

**Stack:**

- LIFO
- Insert at top and delete at top

Insert at top	<code>push()</code>
Remove from top	<code>pop()</code>
Return number of elements in stack	<code>size()</code>
Check empty	<code>empty()</code>
Returns a reference to the newest element in the stack	<code>top()</code>
STL	<code>#include &lt;stack&gt;</code>

**Queue:**

- FIFO

Insert at back	<code>push()</code>
Remove from front	<code>pop()</code>
Return the first element	<code>front()</code>
Return the last element	<code>back()</code>
Return number of elements in queue	<code>size()</code>
Check empty	<code>empty()</code>
Exchange two values in queue with same data types	<code>swap()</code>
STL	<code>#include &lt;queue&gt;</code>

**String:**

- the last character is '\0'

Remove the last element from a vector	<code>pop_back()</code>
Append each character to a string object	<code>push_back()</code>
Make the vector empty	<code>clear()</code>
Check whether it is empty	<code>empty()</code>
Erase character from string	<code>erase()</code>
Swap string values	<code>swap()</code>
Find the content in string	<code>find()</code>
Return the max size of string	<code>max_size()</code>
Get the character with a given position in string	<code>at()</code>
Assign content to string	<code>assign()</code>
STL	<code>#include &lt;string&gt;</code>

## Dynamic Programming:

- create a new array/table
- Initialized by filling with 0, with a specific size n, m
- Store the calculated value, recalculated is not needed anymore, extract at the corresponding index is the only need

## Array:

- can be referenced by index, starting from 0
- Initialized with a given size is a must, but a initialized value is not an obligation(it may randomly filled with garbage value)
- Notation: arrayName[element]
- sizeof()

**70. Climbing Stairs**

Description: You are climbing a staircase. It takes  $n$  steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

**1047. Remove All Adjacent Duplicates In String**

Description: You are given a string  $s$  consisting of lowercase English letters. A **duplicate removal** consists of choosing two adjacent and equal letters and removing them.

We repeatedly make **duplicate removals** on  $s$  until we no longer can.

Return the **final string after all such duplicate removals have been made**. It can be proven that the answer is **unique**.

Example 1:

```
Input: s = "abbaca"
Output: "ca"
Explanation:
For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".
```

**121. Best Time to Buy and Sell Stock**

Description: You are given an array  $\text{prices}$  where  $\text{prices}[i]$  is the price of a given stock on the  $i^{\text{th}}$  day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return the **maximum profit** you can achieve from this transaction. If you cannot achieve any profit, return  $0$ .

**Example 1:**

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

**Example 2:**

**Solution Code:**

```
public:
    int maxProfit(vector<int>& prices) {
        /*
        //greedy search: find the minimum price from left side as buy price; find the maximum price from right side as sell price;
        Make difference;
        //initialization
        int lowest=INT_MAX;
        int profit=0;
        for(int i=0; i<prices.size(); i++){ //set a pointer
            lowest=min(lowest, prices[i]);
            profit=max(profit, prices[i]-lowest);
        }
        return profit;*/
        //Run Time Error...
        //suppose dp[i][0]:ith day holding stock's maximum profit, dp[i][1]: ith day not holding stock corresponding maximum profit
        int len=prices.size();
        if(len==0) return 0; //special case
        vector<vector<int>> dp(len, vector<int>(2)); // create a len*2 matrix
        dp[0][0]=-prices[0]; //买入股票
        dp[0][1]=0;
        for(int i=1; i<len; i++){
            dp[i][0]=max(dp[i-1][0], -prices[i]);
            dp[i][1]=max(dp[i-1][1], prices[i]+dp[i-1][0]);
        }
        return dp[len-1][1];
        //Only the status of ith day and i-1day are enough
        int len=prices.size();
        vector<vector<int>> dp(2, vector<int>(2)); //create a 2*2 matrix
        dp[0][0]=-prices[0]; //买入股票
        dp[0][1]=0;
        for(int i=1; i<len; i++){
            dp[i%2][0]=max(dp[(i-1)%2][0], -prices[i]);
            dp[i%2][1]=max(dp[(i-1)%2][1], prices[i]+dp[(i-1)%2][0]);
        }
        return dp[(len-1)%2][1];
    }
}
```

### 3. Document Chunking

A financial services company is uploading documents to a compliance system for analysis. They use a chunking mechanism as below.

- Each document is divided into equal sized packets.
- Documents are then divided and uploaded in "chunks" of packets. A chunk is defined as a contiguous collection of  $2^n$  packets, where  $n$  is any integer  $\geq 0$ .
- After the document is divided into chunks, randomly selected chunks are uploaded until the entire document is completely uploaded.

There is one document that is partially uploaded, described in *uploadedChunks*. Determine the minimum number of chunks that are yet to be uploaded.

#### Example

```
totalPackets = 5
n = 2 number of chunks already uploaded
uploadedChunks = [[1, 2]]
```

- The document has 5 packets and 1 chunk of  $2^1 = 2$  packets [1, 2] is already uploaded.
- The remaining 3 packets are uploaded in 2 chunks. The length of chunk1 is  $2^1 = 2$ , packets [3, 4], and the length of chunk2 is  $2^0 = 1$  packet [5].



#### Function Description

Complete the *minimumChunksRequired* function in the editor below.

*minimumChunksRequired* has the following parameter(s):

- long int *totalPackets*: the number of packets in the document.
- long int *uploadedChunks*[n][2]: each *uploadedChunks*[i] describes the start and end packet numbers of the uploaded chunks.

#### Returns

- Int: an integer that denotes the minimum number of chunks that need to be uploaded.

#### Constraints

- $1 \leq totalPackets < 10^8$
- $0 \leq uploaded < 10^5$
- The uploaded chunks do not overlap
- $1 \leq starting\ packet \leq ending\ packet \leq totalPackets$

### 2. K Smallest Substring

There is a string *input\_str* consisting of characters '0' and '1' only and an integer *k*. Find a substring of string *input\_str* such that:

- The number of '1's is equal to *k*
- It has the smallest length
- It is lexicographically smallest

Note: It is guaranteed that answer always exists.

#### Example

```
input_str = "0101101"
k = 3
```

Some of the possible substrings following the first condition:

- "01011"
- "1101"
- "1011"

The substring that is smallest in length and lexicographically smallest is "1011".

It can be proven that there is no other substring that is smaller than "1011" in length and lexicographic order. Hence the answer is "1011".

#### Function Description

Complete the function *getSubstring* in the editor below.

*getSubstring* has the following parameters:

- string *input\_str*: a string that consists of '0' and '1'
- int *k*: the number of '1's in the answer

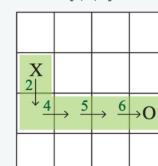
#### Returns

### 1. Laser Grid

A player stands on a cell within a grid. The player can move to one of four adjacent cells, but the motion is constrained by lasers. To move from one position to another involves a cost: the cost to move from row *i* to row *i* ± 1 is *costRows*[*j*] and the cost to move from column *j* to column *j* ± 1 is *costCols*[*j*]. Find the minimum cost to move from a starting point to an ending point within the grid.

#### Example

```
rows = 4
cols = 4
initR = 1
initC = 0
finalR = 2
finalC = 3
costRows = [1, 2, 3]
costCols = [4, 5, 6]
```



The player starts at position (*row*, *col*) = (1, 0) (marked with an X) and must reach position (2, 3) (marked with an O).

- To move from row 1 from row 2, the cost is *costRows*[1] = 2.

- To move from column 0 to column 3, the player must move to column 1, then 2, then 3 at a cost of *costCols*[0] + *costCols*[1] + *costCols*[2] = 4 + 5 + 6 = 15.

- The total cost is 2 + 15 = 17.

### 1. Match Outcomes

Given the initial setup of a match between two players, evaluate the match's outcome.

There are two players, and there is a number sequence of size *n*. Players alternate turns for *n* rounds. Each round, a player removes the first number from the sequence and adds its value to their score. After that, if the 'removed' number is even, the remaining sequence is reversed. Determine the difference in scores between the two players after the game.

More precisely, suppose *first\_score* and *second\_score* are the final scores of the first and second player, respectively. The goal is to calculate the value of *first\_score* - *second\_score*.

#### Example

The number of elements is *n* = 5 and *numSeq* = [3, 6, 2, 3, 5].

- 1<sup>st</sup> round: The first player picks 3, *first\_score* = 3. The remaining sequence: [6, 2, 3, 5].
- 2<sup>nd</sup> round: The second player picks 6, *second\_score* = 6. Since 6 is even, the remaining sequence is reversed: [5, 3, 2].
- 3<sup>rd</sup> round: The first player picks 5, *first\_score* = 3 + 5 = 8. The remaining sequence: [3, 2].
- 4<sup>th</sup> round: Second player picks 3, *second\_score* = 6 + 3 = 9. The remaining sequence: [2].
- 5<sup>th</sup> round (final): First player picks 2, *second\_score* = 8 + 2 = 10. The remaining sequence: [].

The total difference between players' scores is *first\_score* - *second\_score* = 10 - 9 = 1.

#### Function Description

Complete the function *getScoreDifference* in the editor.

*getScoreDifference* has the following parameter:

- *numSeq*: the given array of integers

#### Returns

- *int: first\_score - second\_score*

## 2. Alternating Prime Sequence

A student needs to construct an *alternating prime sequence* from an array of integers for a coding challenge. If an integer cannot be used, its value is added to a penalty accumulator.

An alternating prime sequence is a sequence where adjacent elements alternate between prime and non-prime. For example, [1, 2, 4, 3, 6, 5] is a valid alternating prime sequence while [1, 2, 3, 4, 5, 6] is not as 2 and 3 are both primes and they are adjacent. Find the minimum possible penalty that the student can have after completing the task.

### Example

Consider, for example, a sequence  $arr = [3, 7, 1, 4, 6, 6]$  of length  $n = 6$ . The student can construct a sequence [4, 3, 6, 7, 6] which has a penalty of 1. Note that there are other possible sequences as well, but 1 is the minimum penalty.

### Function Description

Complete the function `minPenalty` in the editor below.

`minPenalty` has the following parameter(s):

`int arr[n]:` an array of integers

### Returns

`int:` the minimum penalty value

### Constraints

- $1 \leq n \leq 10^4$
- $1 \leq arr[i] \leq 10^4$



Problem

My Submissions

All Submissions

Discussion

## Maximum MEX :



Author

Shikhar Mehrotra

Difficulty Level : Medium

Marks :20

Submissions : 68

15 : 3 | 18 : 0

Asked In : Atlassian

Given an array containing  $N$  non negative integers and an element  $X$ , In one operation,  $X$  can be added to or subtracted from any element of the array. MEX of an array is defined as the **smallest non negative integer which is not present in the array**.

for example, the MEX of [0,1,1,3] is 2, and the MEX of [1,2,4] is 0.

Find the maximum possible MEX of the array that can be achieved by doing the above operation any number of times.

### Input



The first line contains a single integer  $1 \leq N \leq 10^9$  denoting size of the array.

The second line contains  $N$  space separated integer denoting elements of the array  $0 \leq a[i] \leq 10^9$  where  $(0 \leq i < n)$ .

Third line contains a single integer  $X$  where  $(1 \leq X \leq 10^5)$

### Output

Output a single integer denoting the maximum MEX possible by applying operations any number of times.

### Examples

#### input

3  
1 3 4  
2

#### output

2

#### input

5  
0 1 2 1 3

### 3. Statistic Indicators

Given an array, find two statistic indicators for the array and report the difference between the two stats. The indicators are defined as follows:

- *Indicator 1:* This is defined by the number of instances where a number  $k$  appears for **exactly**  $k$  consecutive times in the array. For example, in the array [1, 2, 2, 2, 2, 3, 3, 3, 1, 1, 2, 2], we have 1(1), 2(4), 3(3), 1(2), 2(2) where the value in the braces represents the number of times the integer appeared consecutively. The value of indicator 1 for the given array would be 3 corresponding to 1(1), 3(3) and 2(2).
- *Indicator 2:* This is defined by the number of instances where a number  $k$  appears for **exactly**  $k$  consecutive times in the array, starting from index  $k$  assuming 1-based indexing. For example, in the array [2, 2, 2, 4, 4, 4, 4, 4, 4], if we start at index 2 we have exactly 2 consecutive 2s coming up. While when we start at index 4, we have 6 consecutive 4s coming up. Hence the value of indicator 2 for the array would be 1 corresponding to the index 2.

Your task is to find the absolute difference between the two indicators.



Description Editorial Solutions (895) Submissions

**393. UTF-8 Validation**

Hint ⓘ

Medium ⓘ 870 2.8K ⓘ

Companies ⓘ

Given an integer array `data` representing the data, return whether it is a valid **UTF-8** encoding (i.e. it translates to a sequence of valid UTF-8 encoded characters).

A character in **UTF8** can be from **1 to 4 bytes** long, subjected to the following rules:

1. For a **1-byte** character, the first bit is a `0`, followed by its Unicode code.
2. For an **n-bytes** character, the first `n` bits are all one's, the `n + 1` bit is `0`, followed by `n - 1` bytes with the most significant `2` bits being `10`.

This is how the UTF-8 encoding would work:

Number of Bytes	UTF-8 Octet Sequence (binary)
1	0xxxxxxxx
2	110xxxxx 10xxxxxx
3	1110xxxx 10xxxxxx 10xxxxxx
4	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

x denotes a bit in the binary form of a byte that may be either 0 or 1.

i C++ ⓘ Auto

```
1 class Solution {
2 public:
3     bool validUtf8(vector<int>& data) {
4         int len=data.size();
5         for(int i=0; i<len; i++){
6             if(data[i]<0b10000000){
7                 continue;
8             }else{
9                 int count=0;
10                int index=data[i];
11                for(int j=7; j>=1; j--){
12                    if(index>=pow(2,j)) ++count;
13                    else break;
14                    index-=pow(2,j);
15                }
16                if(count==1 || count>4 || count>len-i) return false;
17                for(int j=i+1; j<i+count; ++j){
18                    if(data[j]>0b10111111 || data[j]<0b10000000) return false;
19                }
20                i+=count-1;
21            }
22        }
23    }
24    return true;
25 }
26 }
```

Console ⓘ Run Submit

Leetcode 1319:

DFS:

Search in a specific direction continuously, if it cannot search any more, then change another direction.

While, BFS is search all connected nodes of current node for each node.

Application:

1. Detect cycles in a graph
2. Find the strongly connected components of a graph
3. Find a path

Solution:

1. Construct a visited array(filled with T/F value) and stack
2. Store the adjacent vertex in stack first
3. Store root node in visited array first
4. Once the element is actually called, set the value in array for True, remove the value from stack to visited array; iterated from first left unvisited vertex

- Notes:

- Range-based for loop: for (range\_declaration : range\_expression)
  - range\_declaration: a named variable
  - range\_expression: a suitable sequence or a braced-init-list
  - Reference: <https://www.geeksforgeeks.org/range-based-loop-c/>
- The difference between `++count;` and `count++`:
  - `++count`: pre-increment. Increase the value before executing the statement; value is first incremented and then used inside the expression.
  - `count++`: post-increment. Increase the value after executing this statement; value is first used in an expression and then incremented.

The screenshot shows the Leetcode problem page for "1319. Number of Operations to Make Network Connected". The problem details are as follows:

**Description**: There are  $n$  computers numbered from 0 to  $n - 1$  connected by ethernet cables forming a network where  $\text{connections}[i] = [a_i, b_i]$  represents a connection between computers  $a_i$  and  $b_i$ . Any computer can reach any other computer directly or indirectly through the network.

**Medium**, **Companies**: Companies

**Constraints**:

- $1 \leq n \leq 500$
- $0 \leq a_i, b_i < n$
- $a_i \neq b_i$
- $(a_i, b_i)$  are distinct pairs.

**Example 1:** Input:  $n = 4$ ,  $\text{connections} = [[0, 1], [0, 2], [1, 2]]$ . Output: 1. Explanation: We only need to connect the computer 3 to the network to make it fully connected with minimum number of operations.

**Code Editor (C++)**:

```
1 class Solution {
2 public:
3     int makeConnected(int n, vector<vector<int>>& connections) {
4         //basic cases:
5         //目前所有电脑已经连在一起了，则不需要移动任何网线
6         //链接n台电脑，至少需要n-1根网线
7         //若无法连接说有电脑，则返回-1 invalid value
8         //所以如果< n-1根网线，则返回-1.
9         //什么情况下需要移动网线?
10        //存在孤立电脑 或 不同的群组
11        //需要移动网线的个数: 有多少个彼此孤立的群组
12        if(connections.size()<n-1) return -1;
13        int count=0; //represent numbers of groups
14        vector<vector<int>> g(n); //construct a stack represent all connected nodes
15        with i
16            vector<bool> visited(n); //construct visited array for storing visited value
17            with True/False
18                for(auto &c: connections){ //range-based for loop
19                    g[c[0]].push_back(c[1]);
20                    g[c[1]].push_back(c[0]);
21                }
22                for(int i=0; i<n; i++){
23                    if(visited[i]) continue;
24                    ++count; //若未被遍历, group的数目+1
25                    dfs(g, i, visited);
26                }
27            return count-1;
28        void dfs(vector<vector<int>>& g, int i, vector<bool>& visited){
29            visited[i]=true;
30            for(int next : g[i]){
31                if(visited[next]) continue;
32                dfs(g, next, visited);
33            }
34        };
}
```

At the bottom of the code editor, there are buttons for "Console", "Run", and "Submit". Below the code editor, a message says "Continue to work on your code from Aug 01, 2023 20:27:13" and "Restore".

## Similar Problem: LeetCode200

Description Editorial Solutions (8.2K) Submissions

### 200. Number of Islands

Medium 20.7K 449 Companies

Given an  $m \times n$  2D binary grid `grid` which represents a map of '1' s (land) and '0' s (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

```
Input: grid = [
    ["1","1","1","1","0"],
    ["1","1","0","1","0"],
    ["1","1","0","0","0"],
    ["0","0","0","0","0"]
]
Output: 1
```

**Example 2:**

```
Input: grid = [
```

Console ^ Run Submit

```
i C++ | Auto
1 class Solution {
2 public:
3     int numIslands(vector<vector<char>>& grid) { //grid means a graph
4         //遍历到1的时候是岛屿，遍历到0的时候是水
5         //使用visited array记录所遍历的岛屿，遇到遍历过的岛屿就跳过
6         //DFS
7         if(grid.size() == 0 || grid[0].size() == 0)
8             return 0;
9         //get width and length of 2D binary grid
10        int m=grid.size();
11        int n=grid[0].size();
12        //计算岛屿数量
13        int count=0;
14        //遍历
15        for(int i=0; i<m; i++){
16            for(int j=0; i<n; j++){
17                if(grid[i][j]=='1'){
18                    count++;
19                    dfs(grid, i, j);
20                }
21            }
22        }
23        return count;
24    }
25 private:
26     void dfs(vector<vector<char>>& grid, int i, int j){
27         if(i<0 || j<0 || i>=m || j>=n || grid[i][j]=='0')
28             return;
29         //将当前位置陆地设置为0，深度优先搜索去遍历其他的陆地，将找到的陆地全设为零
30         grid[i][j]='0';
31         dfs(grid, i-1, j);
32         dfs(grid, i, j-1);
33         dfs(grid, i+1, j);
34         dfs(grid, i, j+1);
35     }
36     int m;
37     int n;
38 };
```

## LeetCode561: Array Partition

Description Editorial Solutions (2.5K) Submissions

### 561. Array Partition

Easy 1.8K 226 Companies

Given an integer array `nums` of  $2n$  integers, group these integers into  $n$  pairs  $(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$  such that the sum of  $\min(a_i, b_i)$  for all  $i$  is maximized. Return the maximized sum.

**Example 1:**

```
Input: nums = [1,4,3,2]
Output: 4
Explanation: All possible pairings
(ignoring the ordering of elements) are:
1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3)
= 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4)
= 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4)
= 1 + 3 = 4
So the maximum possible sum is 4.
```

**Example 2:**

```
Input: nums = [1,4,3,2]
Output: 4
Explanation: All possible pairings
(ignoring the ordering of elements) are:
1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3)
= 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4)
= 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4)
= 1 + 3 = 4
So the maximum possible sum is 4.
```

Console ^ Run Submit

```
i C++ | Auto
1 class Solution {
2 public:
3     int arrayPairSum(vector<int>& nums) {
4         //思路:
5         //较大的数字聚到一起，min的组合才会更大
6         //sorting
7         //将index从到2n-1的偶数位元素相加
8         int sum=0;
9         sort(nums.begin(), nums.end());
10        for(int i=0; i<nums.size(); i=i+2){ //注意偶数位
11            sum += nums[i];
12        }
13        return sum;
14    }
15 };
```

## LeetCode621: Task Scheduler

Solution: Greedy Search:

- local optima—>whole optima
- 拆分成子问题
- 找出贪心策略
- 找局部最优
- 堆叠局部最优，得到全局最优

Hash Map:

- key-value pair
- Delete, get, put, DeleteHashTable
- STL: #include <unordered\_map>
- Notations: unordered\_map<keyDataType, valueDataType> mapName;

Binary Search Tree:

- each node's children <=2
- Value of all nodes of left tree < value of root node
- Value of all nodes of right tree > value of root node

Priority Queue:

- with a priority value, higher priority, the corresponding element served first;
- If elements have same priority number, they are served to their order in queue.
- Not FIFO, but removed based on priority. Highest priority elements removed first.
- Can be implemented by array, binary search tree, linked listheap. While, heap provides most efficient implement.
- When inserting, deleting elements of PQ, we must obey the value property of BST.
- Min heap and Max heap:
  - If it is a min heap, the top of tree is minimum value. We only need to get the peak of the tree is enough. <https://www.programiz.com/dsa/heap-sort#heap>
  - If the element of the root is not larger/smaller than its children, use .swap() function to realize.
  - Convert the vector into max\_heap using make\_heap(iter\_first, iter\_last) in STL under #include<algorithm>

LeetCode703

- Template of priority queue: priority\_queue<Type, Container, Functional>
  - Functional: way of element comparison
  - Container: to store data
- .emplace(value): The parameter is added to the priority queue at the top position.
  - The difference between emplace and push: emplace saves unnecessary copy of object

## 703. Kth Largest Element in a Stream

Easy 4.9K 3K

Companies

Design a class to find the  $k^{th}$  largest element in a stream. Note that it is the  $k^{th}$  largest element in the sorted order, not the  $k^{th}$  distinct element.

Implement `KthLargest` class:

- `KthLargest(int k, int[] nums)` Initializes the object with the integer `k` and the stream of integers `nums`.
- `int add(int val)` Appends the integer `val` to the stream and returns the element representing the  $k^{th}$  largest element in the stream.

```
1 class KthLargest { //goal: 求最大K个元素中的最小值
2 private:
3     priority_queue<int, vector<int>, greater<int>> pq; //min heap
4     int k;
5 public:
6     KthLargest(int k, vector<int>& nums) {
7         this->k=k;
8         for(auto& num: nums){
9             pq.emplace(num);
10            if(pq.size()>k) pq.pop();
11        }
12    }
13
14    int add(int val) {
15        pq.emplace(val); // add new element
16        if(pq.size()>k) pq.pop(); //check the length of pq
17        return pq.top(); //get the Kth largest value
18    }
19 };
20
21 /**
22 * Your KthLargest object will be instantiated and called as such:
23 * KthLargest* obj = new KthLargest(k, nums);
24 * int param_1 = obj->add(val);
25 */
```

SQL:  
SELECT  
FROM  
WHERE  
GROUP BY  
HAVING  
ORDER BY  
LIMIT

window functions:

- RANK, DENSE\_RANK, ROW\_NUMBER
- LAG, LEAD
- CASE
  - WHEN...THEN...
  - WHEN...THEN...
  - ELSE...
  - END

join & union

- (LEFT/RIGHT/SELF/INNER/FULL) JOIN...ON

OR

AND

IN

COUNT DISTINCT

LOWER/UPPER

IS NULL

NULLIF

IFNULL

ROUND

AVG

SUM

MIN

MAX

Key Points:

- General linear model VS Generalized linear model
- Frequency VS bayesian
- 异方差
- GARCH, how to calculate volatility by GARCHs
- Local volatility & Implied volatility
- Hash function and hash map
- Variance swap; Swaption 套利策略, volatility swap
- Volatility surface计算, 以及参数
- 负利率, bond with negative interest
- Hull-White Model
- Future VS Forwards
- Barrier Option pricing
- Decision tree VS Random forest, Entropy?

- Greeks and option pricing
- When Vega comes to maximum value
- Point process—>passion distribution
- GBM
- Martingale
- Ito Formula
- 多重共线性怎么解决
- SQL: CTE
- ETL Warehouses Pipeline, multi-threadings
- QuickSort
- Linear regression assumption
- Pandas DataFrame, sort, lambda, join