x will be in the range

/// from 1 to N.

///

/// Also given an integer k, you need to output in which day there exists

/// two flowers in the status of blooming, and also the number of flowers

/// between them is k and these flowers are not blooming.

///

/// If there isn't such day, output -1.

///

/// Example 1:

/// Input:

/// flowers: [1,3,2]

/// k: 1

/// Output: 2

/// Explanation: In the second day, the first and the third flower have

/// become blooming.

///

/// Example 2:

/// Input:

/// flowers: [1,2,3]

/// k: 1

/// Output: -1

///

/// Note:

/// The given array will be in the range [1, 20000].

/// </summary>

int LeetCode::kEmptySlots(vector<int>& flowers, int k)

{

int result = -1;

set<int> sorted\_flowers;

for (size\_t i = 0; i < flowers.size(); i++)

{

// insert the flower first then search for position

sorted\_flowers.insert(flowers[i]);

set<int>::iterator itr = sorted\_flowers.find(flowers[i]);

if (itr != sorted\_flowers.begin())

{

set<int>::iterator prev = itr;

prev--;

if ((\*itr - \*prev) == k + 1)

{

result = i + 1;

break;

}

}

set<int>::iterator next = itr;

next++;

if (next != sorted\_flowers.end())

{

if ((\*next - \*itr) == k + 1)

{

result = i + 1;

break;

}

}

}

return result;

}

## 846. Hand of Straights

Medium

Alice has a hand of cards, given as an array of integers.

Now she wants to rearrange the cards into groups so that each group is size W, and consists of W consecutive cards.

Return true if and only if she can.

**Example 1:**

**Input:** hand = [1,2,3,6,2,3,4,7,8], W = 3

**Output:** true

**Explanation:** Alice's hand can be rearranged as [1,2,3],[2,3,4],[6,7,8].

**Example 2:**

**Input:** hand = [1,2,3,4,5], W = 4

**Output:** false

**Explanation:** Alice's hand can't be rearranged into groups of 4.

**Note:**

1. 1 <= hand.length <= 10000
2. 0 <= hand[i] <= 10^9
3. 1 <= W <= hand.length

### Analysis:

Sort the hand in a priority queue or TreeMap, and collect from the lowest number with repeated with window size of W.

/// <summary>

/// Leet code #846. Hand of Straights

///

/// Alice has a hand of cards, given as an array of integers.

///

/// Now she wants to rearrange the cards into groups so that each group

/// is size W, and consists of W consecutive cards.

///

/// Return true if and only if she can.

///

/// Example 1:

/// Input: hand = [1,2,3,6,2,3,4,7,8], W = 3

/// Output: true

/// Explanation: Alice's hand can be rearranged as [1,2,3],[2,3,4],[6,7,8].

///

/// Example 2:

/// Input: hand = [1,2,3,4,5], W = 4

/// Output: false

/// Explanation: Alice's hand can't be rearranged into groups of 4.

///

/// Note:

/// 1. 1 <= hand.length <= 10000

/// 2. 0 <= hand[i] <= 10^9

/// 3. 1 <= W <= hand.length

/// </summary>

bool LeetCodeSort::isNStraightHand(vector<int>& hand, int W)

{

map<int, int> hand\_map;

for (size\_t i = 0; i < hand.size(); i++)

{

hand\_map[hand[i]]++;

}

while (!hand\_map.empty())

{

int start = hand\_map.begin()->first;

int count = hand\_map.begin()->second;

for (int i = 0; i < W; i++)

{

hand\_map[start + i] -= count;

if (hand\_map[start + i] == 0)

{

hand\_map.erase(start + i);

}

else if (hand\_map[start + i] < 0)

{

return false;

}

}

}

return true;

}

## 857. Minimum Cost to Hire K Workers

Hard

There are N workers.  The i-th worker has a quality[i] and a minimum wage expectation wage[i].

Now we want to hire exactly K workers to form a *paid group*.  When hiring a group of K workers, we must pay them according to the following rules:

1. Every worker in the paid group should be paid in the ratio of their quality compared to other workers in the paid group.
2. Every worker in the paid group must be paid at least their minimum wage expectation.

Return the least amount of money needed to form a paid group satisfying the above conditions.

**Example 1:**

**Input:** quality = [10,20,5], wage = [70,50,30], K = 2

**Output:** 105.00000

**Explanation**: We pay 70 to 0-th worker and 35 to 2-th worker.

**Example 2:**

**Input:** quality = [3,1,10,10,1], wage = [4,8,2,2,7], K = 3

**Output:** 30.66667

**Explanation**: We pay 4 to 0-th worker, 13.33333 to 2-th and 3-th workers seperately.

**Note:**

1. 1 <= K <= N <= 10000, where N = quality.length = wage.length
2. 1 <= quality[i] <= 10000
3. 1 <= wage[i] <= 10000
4. Answers within 10^-5 of the correct answer will be considered correct.

### Analysis:

We sort by ratio and select the first K workers, and use the highest ratio to adjust the remaining wage, all those wages should be increased because they have lower ratio. But this is not necessarily the best answer. In the next step you pick the workers with higher ratio, who may have lower salary, and replace the selected one who has the highest quality, (i.e. highest wage) and adjust the wage for the selected workers, to do so you do not need to remember wage for everyone, just remember the total quality the selected workers generated. Keep on doing so and record the minimum total wage, you will finally get the answer.

/// <summary>

/// Leet code #857. Minimum Cost to Hire K Workers

///

/// There are N workers. The i-th worker has a quality[i] and a minimum

/// wage expectation wage[i].

///

/// Now we want to hire exactly K workers to form a paid group. When hiring

/// a group of K workers, we must pay them according to the following rules:

///

/// Every worker in the paid group should be paid in the ratio of their

/// quality compared to other workers in the paid group.

/// Every worker in the paid group must be paid at least their minimum wage

/// expectation.

/// Return the least amount of money needed to form a paid group satisfying

/// the above conditions.

///

///

/// Example 1:

/// Input: quality = [10,20,5], wage = [70,50,30], K = 2

/// Output: 105.00000

/// Explanation: We pay 70 to 0-th worker and 35 to 2-th worker.

///

/// Example 2:

/// Input: quality = [3,1,10,10,1], wage = [4,8,2,2,7], K = 3

/// Output: 30.66667

/// Explanation: We pay 4 to 0-th worker, 13.33333 to 2-th and 3-th workers

/// seperately.

///

///

/// Note:

///

/// 1. 1 <= K <= N <= 10000, where N = quality.length = wage.length

/// 2. 1 <= quality[i] <= 10000

/// 3. 1 <= wage[i] <= 10000

/// 4. Answers within 10^-5 of the correct answer will be considered correct.

/// </summary>

double LeetCodeSort::mincostToHireWorkers(vector<int>& quality, vector<int>& wage, int K)

{

vector<pair<double, int>> ratio\_list;

priority\_queue<int> quality\_heap;

for (size\_t i = 0; i < quality.size(); i++)

{

ratio\_list.push\_back(make\_pair((double)wage[i] /

(double)quality[i], quality[i]));

}

sort(ratio\_list.begin(), ratio\_list.end());

unsigned long long sum\_quality = 0;

// if we select K worker, the highest ratio worker is paid as minimum wage,

// for any lower wage, they are upgrade with highest ratio

for (size\_t i = 0; i < (size\_t)K; i++)

{

quality\_heap.push(ratio\_list[i].second);

sum\_quality += ratio\_list[i].second;

}

double ratio = ratio\_list[K - 1].first;

double result = ratio \* sum\_quality;

// we keep on raise the ratio, but because we may have the lower quality

// and wage, we may end up with lower minimum total wage

for (size\_t i = K; i < quality.size(); i++)

{

sum\_quality -= quality\_heap.top();

quality\_heap.pop();

quality\_heap.push(ratio\_list[i].second);

sum\_quality += ratio\_list[i].second;

ratio = ratio\_list[i].first;

result = min(result, ratio \* sum\_quality);

}

return result;

}

## 632. Smallest Range Covering Elements from K Lists

Hard

You have k lists of sorted integers in ascending order. Find the **smallest** range that includes at least one number from each of the k lists.

We define the range [a,b] is smaller than range [c,d] if b-a < d-c or a < c if b-a == d-c.

**Example 1:**

**Input:** [[4,10,15,24,26], [0,9,12,20], [5,18,22,30]]

**Output:** [20,24]

**Explanation:**

List 1: [4, 10, 15, 24,26], 24 is in range [20,24].

List 2: [0, 9, 12, 20], 20 is in range [20,24].

List 3: [5, 18, 22, 30], 22 is in range [20,24].

**Note:**

1. The given list may contain duplicates, so ascending order means >= here.
2. 1 <= k <= 3500
3. -105 <= value of elements <= 105.

### Analysis:

We pick the smallest number in each array, and put in in a priority queue, and keep on pop on the range head, and move to next number until we exhaust the array.

/// <summary>

/// Leet code #632. Smallest Range

///

/// You have k lists of sorted integers in ascending order. Find the

/// smallest range that includes at least one number from each of the

/// k lists.

/// We define the range [a,b] is smaller than range [c,d] if b-a <

/// d-c or a < c if b-a == d-c.

/// Example 1:

/// Input:[[4,10,15,24,26], [0,9,12,20], [5,18,22,30]]

/// Output: [20,24]

/// Explanation:

/// List 1: [4, 10, 15, 24,26], 24 is in range [20,24].

/// List 2: [0, 9, 12, 20], 20 is in range [20,24].

/// List 3: [5, 18, 22, 30], 22 is in range [20,24].

///

/// Note:

/// The given list may contain duplicates, so ascending order means

/// >= here.

/// 1 <= k <= 3500

/// -10^5 <= value of elements <= 10^5.

/// For Java users, please note that the input type has been changed to

/// List<List<Integer>>. And after you reset the code template, you'll see

/// this point.

/// </summary>

vector<int> LeetCodeSort::smallestRange(vector<vector<int>>& nums)

{

vector<int> range = { 0, INT\_MAX };

map<pair<int, int>, int> sort\_map;

for (size\_t i = 0; i < nums.size(); i++)

{

sort\_map[make\_pair(nums[i][0], i)] = 0;

}

while (true)

{

pair<int, int> start = sort\_map.begin()->first;

pair<int, int> end = sort\_map.rbegin()->first;

if (end.first - start.first < range[1] - range[0])

{

range[0] = start.first;

range[1] = end.first;

}

int index = sort\_map.begin()->second + 1;

// exhaust one list, game over

if (index == nums[start.second].size()) break;

// delete the head and move the position in that list to next

sort\_map.erase(start);

sort\_map[make\_pair(nums[start.second][index], start.second)] = index;

}

return range;

}

## 295. Find Median from Data Stream

Hard

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

For example,

[2,3,4], the median is 3

[2,3], the median is (2 + 3) / 2 = 2.5

Design a data structure that supports the following two operations:

* void addNum(int num) - Add a integer number from the data stream to the data structure.
* double findMedian() - Return the median of all elements so far.

**Example:**

addNum(1)

addNum(2)

findMedian() -> 1.5

addNum(3)

findMedian() -> 2

**Follow up:**

1. If all integer numbers from the stream are between 0 and 100, how would you optimize it?
2. If 99% of all integer numbers from the stream are between 0 and 100, how would you optimize it?

### Analysis:

This is a famous problem, to find the median you need to keep two heap, one for the smaller half, another is for the larger half, normally you keep these two heap size balanced, and when you need a median, you can take the largest from small heap and the smallest from the large heap and calculate the average.

Follow up: if the value is known in certain range you can do bucket count.

/// <summary>

/// Leet code #295. Find Median from Data Stream

///

/// Median is the middle value in an ordered integer list. If the size of

/// the list is even, there is no middle value. So the median is the mean

/// of the two middle value.

///

/// For example,

/// [2,3,4], the median is 3

///

/// [2,3], the median is (2 + 3) / 2 = 2.5

///

/// Design a data structure that supports the following two operations:

///

/// void addNum(int num) - Add a integer number from the data stream to

/// the data structure.

/// double findMedian() - Return the median of all elements so far.

///

/// Example:

///

/// addNum(1)

/// addNum(2)

/// findMedian() -> 1.5

/// addNum(3)

/// findMedian() -> 2

///

/// Follow up:

///

/// If all integer numbers from the stream are between 0 and 100, how

/// would you optimize it?

/// If 99% of all integer numbers from the stream are between 0 and

/// 100, how would you optimize it?

/// </summary>

class MedianFinder

{

private:

priority\_queue<int, vector<int>, greater<int>> m\_Large;

priority\_queue<int> m\_Small;

public:

// Default constructor.

MedianFinder()

{

}

// Adds a number into the data structure.

void addNum(int num)

{

if ((m\_Small.size() == 0) || (m\_Small.top() > num))

{

m\_Small.push(num);

}

else

{

m\_Large.push(num);

}

if (m\_Small.size() > m\_Large.size() + 1)

{

m\_Large.push(m\_Small.top());

m\_Small.pop();

}

if (m\_Large.size() > m\_Small.size())

{

m\_Small.push(m\_Large.top());

m\_Large.pop();

}

}

// Returns the median of current data stream

double findMedian()

{

double value;

if (m\_Small.size() == m\_Large.size() + 1)

{

value = (double)m\_Small.top();

}

else

{

value = ((double)m\_Small.top() + (double)m\_Large.top()) / 2;

}

return value;

}

};

# Bucket Sort

Bucket Sort is to divide the original data into multiple groups, the groups themselves are in order. After the data are in groups you can further do the next level bucket sort or simply select the minimum or maximum data in each group. Please look at the following example:

## 164. Maximum Gap

[Maximum Gap - LeetCode](https://leetcode.com/problems/maximum-gap/description/)

/// <summary>

/// Leet code #164. Maximum Gap

/// Given an unsorted array, find the maximum difference between the successive

/// elements in its sorted form.

/// Try to solve it in linear time/space.

/// Return 0 if the array contains less than 2 elements.

/// You may assume all elements in the array are non-negative integers and fit

/// in the 32-bit signed integer range.

/// </summary>

int LeetCode::maximumGap(vector<int>& nums)

{

if (nums.size() < 2) return 0;

int minimum = INT\_MAX;

int maximum = INT\_MIN;

for (size\_t i = 0; i < nums.size(); i++)

{

if (nums[i] < minimum) minimum = nums[i];

if (nums[i] > maximum) maximum = nums[i];

}

// The plus one is to make sure we do not have a slot\_gap as 0.

int slot\_gap = (maximum - minimum) / nums.size() + 1;

// In case the slot gap is not divisible by the range.

vector<pair<int, int>> slot((maximum - minimum) / slot\_gap + 1, make\_pair(-1, -1));

for (size\_t i = 0; i < nums.size(); i++)

{

int index = (nums[i] - minimum) / slot\_gap;

pair<int, int> pair = slot[index];

if (pair.first == -1)

{

pair.first = nums[i];

pair.second = nums[i];

}

else

{

if (nums[i] < pair.first)

{

pair.first = nums[i];

}

else if (nums[i] > pair.second)

{

pair.second = nums[i];

}

}

slot[index] = pair;

}

int max\_gap = INT\_MIN;

int previous = -1;

pair<int, int> pair;

if (slot.size() == 1)

{

pair = slot[0];

max\_gap = pair.second - pair.first;

}

else

{

for (size\_t i = 0; i < slot.size(); i++)

{

pair = slot[i];

if (pair.first == -1) continue;

if ((previous != -1) && (pair.first - previous) > max\_gap)

{

max\_gap = pair.first - previous;

}

previous = pair.second;

}

}

return max\_gap;

}

## [274. H-Index](https://leetcode.com/problems/h-index/)

Medium

Given an array of integers citations where citations[i] is the number of citations a researcher received for their ith paper, return the researcher's h-index.

According to the [definition of h-index on Wikipedia](https://en.wikipedia.org/wiki/H-index): The h-index is defined as the maximum value of h such that the given researcher has published at least h papers that have each been cited at least h times.

**Example 1:**

**Input:** citations = [3,0,6,1,5]

**Output:** 3

**Explanation:** [3,0,6,1,5] means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively.

Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, their h-index is 3.

**Example 2:**

**Input:** citations = [1,3,1]

**Output:** 1

**Constraints:**

* n == citations.length
* 1 <= n <= 5000
* 0 <= citations[i] <= 1000

[H-Index - LeetCode](https://leetcode.com/problems/h-index/description/)

/// <summary>

/// Leet Code 274. H-Index

///

/// Medium

///

/// Given an array of integers citations where citations[i] is the number

/// of citations a researcher received for their ith paper, return the

/// researcher's h-index.

///

/// According to the definition of h-index on Wikipedia: The h-index is

/// defined as the maximum value of h such that the given researcher has

/// published at least h papers that have each been cited at least h times.

///

/// Example 1:

/// Input: citations = [3,0,6,1,5]

/// Output: 3

/// Explanation: [3,0,6,1,5] means the researcher has 5 papers in total

/// and each of them had received 3, 0, 6, 1, 5 citations respectively.

/// Since the researcher has 3 papers with at least 3 citations each and

/// the remaining two with no more than 3 citations each, their

/// h-index is 3.

///

/// Example 2:

/// Input: citations = [1,3,1]

/// Output: 1

///

/// Constraints:

/// 1. n == citations.length

/// 2. 1 <= n <= 5000

/// 3. 0 <= citations[i] <= 1000

/// </summary>

int LeetCodeSort::hIndex(vector<int>& citations)

{

vector<int> count\_map(citations.size() + 1);

for (size\_t i = 0; i < citations.size(); i++)

{

if (citations[i] >= (int)citations.size())

{

count\_map[citations.size()]++;

}

else

{

count\_map[citations[i]]++;

}

}

int index;

for (index = citations.size(); index >= 0; index--)

{

if (index < (int)citations.size()) count\_map[index] += count\_map[index + 1];

if (count\_map[index] >= index) break;

}

return index;

}

# Dutch Flag Sort

Assume you have only 3 unique numbers with duplication in a long list, and you want to sort it. The solution is using 3 pointers, low, from left to right, keep track on the smallest number, high, from right to left, keep track on the largest number, mid is used to scan from left to right in case both end stuck on the middle number.

## [75. Sort Colors](https://leetcode.com/problems/sort-colors/)

Medium

Given an array nums with n objects colored red, white, or blue, sort them [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm)so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

**Example 1:**

**Input:** nums = [2,0,2,1,1,0]

**Output:** [0,0,1,1,2,2]

**Example 2:**

**Input:** nums = [2,0,1]

**Output:** [0,1,2]

**Constraints:**

* n == nums.length
* 1 <= n <= 300
* nums[i] is either 0, 1, or 2.

[Sort Colors - LeetCode](https://leetcode.com/problems/sort-colors/description/)

/// <summary>

/// Leet Code 75. Sort Colors

///

/// Medium

///

/// Given an array nums with n objects colored red, white, or blue, sort

/// them in-place so that objects of the same color are adjacent, with

/// the colors in the order red, white, and blue.

///

/// We will use the integers 0, 1, and 2 to represent the color red,

/// white, and blue, respectively.

///

/// You must solve this problem without using the library's sort function.

///

/// Example 1:

/// Input: nums = [2,0,2,1,1,0]

/// Output: [0,0,1,1,2,2]

///

/// Example 2:

/// Input: nums = [2,0,1]

/// Output: [0,1,2]

///

/// Constraints:

/// 1. n == nums.length

/// 2. 1 <= n <= 300

/// 3. nums[i] is either 0, 1, or 2.

/// </summary>

void LeetCodeSort::sortColors(vector<int>& nums)

{

int low, high, mid;

low = 0;

high = nums.size() - 1;

mid = low;

while (mid <= high)

{

if (nums[mid] == 0)

{

swap(nums[low], nums[mid]);

low++;

mid++;

}

else if (nums[mid] == 2)

{

swap(nums[mid], nums[high]);

high--;

}

else

{

mid++;

}

}

}

# Reconstruct Sort

## [406. Queue Reconstruction by Height](https://leetcode.com/problems/queue-reconstruction-by-height/)

[Queue Reconstruction by Height - LeetCode](https://leetcode.com/problems/queue-reconstruction-by-height/description/)

Medium

You are given an array of people, people, which are the attributes of some people in a queue (not necessarily in order). Each people[i] = [hi, ki] represents the ith person of height hi with **exactly** ki other people in front who have a height greater than or equal to hi.

Reconstruct and return the queue that is represented by the input array people. The returned queue should be formatted as an array queue, where queue[j] = [hj, kj] is the attributes of the jth person in the queue (queue[0] is the person at the front of the queue).

**Example 1:**

**Input:** people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]

**Output:** [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]

**Explanation:**

Person 0 has height 5 with no other people taller or the same height in front.

Person 1 has height 7 with no other people taller or the same height in front.

Person 2 has height 5 with two persons taller or the same height in front, which is person 0 and 1.

Person 3 has height 6 with one person taller or the same height in front, which is person 1.

Person 4 has height 4 with four people taller or the same height in front, which are people 0, 1, 2, and 3.

Person 5 has height 7 with one person taller or the same height in front, which is person 1.

Hence [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]] is the reconstructed queue.

**Example 2:**

**Input:** people = [[6,0],[5,0],[4,0],[3,2],[2,2],[1,4]]

**Output:** [[4,0],[5,0],[2,2],[3,2],[1,4],[6,0]]

**Constraints:**

* 1 <= people.length <= 2000
* 0 <= hi <= 106
* 0 <= ki < people.length
* It is guaranteed that the queue can be reconstructed.

/// <summary>

/// Leet Code 406. Queue Reconstruction by Height

///

/// Medium

///

/// You are given an array of people, people, which are the attributes of

/// some people in a queue (not necessarily in order). Each

/// people[i] = [hi, ki] represents the ith person of height hi with

/// exactly ki other people in front who have a height greater than or

/// equal to hi.

///

/// Reconstruct and return the queue that is represented by the input

/// array people. The returned queue should be formatted as an array

/// queue, where queue[j] = [hj, kj] is the attributes of the jth person

/// in the queue (queue[0] is the person at the front of the queue).

///

/// Example 1:

/// Input: people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]

/// Output: [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]

/// Explanation:

/// Person 0 has height 5 with no other people taller or the same height

/// in front.

/// Person 1 has height 7 with no other people taller or the same height

/// in front.

/// Person 2 has height 5 with two persons taller or the same height in

/// front, which is person 0 and 1.

/// Person 3 has height 6 with one person taller or the same height in

/// front, which is person 1.

/// Person 4 has height 4 with four people taller or the same height in

/// front, which are people 0, 1, 2, and 3.

/// Person 5 has height 7 with one person taller or the same height in

/// front, which is person 1.

/// Hence [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]] is the reconstructed queue.

///

/// Example 2:

/// Input: people = [[6,0],[5,0],[4,0],[3,2],[2,2],[1,4]]

/// Output: [[4,0],[5,0],[2,2],[3,2],[1,4],[6,0]]

///

/// Constraints:

/// 1. 1 <= people.length <= 2000

/// 2. 0 <= hi <= 10^6

/// 3. 0 <= ki < people.length

/// 4. It is guaranteed that the queue can be reconstructed.

/// </summary>

vector<vector<int>> LeetCodeSort::reconstructQueue(vector<vector<int>>& people)

{

vector<vector<int>> result;

struct heightCompare

{

bool operator() (vector<int>& a, vector<int>& b)

{

if ((a[0] > b[0]) || ((a[0] == b[0]) && (a[1] < b[1])))

{

return true;

}

else

{

return false;

}

}

};

sort(people.begin(), people.end(), heightCompare());

for (size\_t i = 0; i < people.size(); i++)

{

vector<vector<int>>::iterator itr = result.begin();

result.insert(itr + people[i][1], people[i]);

}

return result;

}

# Wiggle Sort

## [280. Wiggle Sort](https://leetcode.com/problems/wiggle-sort/)

Medium

Given an integer array nums, reorder it such that nums[0] <= nums[1] >= nums[2] <= nums[3]....

You may assume the input array always has a valid answer.

**Example 1:**

**Input:** nums = [3,5,2,1,6,4]

**Output:** [3,5,1,6,2,4]

**Explanation:** [1,6,2,5,3,4] is also accepted.

**Example 2:**

**Input:** nums = [6,6,5,6,3,8]

**Output:** [6,6,5,6,3,8]

**Constraints:**

* 1 <= nums.length <= 5 \* 104
* 0 <= nums[i] <= 104
* It is guaranteed that there will be an answer for the given input nums.

**Follow up:** Could you solve the problem in O(n) time complexity?

/// <summary>

/// Leet Code 280. Wiggle Sort

///

/// Medium

///

/// Given an integer array nums, reorder it such that

/// nums[0] <= nums[1] >= nums[2] <= nums[3]

///

/// You may assume the input array always has a valid answer.

/// Example 1:

/// Input: nums = [3,5,2,1,6,4]

/// Output: [3,5,1,6,2,4]

/// Explanation: [1,6,2,5,3,4] is also accepted.

///

/// Example 2:

/// Input: nums = [6,6,5,6,3,8]

/// Output: [6,6,5,6,3,8]

///

/// Constraints:

/// 1. 1 <= nums.length <= 5 \* 10^4

/// 2. 0 <= nums[i] <= 10^4

/// 3. It is guaranteed that there will be an answer for the given input

/// nums.

/// </summary>

void LeetCodeSort::wiggleSort(vector<int>& nums)

{

for (size\_t i = 1; i < nums.size(); i++)

{

if (i % 2 == 1)

{

if (nums[i] < nums[i - 1]) swap(nums[i], nums[i - 1]);

}

else

{

if (nums[i] > nums[i - 1]) swap(nums[i], nums[i - 1]);

}

}

}