

379. Design Phone Directory

Design a Phone Directory which supports the following operations:

get: Provide a number which is not assigned to anyone.

check: Check if a number is available or not.

release: Recycle or release a number.

Stack<int> released

boolean[] used

int index(index $\in [0, \text{maxnumber} - 1]$)

get():

if (!released.isEmpty()):

phone · number = released.pop()

used[phone · number] = used[phone · number] + 1

return phone · number

else:

if (index < maxnumber):

used[index] = true

index = index + 1

return index - 1

else: return - 1

release(int phone · number):

if (used[phone · number]): release.add(phone · number)

used[phone · number] = false

check(int phone · number): return !used[phone · number]

355. Design Twitter

map<Integer, Set<Integer>> fans

map<Integer, Tweet> tweets

Tweet $\left\{ \begin{array}{l} \text{tweet}_{id} \\ \text{timestamp} \\ \text{next} \end{array} \right.$

quite simialr to merge k sorted list(in this case, sorted by timestamp)

```

friendlist  $\leftarrow$  fans[userid]
result  $\leftarrow$  new list(Tweet)
for(friend  $f$  : friendlist):
    Tweet tw  $\leftarrow$  tweets[friendid].peekFirst()
    if(tw is not nil): pq.offer(tw)
count  $\leftarrow$  0
while(!pq.empty & count < 10):
    Tweet top = pq.poll()
    result.add(top)
    if(top.next  $\neq$  null):pq.offer(top.next)
    count  $\leftarrow$  count + 1

```

146. LRU Cache Facebook Uber

$$\text{Node} \left\{ \begin{array}{l} \text{key} \\ \text{val} \\ \text{prev} \\ \text{next} \end{array} \right. \rightarrow \text{dll} \left\{ \begin{array}{l} \text{head} \\ \text{tail} \end{array} \right\}$$

```

int get(K):
    if(keyNode[K]  $\neq$  nil):
        node := keyNode[K]
        V := node.val
        dll.unlink(node)
        dll.addHead(node)
        return V
    return - 1

void set(K, V):
    if(keyNode[K]  $\neq$  nil):
        node := keyNode[K], node.val := V
        dll.unlink(node), dll.addHead(node)
    else:
        if(keyNode.size < cap):

```

```

    new := (K, V)
    dll.addHead(new)
    keyNode[key] := new
else:
    lru := dll.tai.prev
    dll.unlink(lru)
    keyNode[lru.key] = nil
    new := (K, V)
    dll.addHead(new)
    keyNode[key] := new

```

Follow Up: Distributed LRU Uber

We can maintain a hash-table that maps resource (it could be key of KV pair) to corresponding machine. Then, when a request comes in for $source_A$, it will be directed to its corresponding machine, say $machine_M$. At the $machine_M$, it does the same thing as the single LRU cache strategy.

Consistent Hashing

$shardNum = n(2^{64} - 1) \rightarrow$ means there are $(2^{64} - 1)$ virtual node (vnode)

$microShardNum = 1k$

$shard2MachineId = \{\phi\}$

void addMachine(int machine_{id}):

count = 0

Set<int> group = { ϕ }

while(count < microShardNum):

rid = rand.nextInt(shardNum)

if(shard2MachineId[rid] == null):

shard2MachineId[rid] = machine_{id}

count = count + 1

group.add(rid)

```

for(int id: group):
    copy partial source from vnode[(id + 1)/size] to vnode[id]

```

```

int getMachineByHash(int hashId):
    i = hashId
    while(true):
        if(shard2MachineId[i] ≠ null):return shard2MachineId[i]
        i = (i + 1) mod shardNum

```

```

void put(K source, V value):
    hid ← hashCode(K) mod shardNum
    machineid ← getMachineByHash(hid)
    update(machineid, source, value)

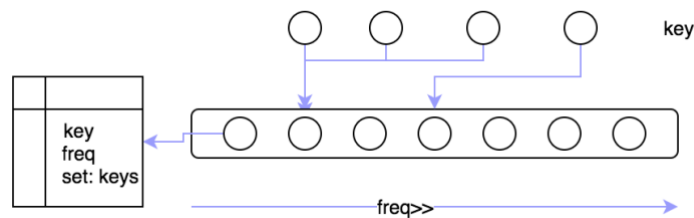
```

```

V request(K resource):
    hid ← hashCode(K) mod shardNum
    machineid ← getMachineByHash(hid)
    request(machineid, K)

```

460. LFU Cache Google Amazon



```

class Node:
    Node pre, next
    int key, freq
    keys: LinkedHashSet

class DoubleLinkedList:
    Node head, tail
    void addAfter(Node node, Node newNode):
    void unlink(Node node):

```

```

map <int, int> key2value
map <int, Node> key2node
DoubleLinkedList dll

```

```

void get(int key):
    if (key2value[key] ≠ null):
        increase(key)
        retrun key2value[key]
    return - 1

```

```

void set(int key, int value):
    if (key2value[key] ≠ null):
        key2value[key] ← value
        increase(key)
    else:
        if (map.size > cap): removeOld()
        key2value.put(key, value)
        increase(key)

```

```

void increase(int key):
    Node nd ← key2node[key]
    if (nd ≠ nil):
        nd.keys.remove(key)
        if (nd.next == dll.tail or nd.next.freq ≠ nd.freq + 1):
            Node nx ← Node(key, nd.freq + 1)
            dll.addAfter(nd, nx)
            key2node.put(key, nx)
        elif (nd.freq + 1 == nd.next.freq)
            nd.next.keys.add(key)
            key2node.put(key, nd.next)
        if (nd.keys.size == 0): remove(nd)
    else if (dll.head.next == dll.tail or dll.head.next.freq ≠ 1):
        Node nd ← Node(key = key, freq = 1)
        nd.keys.add(key)
        dll.addAfter(dll.head, nd)

```

```

    key2node.put(key, nd)
else if (dll.head.next.freq == 1):
    dll.head.next.keys.add(key)
    key2node.put(key, dll.head.next)

```

```

void removeOld(): ★ evict one of key in head node(least freq)
    if (head is null): return
    int tobeRemoved := nil
    for (int k: head.keys):
        tobeRemoved ← k, break
    key2value.remove(tobeRemoved)
    key2node.remove(tobeRemoved)
    if (head.keys.size == 0): remove(head)

```

432. All O'one Data Structure Uber

```

    inc(x),
    dec(x),
    getMaxkey(),
    getMinkey()

```

Intuition & Algorithm

$$Node \left\{ \begin{array}{l} freq \\ Set[string] \text{ keys} \\ prev \\ next \end{array} \right. \rightarrow dll \left\{ \begin{array}{l} head \\ tail \end{array} \right.$$

380. Insert & Delete Get-Random O(1)

381. Insert & Delete Get-Random O(1) Uber

Solution I: ArrayList + hashmap

```

list<int> list
map<int, int> val2index
insert(int val): list.add(val)

```

delete(int val):

swap(list, list.size - 1, val2index[val]), list.remove(list.size - 1)

getrandom(void): return list[rand.nextInt(list.size)]

Solution II: two hashmap: val \rightarrow index, index \rightarrow val, count = 0

boolean insert(int val):

if(val \rightarrow index[val] is null): return false

val \rightarrow index[val] \leftarrow count

index \rightarrow val[count] \leftarrow val

count \leftarrow val \rightarrow index.size

boolean delete(int val):

if(val \rightarrow index[val] is null): return false

idx \leftarrow val \rightarrow index[val]

val \rightarrow index[val] \leftarrow null

if(idx \neq count - 1):

lastvalue = index \rightarrow val[count - 1]

index \rightarrow val[idx] \leftarrow lastvalue

val \rightarrow index[lastvalue] \leftarrow idx

else: index \rightarrow val[idx] \leftarrow null

count \leftarrow val \rightarrow index.size

Design Rate Limiter Google Uber

For example, no more than 100 requests per second.

for(t in 0 \rightarrow 59):

key = event(url_{shorten}) + feature(ip) + |current ts - t|

** event \in {url_{request}, login, logout, register}*

sum += memcached.get(key, default = 0)

check sum is in limitation

interface RateLimit {

void setQPS(int qps);

boolean allowThisRequest()

```
}
```

```
public class RateLimitImpl implements RateLimit {
    private long qpsms
    private Map<ip, Queue<int> > map
    public void setQPS(int qps):
        if(qps < 1 or qps > 1million): throw new RuntimeException()
        qpsms = qps * 1k
        map ← new HashMap<>
    public boolean allowThisRequest(Request req):
        boolean allowed ← false
        Q = map.getOrDefault(req.ip, new Queue<>())
        if(Q.isEmpty()):
            Q.offer(req)
        elif(Q.size() < qpsms || req.ts - Q.peek() > 1000ms):
            allowed = true
            while(Q.size() > 0 & req.ts - Q.peek() > 1000ms): Q.poll()
        else: allowed = false
    return allowed
}
```

For example



Case I: $qps = 5$, then

$req(0.8s): ac(Q = \{0.8s\})$

if a request comes at 0.81s, it seems that the local $qps = \frac{2}{0.81-0.8} \gg 5(qps)$, but we should allow this request, why? because we only care how many requests comes in a window of 1 second, we don't need to care how request distributed in the 1s window. So a sequence of request of $req(0.8s, 0.81s, 0.82s, 0.83s, 0.84s)$ should be allowed under the limitation of 5 qps.

$req(1.2s): ac(Q = \{0.8s, 1.2s\})$

$req(1.3s): ac(Q = \{0.8s, 1.2s, 1.3s\})$


```

req(1.5s): ac (Q = {0.8s, 1.2s, 1.3s, 1.5s})
req(1.7s): ac (Q = {0.8s, 1.2s, 1.3s, 1.5s, 1.7s})
req(1.9s): ac (Q = {0.8s, 1.2s, 1.3s, 1.5s, 1.7s, 1.9s})
req(2.1s): deny (Q = {0.8s, 1.2s, 1.3s, 1.5s, 1.7s, 1.9s} 2.1 - 1.2 < 1s)
req(2.5s): ac (Q = {1.2s, 1.3s, 1.5s, 1.7s, 1.9s, 2.5s} 2.5 - 1.2 > 1s)

```

359. Logger Rate Limiter Uber

```

boolean shouldPrintMessage(String msg, int ts):
    if (map[msg] == null || ts - map[msg] ≥ 10s):
        map[msg] = ts
        return true
    return false

```

636. Exclusive Time of Functions Facebook Uber

Solution: "start" → push & "end" → pop

```

int[] result = new int[n];
Stack<int> stack = new stack <> ();
int prev = 0;
for (String log: logs):
    String[] token = log.split(":");
    int funcId = Integer.parseInt(token[0]);
    boolean start = token[1].equals("start");
    int ts = Integer.parseInt(token[2]);
    if (start):
        //calculate outer function time cost
        if (!stack.empty): re[stack.peek] += ts - prev;
        stack.push(funcId);
        prev = ts;
    else: //calculate inner function time cost
        if (!stack.empty): re[stack.pop] += ts - prev + 1
        prev = ts + 1;
return result;

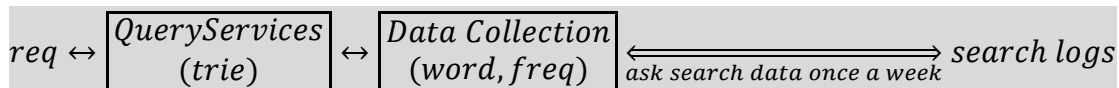
```

Design PokemonGo. **Uber**

<http://www.1point3acres.com/bbs/thread-287893-1-1.html>

Design Type-head **Uber**

$$\text{TrieNode} \begin{cases} \text{next}[26] \\ \text{maxheap}(\text{TrieNode}) \end{cases} \text{ (size of maxheap} \leq 10 \text{)}$$

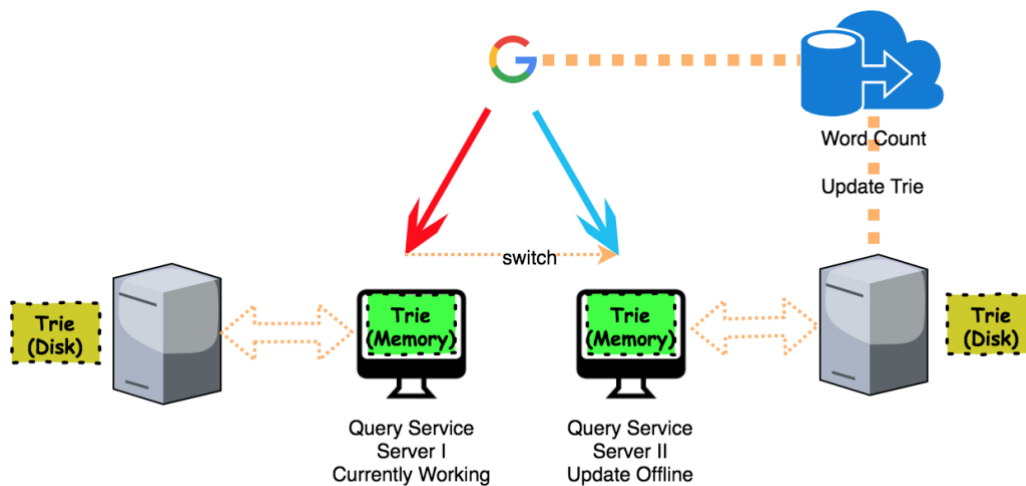


Workable solution: when user types a prefix, say “am” into google search textbox, it automatically makes a request to the *QueryServices*, and the *QueryServices* scan the “am” char by char in the Tire. At node of “m”, there is a list of nodes, say “amazon”, “amaze”, etc. which is associated with it (“am”) are the top 10 hot words starting with prefix “am”.

Since the hot words will be changed every day, so we need to update the trie built so far with the most up-to-date searched words or logs, thus the *Data Collection Service* comes into use for this purpose. This service works at servers to ask data from search logs which are stored in database, doing kind of like sql-queries thing:

```
SELECT Word, Count(Word) AS CNT
FROM WordTable
GROUP BY Word
WHERE DATEDIFF(@CurrTimeStamp - TimeStamp) < 7
HAVING CNT > 1b
```

At this stage, **Data Collection Service** maintains a “bigtable”, then it begins to update the trie. Because we cannot interrupt a working **Query Service** machines, we can copy the trie from disk to disk, and load the trie into another machine’s memory. Last, we update the trie using the **most up-to-date hot words big-table**. After it finished, we can switch the type-head query flows goes into the updated-trie machine.



Scalability: hire multiple Query Service machine

For example, the word *amax* has been search 20b times, then we need to update it to trie four time (because $\text{len}('amax') = 4$). We need to update 4 *prefix*: '*a*', '*am*', '*ama*', '*amax*'. Here, we can compute hash value for all the four prefix, and use $\text{machine id} = \text{hashvalue} \% \text{machines} \cdot \text{number}$ or consistent hashing to target the query service machine.

Storage for search words log

For example, a word will be counted only if its occurrence is more than 1k.

Reservoir Sampling

382. Linked List Random Node Google

```

ListNode p ← head, e = 1, result = null
while(p ≠ null):
    if(rand.nextInt(e++) = 0): result = p
    p = p.next
return result.val

```

398. Random Pick Index Facebook

```

e := 1, re := -1
for(i = 0 → len(A) - 1):

```

```
    if( $A[i] = \mathcal{T}$  &  $rand.nextInt(e++) = 0$ ):  $re := i$   
return  $re$ 
```

336. Palindrome Pairs

Given a list of unique words, find all pairs of distinct indices (i, j) such that $words[i] + words[j]$ is palindromic string.

Solution: $map(words[i]) \rightarrow \{s_0, s_1, \dots, s_n\}$, $words[i] + s_k$ is palindromic string.

Dynamic Programming

238. Product of Array Except Self

$$l_i = l_{i-2} \times a_{i-1},$$

$$r_i = r_{i+2} \times a_{i+1}$$

Minimum number deletion chars **ForUsAll**

making the all characters in ascending order after delete. For example, $str = 'banana'$, delete minimum number characters $'b', 'n', 'n' \rightarrow 'aaa'$

$f = (\text{array}, \text{size} = 26, \text{default} = 0)$

$\text{for}(i = 0 \rightarrow s.\text{length} - 1):$

$\text{temp} = 1$

$\text{for}(j = 'a' \rightarrow s[i]):$

$\text{temp} = \max(\text{temp}, f_j + 1)$

$f[s[i]] = \text{temp}$

$\text{result} = \max(f[s[i]], \text{result})$

10. Wildcard Matching

Q. '?' matches 1 character and '*' matches 0,1, multiple characters.

$f[0,0] = \text{true}$

$\text{for}(i = 1 \rightarrow \text{len}(t)): f[0,i] := f[0,i-1] \ \& \ s[i-1] = *$

$\text{for}(i = 1 \rightarrow \text{len}(s)):$

$\text{for}(i = 1 \rightarrow \text{len}(t)):$

$\text{if}(t[j-1] = ? \text{ or } s[i-1] = t[j-1]): f[i,j] = f[i-1,j-1]$

$\text{elif}(t[j-1] = *): f[i,j] = f[i-1,j-1] \ \& \ f[i,j-1] \ \& \ f[i-1,j]$

K Sum Problem

Intuition & Algorithm

Define $f[i,j,v]$ as the number of options of selecting j numbers from the first i numbers that sum to v . So for the $(i-1)^{th}$ number, there two options can be selected. Option #1, not selecting the $(i-1)^{th}$ number, so $f[i,j,v] := f[i,j,v] + f[i-1,j,v]$. Option #2, selecting the $(i-1)^{th}$ number, so $f[i,j,v] := f[i,j,v] + f[i-1,j-1,v-A[i-1]]$

```

for( $i = 1 \rightarrow n$ ):
    for( $j = 1 \rightarrow k$ ):
        for( $v = target \rightarrow A[i - 1]$ ):
             $f[j, v] := f[j, v] + f[j - 1, v - A[i - 1]]$ 

```

68. Text Justification **Uber**

Solution: Greedy Algorithm, fit as many words as possible per line

Follow Up: Balance the amount of Empty Space Allocated per line

For example, when justify the text 'aaa bb cc dddd' with the maximum 6 characters per line, there are at-least two ways of justification.

a	a	a		b	b
c	c				
d	d	d	d	d	

a	a	a			
b	b		c	c	
d	d	d	d	d	

The cost of empty space of first plan is: $1 + 4^3 + 1^3 = 64$, while that of second plan is $3^3 + 2^3 + 1^3 = 36$, obviously, the second plan is better meaning the empty space allocated per line is more balanced.

Intuition & Algorithm

Dynamic Programming, first we compute costs of all possible lines in a 2D table $lc[][]$, The value $lc[i][j]$ indicates the cost to put words from i to j in a single line where i and j are indexes of words in the input sequences. If a sequence of words from i to j cannot fit in a single line, then $lc[i][j]$ is considered ∞ (to avoid it from being a part of the solution). Once we have the $lc[][]$ table constructed, we can calculate total cost using following recursive formula. In the following formula, $cost[i]$ is the optimized total cost for arranging words from 1 to i .

$$cost_i = \begin{cases} 0, & \text{if } i == 0 \\ \min_{1 \leq j \leq i} (cost_{j-1} + lc_{j,i}), & \text{if } i > 0 \end{cases}$$

```

void text · justification (String[] words, int maxChar)
    n = words.length
    # extra: number of extra space if (words[i], ... words[j]) are put in one single line
    int[ ][ ] extra = [n][n]
    # lc: cost of a line which contains (words[i], ..., words[j])
    int[ ][ ] lc = [n][n]
    int[ ] cost = [n + 1]
    mx = 231 - 1
    for(i = 0 → n - 1):
        for(j = i → n - 1):
            extra[i, j] = maxCharLimit - (∑ij len(words[k]) + j - i)

    for(i = 0 → n - 1):
        for(j = 0 → n - 1):
            if(extra[i, j] < 0): lc[i, j] := 231 - 1
            elif(j = n & extra[i, j] ≥ 0): lc[i, j] = 0 * in case of last line
            else: lc[i, j] = (extra[i, j])2

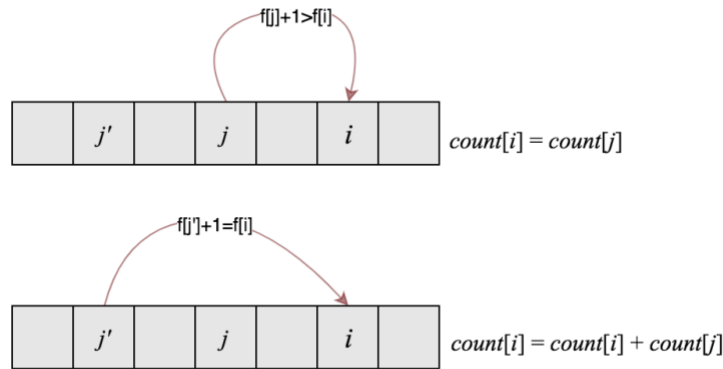
    cost[0] = 1 * if there is no more word to be dealt with
    for(i = 1 → n):
        cost[i] = 231 - 1
        for(pre = 1 → i - 1):
            if (cost[pre] ≠ mx & lc[pre - 1, i - 1] ≠ mx
                cost[pre] + lc[pre - 1, i - 1] < cost[i]):
                cost[i] := cost[pre] + lc[pre - 1, i - 1]
                path[i - 1] = pre - 1

    print · solution(path, idx):
        do:
            print(idx → path[idx])
            idx = path[idx] - 1
        while(path[idx] ≠ 0)

```

673. Number of Longest Increasing Subsequence Facebook

Given an unsorted array of integers, find the number of longest increasing subsequence.



Intuition & Algorithm

Apply DP method by defining $f[i]$ as the LIS ending up with index i . Also, define $counts[i]$ as the number of LIS ending up with index i .

```

int[ ] f := [n]
int[ ] counts := [n], counts[i] := 1
LIS := 0
for(i = 0 → n - 1):
    mx := 1
    for(j = 0 → i - 1):
        if(A[j] < A[i] & f[j] + 1 > mx):
            counts[i] := counts[j]
            mx := f[j] + 1
        elif(A[j] < A[i] & f[j] = mx):
            counts[i] := counts[i] + counts[j]
    f[i] := mx
    LIS := max(LIS, f[i])

re := 0
for(i = 0 → n - 1):
    if(f[i] = mx): re := re + counts[i]
return re

```

10. Regular Expression Matching

[Google](#) [Facebook](#) [Twitter](#) [Snapchat](#) [Two Sigma](#)

Intuition & Algorithm

'.' Matches 1 character, $\{e\}^*$ matches $\{0,1,\infty\}$ character 'e'

```
int[ ][ ] f = [len(s) + 1][len(t) + 1]
f[0,0] = true, f[0,1] = false
for(i = 2; i ≤ len(T)): f[0,i] := f[0,i - 2] & s[i - 1] = *
for(i = 1 → len(S)):
    for(i = 1 → len(t)):
        if(T[j - 1] = *):
            if(S[i - 1] = T[j - 2] || T[j - 2] = .): * S[i - 1] can match T[j - 2]
                f[i,j] := f[i,j] || (f[i - 1,j - 2] || f[i - 1,j] || f[i,j - 2])
            else: * S[i - 1] doesn't match T[j - 2]
                f[i,j] := f[i,j] || f[i,j - 2] ⇒ threat (T[j - 2], T[j - 1]) as (T[j - 2])0
        elif(T[j - 1] = .):
            f[i,j] := f[i,j] || f[i - 1,j - 1]
        elif(T[j - 1] = S[i - 1]):
            f[i,j] := f[i,j] || f[i - 1,j - 1]
```

97. Interleaving String [Uber](#)

Given three strings s_1, s_2, s_3 , determine if s_3 is formed by interleaving s_1, s_2

Intuition & Algorithm

```
int[ ][ ] f = [len(s1 + 1)][len(s2 + 1)]
f[i, 0] = s1[0:i - 1] = s3[0:i - 1] → true: false
f[0, j] = s2[0:j - 1] = s3[0:j - 1] → true: false
for(i = 1 → len(s1)):
    for(j = 1 → len(s2)):
        if(f[i - 1, j] & s1[i - 1] = s3[i + j - 1]): f[i, j] = true
        if(f[i, j - 1] & s2[j - 1] = s3[i + j - 1]): f[i, j] = true
return f[len(s1)][len(s2)]
```

5. Longest Palindromic Substring Uber

Solution I: $f_{i,j} = f_{i+1,j-1}$ & $s_{i+1} = s_{j-1}$ Time Complexity: $O(K^2)$, Space Cost: $O(K^2)$

Solution II: Time Complexity: $O(K^2)$, Space Cost: $O(1)$

$maxLen = 0, re = (l = 0, r = 0)$

for ($i = 0; i < s.length$):

$int[] p \leftarrow centerAt(s, i, i)$ ★ *find the left & right boundary of s s.t. $s[p_0, p_1]$ is palindromic*

$int[] q \leftarrow centerAt(s, i, i + 1)$ ★ *find the left & right boundary s.t. $s[p_0, p_1]$ is palindromic*

if ($p[1] - p[0] > maxLen$):

$maxLen = p[1] - p[0]$

$(re.l, re.r) = (p[0], p[1])$

if ($q[1] - q[0] > maxLen$):

$maxLen = q[1] - q[0]$

$(re.l, re.r) = (q[0], q[1])$

return $s[re.l, re.r]$

516. Longest Palindromic Subsequence

Given a string s , find the longest palindromic subsequence's length in s . You may assume that the maximum length of s is 1000.

410. Split Array Largest Sum Facebook

Split the array into m consecutive subarrays s.t. the largest subarray's sum smallest.

Intuition & Algorithm

Binary-Search by initializing the range $\mathcal{L} := \min(A[i]), \mathcal{R} := \sum_0^{n-1} A[i]$. Then, each time picking up the middle value as the *guess*, and testify whether we can split array into m chunks with the largest sum equals to the *guess*. If we can, we continue test smaller largest sum by $\mathcal{R} := mid - 1$.

1. Otherwise, $\mathcal{L} := mid + 1$

int $splitArray(int[] A, int m)$:

$long l := \max(A[i]), \forall i \in [0, n - 1]$

$long r := \sum_0^i A[i]$

```

re := r
while(l ≤ r):
    mid :=  $\frac{l+r}{2}$ 
    sum := 0, chunks := 1
    for(i = 0 → n - 1):
        if(sum + A[i] > mid):
            chunks := chunks + 1
            sum := A[i]
        else: sum := sum + A[i]
    if(chunks ≤ m):
        re := min(re, mid)
        r := mid - 1
    else: l := mid + 1
return (int) re

```

53. Maximum Subarray

```

global[i] = max(global[i - 1], local[i])
local[i] = max(local[i - 1] + a[i], a[i])

```

Constant Space Solution

```

max · here = A[0], re = A[0]
for(i = 1 → A.length;):
    max · here = max(maxh · here, 0) + A[i]
    re = max(re, max · here)

```

Follow Up: Maximum Subarray with At-least Length of K

```

for(i = 0 → k - 1): sum = sum + a[i]
presum = 0, premin = 0, re = sum
for(i = k → n - 1):
    sum = sum + a[i]
    presum = presum + a[i - k]
    premin = min(premin, presum)
    re = max(re, sum - premin)

```

691. Stickers to Spell Word IXL

42. Trapping Rain Water

$water[i] = \min(left^{high}[i], right^{high}[i]) - height[i]$
 $re := re + water[i]$

62. Unique Paths & 63. Unique Path II

$f_{i,j} \leftarrow grid_{i,j} == 0 ? 0 : f_{i-1,j} + f_{i,j-1}$

174. Dungeon Game

$f[m-1, n-1] = 1$
 $choice_1 \leftarrow f_{i+1,j} - dungeon_{i,j} \geq 1 ? f_{i+1,j} - dungeon_{i,j} : 1$
 $choice_2 \leftarrow f_{i,j+1} - dungeon_{i,j} \geq 1 ? f_{i,j+1} - dungeon_{i,j} : 1$
 $f[i,j] = \min(choice_1, choice_2)$

64. Minimum Path Sum

$f[i,j] = \min(f[i,j-1], f[i-1,j]) + grid[i,j]$

474. Zeros & Ones

Find the maximum number of strings that you can form with given m 0s and n 1s. Each 0 and 1 can be used at most once.

Intuition & Algorithm

Define $f[i,j]$ as the max number of strings that we can form with given i 0s and j 1s. Apply “01” backpack problem solver.

$$f[i,j] = \max(f[i - \#0(s[k]), j - \#1(s[k])]) + 1$$

$for(k = 1; k \leq n;)$

$\#0 = count \cdot zero(s[k]), \#1 = count \cdot one(s[k])$

$for(i = m \rightarrow \#0):$

$for(j = n \rightarrow j \geq \#1):$

$$f[i,j] = \max(f[i - \#0(s[k]), j - \#1(s[k])]) + 1$$

322. Coin Change

Find minimum amount of coin change making up total amount, each coin can be used multiple times.

Intuition & Algorithm

Apply multiple backpack problem solver.

```
for(i = 1 → n):  
    for(j = value(i - 1) → total · amount):  
        f[j] = min(f[j - values[i - 1]]) + 1
```

279. Perfect Square [Google](#)

$$f[n] = \min(f[n - k^2]) + 1, \forall k \in [1, \sqrt{n}]$$

```
int DP(int n, int[] f):  
    if(n = 0): return 0  
    if(f[n] ≠ nil): return f[n]  
    for(v := 0 → A[i]):  
        if(A[i] ≤ v): f[i, v] := min(f[i, v], f[i - 1, v - A[i]] + 1)
```

472. Concatenated Words

$$f[i] = f[j] \ \& \ wordDict[s[j + 1:i]] \neq nil$$

140. Word Break II

139. Word Break II [Google](#) [Uber](#) [Twitter](#) [Snapchat](#) [Dropbox](#)

```
bool[] f = [len(s)]  
for(i = 0 → len(s) - 1):  
    if(dict[s[0:i]] ≠ nil):  
        f[i] = true, continue  
    for(j = i - 1 → 0):  
        if(dict[s[j + 1:i]] ≠ nil & f[j]): f[i] = true, break
```

$$re = [\phi]^{string}, cand = [\phi]^{string}$$
$$DFS(re, cand, dict, s, index)$$

return re

DFS(re, cand, dict, s, index):

if(index = len(s)):

re.add(cand[i] + " ",) $\forall i \in [0, \text{len}(cand) - 1]$

return

for(next = index \rightarrow len(s) - 1):

if(f[next] & dict[s[index:next]] \neq nil):

cand.add(s[index:next])

DFS(re, cand, dict, s, next + 1)

cand.remove(len(cand) - 1)

131. Palindrome Partitioning

132. Palindrome Partitioning II

palindrome[i, j] := palindrome[i + 1, j - 1] & s[i] = s[j]

for(i = 0 \rightarrow len(s) - 1):

cut[i] := (palindrome[0, i] = True) \rightarrow 0: i

for(pre = 0 \rightarrow i - 1):

if(palindrome[pre + 1, i] = True): cut[i] := min(cut[i], cut[pre] + 1)

cut[i] := min(cut[i], cut[pre] + (i - pre))

416. Partition Equal Subset Sum eBay

half = $\frac{(\sum_0^{n-1} A[i])}{2}$

for(i = 1 \rightarrow n):

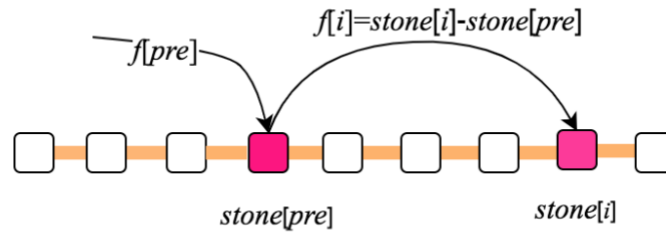
for(ν = half \rightarrow A[i - 1]):

f[ν] := f[ν] || f[ν - A[i - 1]]

403. Frog Jump Snapchat

Given a list of stones' positions (in units) in sorted ascending order, determine if the frog is able to cross the river by landing on the last stone. Initially, the frog is on the first stone and assume

the first jump must be 1 unit. If the frog's last jump was k units, then its next jump must be either $k - 1$, k , or $k + 1$ units. Note that the frog can only jump in the forward direction.



Intuition & Algorithm

The idea here is to maximize the jumps reach to $stone[i]$, so we define $f[i]$ as the maximal jumps reach to $stone[i]$.

$f[0] := 0, f[1] := 1$

for($i = 2 \rightarrow n - 1$):

for($pre = 0 \rightarrow i - 1$):

if($f[pre] < 0$): *continue* ** means this guy cannot make to pre*

$curstep := stone[i] - stone[pre]$

if($f[i] = curstep \pm 1 \parallel f[i] = curstep$):

$f[i] := curstep$

break

return $f[i] > 0 \rightarrow True: False$

376. Wiggle Subsequence

$f_{min}[i] = \max(f_{max}[j]) + 1, \text{ if } A[j] > A[i]$

$f_{max}[i] = \max(f_{min}[j]) + 1, \text{ if } A[j] < A[i]$

$re = \max(f_{min}[i], f_{max}[i])$

120. Triangle Google

$f[i, j] = \max(f[i + 1, j], f[i + 1, j + 1]) + triangle[i, j]$

368. Largest Divisible Subset

```

sort(A)
f[i] = max(f[j]) + 1, ∀j ∈ [0, i - 1], A[i] % A[j] = 0

```

```

int[ ] pre = [n], int[ ] f = [n]
pre[i] := -1, ∀i ∈ [0, n - 1]
f[i] := 1, ∀i ∈ [0, n - 1]
for(i = 0 → n - 1):
    Max := -231, maxj = -1
    for(j = 0 → i - 1):
        if(A[i] % A[j] = 0 & f[j] > Max):
            Max := f[j], maxj := j
    if(maxj ≠ -1):
        pre[i] = maxj, f[i] = Max + 1
maxi := -1, max := -1
for(i = 0 → n - 1):
    if(f[i] > max):
        max := f[i], maxi := i
pos := maxi
Do: re.add(A[pos]), p := pre[p]
while(p ≠ -1)
return reverse(re)

```

Stock Problem Series

121. Best Time to Buy & Sell Stock I

```

dif[i] := prices[i] - prices[i - 1], ∀i ∈ [1, n - 1]
return maxSumSubarray(dif)

```

122. Best Time to Buy & Sell Stock II

Find all ascending subarray $prices[i:j]$ such that $prices[i] < \dots < prices[j]$

$$result := result + \sum_i^j prices[p]$$

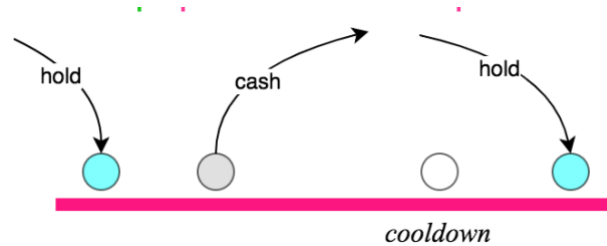
123. Best Time to Buy & Sell Stock III

188. Best Time to Buy & Sell Stock IV

$global[i, j] := \max(global[i - 1, j], local[i, j])$

$local[i, j] := \max(local[i - 1, j], global[i - 1, j - 1]) + dif[i]$

309. Best Time to Buy & Sell Stock with Cooldown Google



$hold := -prices[0], cash := 0, cooldown := 0$

for($i = 1 \rightarrow \text{len}(\text{prices}) - 1$):

$temp := hold$

★ either still hold when it's held at previous day,

★ or buy a new share of stock when previous day is cooldown day.

$hold := \max(hold, cooldown - prices[i])$

★ if today is cooldown day, then previous day must either be cooldown day or selling day

$cooldown := \max(cooldown, cash)$

★ if today is selling day, then previous day must either be selling day or holding day

$cash := \max(cashsell, temp + prices[i])$

return $\max(sell, cooldown)$

714. Best Time to Buy and Sell Stock with Transaction Fee Facebook Bloomberg

At the end of the i^{th} day, we maintain *cash*, the maximum profit we have sold the stock at-hand, and *hold*, the maximum profit we could have if we owned a share of stock.

max · profit(*prices*, *fee*):

$nothold = [n], hold = [n]$

$nothold[0] = 0, hold[0] = -prices[0]$

for($i = 1 \rightarrow \text{len}(\text{prices}) - 1$):

```

    nothold[i] := max(nothold[i - 1], hold[i - 1] + prices[i] - fee)
    hold[i] := max(hold[i - 1], nothold[i - 1] - prices[i])
return cash

```

Constant Space Optimization

```

max · profit(prices, fee):
    * cash: max profit when no share of stock at hand
    * hold: max profit when holding a share of stock
    cash = 0, hold = -prices[i]
    for(i = 1 → len(prices) - 1):
        * sell current stock get profit of prices[i], and minus the transaction fee
        cash = max(cash, hold + prices[i] - fee)
        * buy current stock with prices[i]
        hold = max(hold, cash - prices[i])
    return cash

```

198. House Robber I [LinkedIn](#) [Airbnb](#)

```

int[ ] rob(int[ ] A):
    if(len(A) = 1): return A[0]
    yes = A[0], no = 0, re := yes
    for(i = 1 → n - 1):
        temp := no
        no = max(yes, no)
        yes = no + A[i]
        re := max(yes, no)
    return re

```

213. House Robber II (Rob in a Circle)

```

re := max(rob(A, 0, n - 2), rob(A, 1, n - 1))

```

337. House Robber III (Rob in a BT)

```

re := -231

```

```

int[ ] maxRob(TreeNode o):
    int[ ] left = maxRob(o.left)
    int[ ] right = maxRob(o.right)
    notRob = max(left0, left1) + max(right0, right1)
    rob = left0 + right0 + o.val
    rob[0], rob[1] = notRob, rob
    re = max(re, rob[0], rob[1])
    return rob

```

70. Climbing Stairs [Apple](#) [Adobe](#)

$$f_i = f_{i-1} + f_{i-2}, f_0 = 0, f_1 = 1$$

Apply the same formula as Fibonacci equation, can use fast exponentiation matrix algorithm.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{\frac{n}{2}} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{\frac{n}{2}} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{(n\&1)}$$

$$F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

72. Edit Distance

$$f[i, j] := f[i - 1, j - 1], word_1[i] = word_2[j]$$

$$f[i, j] := \min(f[i - 1, j], f[i, j - 1], f[i - 1, j - 1]) + 1$$

115. Distinct Subsequences [Google](#)

$s[0:i - 1]$ contains how many # of $t[0:j - 1]$

$$f_{i,j} = f_{i-1,j-1} + f_{i-1,j} \text{ (if } s_{i-1} = t_{j-1})$$

$$f_{i,j} = f_{i-1,j} \text{ (if } s_{i-1} \neq t_{j-1})$$

87. Scramble String

$$dfs(s, t, ls, rs, lt, rt)$$

95. Unique Binary Search Trees I Snapchat

```
int[ ][ ] f := [n][n]
for(i = 0 → n - 1):
    for(j = 0 → i): f[i, j] := 1
for(len = 2 → n):
    for(i = 0 → i + len ≤ n):
        j := i + len - 1
        for(k = i → j):
            if(k = i): f[i, j] := f[i, j] + f[k + 1, j]
            elif(k = j): f[i, j] := f[i, j] + f[i, k - 1]
            else: f[i, j] := f[i, j] + f[i, k - 1] * f[k + 1, j]
```

96. Unique Binary Search Trees II

```
List<TreeNode> dfs(l, r):
    if(l > r): return null
    List<TreeNode> roots = {ϕ}
    for(k = l; k < r;)
        List<TreeNode> lchilds = dfs(l, k - 1)
        List<TreeNode> rchilds = dfs(k + 1, r)
        for(TreeNode lc: lchilds):
            for(TreeNode rc: rchilds):
                TreeNode o = new TreeNode(nums[k])
                o.left = lc
                o.right = rc
                roots.add(o)
    return roots
```

304. Range Sum Query 2D – Immutable

308. Range Sum Query 2D - Mutable

Intuition & Algorithm #1

[Fenwick Tree](#)

Time Cost: $O(\log(m) * \log(n))$ per update or get sum operation.

```

class FenwickTree
    int R, C
    int[ ][ ] sum
    public FenwickTree(R, C): sum = [R + 1][C + 1]
    void update(int r, int c, int delta):
        for(x = r; x ≤ R; x := x + x & (-x)):
            for(y = c; y ≤ C; y := y + y & (-y)):
                sum[x, y] := sum[x, y] + delta
    int getSum(r, c):
        for(x = r; x ≥ 1; x := x - x & (-x)):
            for(y = c; y ≥ 1; y := y - y & (-y)):
                re := re + sum[x, y]

```

Intuition & Algorithm #2

Compress the 2d matrix into 1d array, then apply the algorithm the same as “find subarray with largest sum”.

Time Cost: $O(m * n)$ for initialization, $O(m)$ for update and $O(n)$ for get-sum

```

for(j = 0 → len(mat[0]) - 1):
    colsum[j, 0] := mat[i, j]
    for(i = 1 → len(mat) - 1):
        colsum[j, i] = colsum[j, i - 1] + mat[i, j]
sumRegion(r1, c1, r2, c2):
    for(c = c1 → c2):
        re := re + colsum[c, r2] - colsum[c, r1] + mat[r1, c]
update(r, c, val):
    for(i = r; i < matrix.length; ):
        colSumc,i += val - matrixi,c

```

303. Range Sum Query Immutable

307. Range Sum Query Mutable

$sum_i = arr_i + sum_{i-1}$

Mutable: Segment tree (*avg* $\log(n)$ time)

```

buildSegmentTree(int l, int r):
    if(l > r): return null
    mid ← l + ((r - l) >> 1)
    Node o ← new Node(l, r)
    o.left ← build(l, mid):
    o.right ← build(mid + 1, r)
    return o;

update(Node o, int pos, int val):
    if(o.l == o.r & o.l == pos): o.sum = val; return;
    mid = l + ((r - l) >> 1)
    if(pos ≤ mid): update(o.left, pos, val)
    else: update(o.right, pos, val)
    o.sum += o.left.sum + o.right.sum

querySum(Node o, int from, int to):
    if(o.l == from & o.r == to): return o.sum
    mid = l + ((r - l) >> 1)
    if(to ≤ mid): return querySum(oleft, from, to)
    elif(from ≥ mid): return querySum(oright, from, to)
    else: return querySum(oleft, from, mid) + querySum(oright, mid + 1, to)

```

312. Burst Balloons Google Snapchat

Intuition & Algorithm

Add two balls (value=1) to the head & tail of ball list. By defining $f[i, j]$ as the maximum score can get from $ball[i: j]$. Suppose the last balloon to be removed is $ball[k]$, then we have such formula: $f[i, j] = \max(f[i, k] + f[k, j] + a[i] * a[j] * a[k]), \forall k \in [i, j]$

```

int maxCoins(int[ ] nums):
    N := len(nums)
    int[ ] A := [N + 2]
    A[0] = A[N + 1] = 1
    for(i = 0 → len(nums) - 1): A[i + 1] = nums[i]

```

```

int[][] f := [N + 2][N + 2]
for(len = 2 → N + 2):
    for(i = 0 → i + len ≤ N + 2):
        j := i + len - 1
        for(k = i + 1 → j - 1):
            f[i, j] := f[i, j] + f[i, k] + f[k, j] + A[i] * A[j] * A[k]
return f[0, N + 1]

```

354. Russian Doll Envelopes

Solution: Sort envelopes by width, then by length if two envelopes share the same width

435. Non-overlapping Intervals

Sort the envelopes/intervals list (first length, then width), then apply LIS mode algorithm.

300. Longest Increasing Subsequence

$O(n^2)$ solution: LIS mode algorithm

$O(n \cdot \log(n))$ solution:

```

longest · increasing · subseq(arr: type = int array):
    f = array1d(default = 0)
    len = 0
    for(int x: arr):
        pos = Arrays.binarySearch(f, 0, len, x)
        pos = pos < 0 ? -(pos + 1) : pos
        f[pos] = x
        if(pos == len): len = len + 1
    return len

```

221. Maximal Square [Facebook](#) [Apple](#) [Airbnb](#)

$f[i, j] := \min(f[i - 1, j], f[i, j - 1], f[i - 1, j - 1]) + 1$

674. Longest Continuous Increasing Subsequence [Facebook](#)

$f[i] := f[i - 1] + 1, \text{ if } \mathcal{A}[i] = \mathcal{A}[i - 1] + 1$

$f[i] := 0, \text{ otherwise}$

651. 4 Keys Keyboard [Google](#) [Microsoft](#)

(A, Ctrl-A, Ctrl-C, Ctrl-V), print as much 'A' as possible

Intuition & Algorithm

Define $f[i]$ as the maximum number of A's printed with i steps. Assume we already know how many A's can be printed with the best strategy in j steps, namely $f[j]$. It is clearly that the current best strategy is to collect all the A's already have (Ctrl-A), then copy all of them (Ctrl-C) and finally paste them back to the screen (Ctrl-V) one time or multiple times. So it takes at least 3 steps to finish this series. For example, $i = 10, j = 6$, e, then the best strategy here is Ctrl-A, Ctrl-C, Ctrl-V, Ctrl-V. As a result, we can get $f[j] * (10 - 6 - 1) = 3 * f[j]$.

$$f[i] = \max(f[j] * (i - j - 1)), \forall j \in [0, i - 3]$$

650. 2 Keys Keyboard [Microsoft](#)

the minimum step of printing n 'A' s

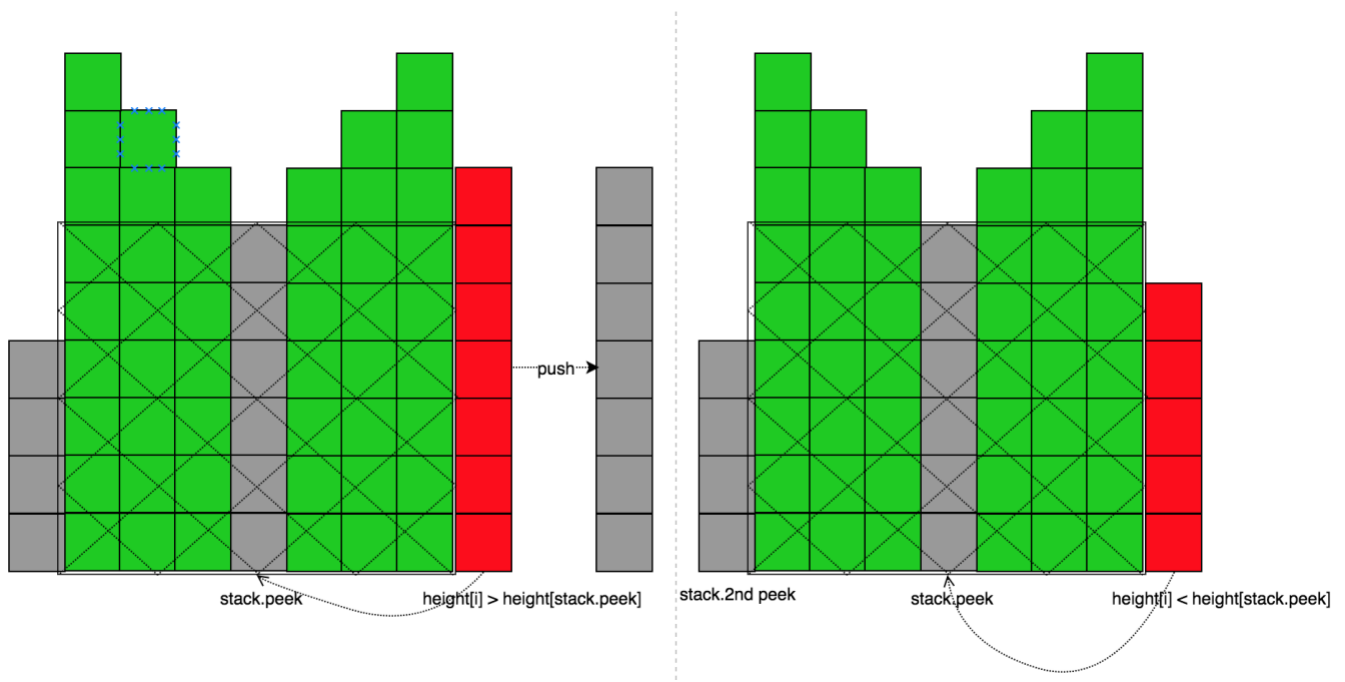
Intuition & Algorithm

CopyAll & Paste, print as much 'A' as possible. Assume the minimum number of steps for printing j A's is $f[j]$. Then the best strategy is *copyall, paste, ...*. For example, $i = 9, j = 3$, then the minimum cost is $f[i] = f[j] + \frac{9}{3} = f[j] + 3$ because a series of *copyall, paste, paste* action is required.

$$f[i] = \min\left(i, f[j] + \frac{i}{j}\right), \forall j, f[i] = k * f[j]$$

85. Maximal Rectangle

84. Largest Rectangle in Histogram [Facebook](#)



Intuition & Algorithm

Always maintain a stack which maintains the ascending order of $height[i]$, as the above diagram suggests, the maximal rectangular area form by $stack.peak$ is:

$$area = (h[st.peak]) * (i - st.2nd\ peek - 1)$$

$largestRectangularArea(int[] h):$

$re := 0, st := Stack$

★ append $h[n] = -1 \rightarrow$ very important

for($i = 0 \rightarrow h.length$):

$height := (i = h.length) \rightarrow -1: h[i] \Rightarrow VERY IMPORTANT !!!!!$

while($\sim st.empty \ \& \ h[st.peak] > height$):

$top := st.pop$

if($st.empty$): $re = \max(re, h[top] * (i - st.peak - 1))$

else: $re := \max(re, h[top] * i)$

$st.push(i)$

return re

363. Max Sum of Rectangle No Larger Than k [Google](#)

Intuition & Algorithm

Convert 2d problem into “find max subarray no larger than k”

```

for(j = 0 → n - 1):
    col[j, 0] = mat[i, j]
    for(i = 1 → m - 1):
        col[j, 0] := col[j, i - 1] + mat[i, j]

re = 0
for(r1 = 0 → m - 1):
    for(r2 = r1 → m - 1):
        set = (ϕ)TreeSet, sum = 0 ★ set: store the presum in ascending order
        set.add(0)
        int[ ] A = [n]
        ★ compress multiple rows into one line
        for(j = 0 → n - 1):
            A[j] = col[j, r2] - col[j, r1] + mat[r1, j]
            sum = sum + A[j]
            ★ if there exists  $\sum_0^k A[k] \geq \text{sum} - k$ , then there exists  $\sum_{k+1}^j A[k] \leq k$ 
            e = s.ceilKey(sum - k)
            if(e ≠ null): result = max(result, sum - e)
            set.add(sum)

return result

```

Follow Up: Max Sum Rectangle

Find a rectangular area whose sum equals s

```

result = 0
for(r1 = 0 → m - 1):
    for(r2 = r1 → m - 1):
        for(j = 0 → n - 1): colj = ar2,j - ar1,j + matrixr1,j
        result = max(result, maxsubarray(col))

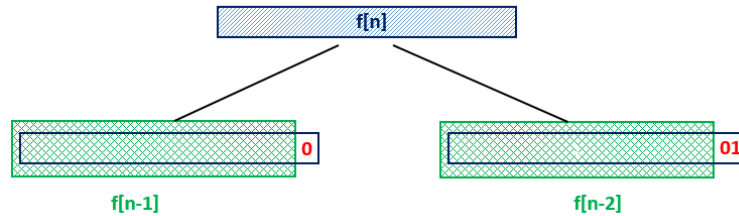
return result

```

600. Non-negative integers without consecutive ones Pocket Gems

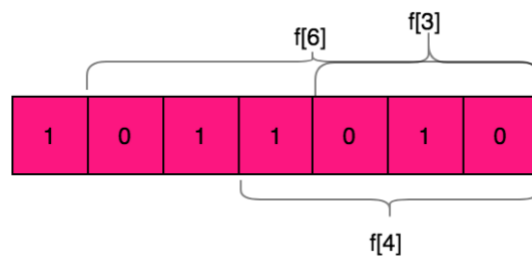
Intuition

Suppose, we need to find the count of binary numbers with n bits such that these numbers don't contain consecutive 1's. In order to do so, we can look at the problem in a recursive fashion. Suppose $f[i]$ gives the count of such binary numbers with i bits. To determine the value of $f[n]$, which is the requirement, we can consider the cases shown below:



From the above figure, we can see that if we know the value of $f[n - 1]$ and $f[n - 2]$. In order to generate the required binary numbers with n bits, we can append a 0 to all the binary numbers contained in $f[n - 1]$ without creating an invalid number. But, we can't append a 1 to all these numbers, since it could lead to the presence of two consecutive ones in the newly generated numbers. Thus, for the currently generated numbers to end with a 1, we need to ensure that the second last position is always 0. Thus, we need to fix a 01 at the end of all the numbers contained in $f[n - 2]$. Thus, in total, we get $f[n] = f[n - 1] + f[n - 2]$

For example, $n = \frac{101100}{543210} (44)$, at the digit of 5,3,2 (d_5, d_3, d_2) they are all 1's. Thus, $result = f_5 + f_3 + f_2$, since (d_3, d_2) are consecutive 1's, so it stops counting at d_2



Intuition & Algorithm

First, calculate the number of integers without consecutive ones in length of n

$$f[n] := f[n - 1] + f[n - 2]$$

Second, for each of bit set to one $bit[i] = 1$, sum all corresponding $f[i]$ up ($re := re + f[i]$) until consecutive one's occurs ($bit[i] := 1 \ \& \ bit[i - 1] := 1$). Finally, if consecutive one's occurs, return $re - 1$

```
f[0] = 1, f[1] = 2, f[2] = 3
for(i = 3 → 32): f[i] = f[i - 1] + f[i - 2]
i = 31, re = 0, prevbit = 0
while(i ≥ 0):
    if ((num & (1 << i)) >> i = 1):
        re := re + f[i]
        if(prevbit = 1): break
        prevbit = 1
    else: prevbit = 0
    i := i - 1
return re + checkIfExistsConsecutiveOnes(num)
```

```
int checkIfExistsConsecutiveOnes(num):
    preb = 0, le = 32 - int.leadingZeros(num)
    i = le - 1
    while(i ≥ 0):
        if(num & (1 << i) ≠ 0):
            if(preb = 1): return 1
            preb = 1
        else: preb = 0
        i := i - 1
    return 0
```

152. Maximum Product Subarray [LinkedIn](#)

```
result = A[0]
minhere = A[0], maxhere = A[0]
for(i = 1 → len(A) - 1):
    temp = minhere
    minhere = min(minhere, maxhere, 1) * A[i]
```

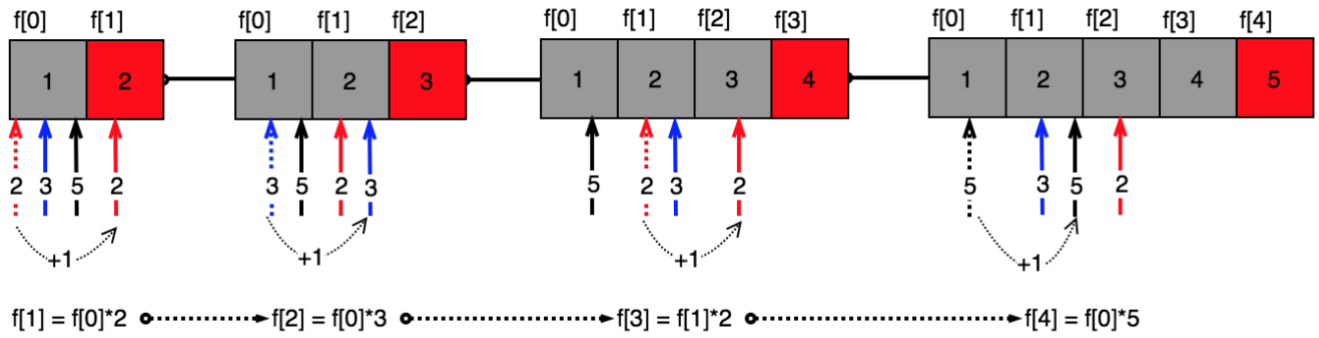
$$max_{here} = \max(temp, max_{here}, 1) * A[i]$$

$$re = \max(max_{here}, re)$$

313. Super Ugly Number

264. Ugly Number I

263. Ugly Number II



```
for(i = 0 → len(primes) - 1):
```

```
    prime2index[primes[i]] := 0
```

```
int[] f = [n + 1], f[0] = 1
```

```
for(i = 1; i < n):
```

```
    minv = 231 - 1, minp = -1
```

```
    for(p = 0 → len(primes) - 1):
```

```
        if (f[prime2index[primes[p]]] * primes[p] > min):
```

```
            minv = f[prime2index[primes[p]]] * primes[p], minp = p
```

```
    f[i] = minv
```

```
    prime2index[primes[minp]] := prime2index[primes[minp]] + 1
```

```
    for(p = 0 → len(primes) - 1):
```

```
        if (minp ≠ p & f[prime2index[primes[p]]] * p = minv):
```

```
            prime2index[primes[p]] := prime2index[primes[p]] + 1
```

91. Decode Ways Facebook Microsoft Uber

$f_i = f_{i-1} + f_{i-2}$ (if s_i, s_{i-1} can form a number $\in [1, 26]$)

$f_i = f_{i-1}$

338. Counting Bits

$$f[i] = f[i \gg 1] + (i \& 1)$$

329. Longest Increasing Path in a Matrix Google

$$f[i, j] = \max(f[i - 1, j], f[i, j - 1], f[i + 1, j], f[i, j + 1]) + 1$$

241. Different Ways to Add Parentheses

Given a string of numbers and operators, return all possible results from computing all the different possible ways to group numbers and operators. The valid operators are +, -, *.

Tree Set & Tries & Linked List

456. 132 Pattern

```

lmini = min(lmini-1, A[i]):
for(i = n - 1 → 0):
    j = set.lower(A[i]) * find the largest value that is smaller than a[i]
    rmini := j = nil → A[i], j
    set.add(A[i])
for(i = 1; i ≤ n - 2;):
    if(lmini < rmini < A[i]): return true
return false

```

Breaking Bad Uber

Given a list of words $word \cdot list$, and a string $name$, wrap all substring of $name$ which show up in $word \cdot list$ with '[' & ']', and capitalize the first letter of such substring. For example,

```
word · list = {'apple', 'bce', 'bcd', 'e', 'app'}
```

```
name = iloveapple,
```

```
namwnew = ilov[E][Apple]
```

(note: both $name_{5,7} = 'app'$ & $name_{5,9} = 'apple'$ show up in the $word \cdot list$. It should choose the longer one $name_{5,9} = 'apple'$ to wrap out.

<https://drive.google.com/open?id=1QdTRFv-RNSd3CypcFNz1s7fYdOtGQuPV>

421. Maximum XOR of Two Numbers in an Array

```
for(int x: nums):
    result = max{result, x  $\oplus$  most · different(x)}
most · different(TrieNode root, x):
    TrieNode p = o, x = 0
    for(i = 31  $\rightarrow$  0):
        mask = 1  $\ll$  i
        b = (x & mask)  $\gg$  i
        if(p.next[1 - b]  $\neq$  null):
            x = x * 2 + (1 - b)
            p = p.next[1 - b]
        else:
            x = x * 2 + b
            p = p.next[b]
    return x
```

211. Add & Search Word (Single word)

212. Add & Search Word II (Multiple words) [Google](#) [Airbnb](#) [Microsoft](#)

```
for(i = 0  $\rightarrow$  A.length - 1):
    for(j = 0  $\rightarrow$  A[0].length - 1):
        if( $\mathcal{R}$ .next[A[i, j] - a]  $\neq$  nil):
            used[i, j] := True
            s := ""
            dfs(A, i, j,  $\mathcal{R}$ .next[A[i, j] - a], used, s + A[i, j])
            used[i, j] := False

void DFS(A, i, j, p, used, s):
    if(p.isword):
        re.add(s)
        ★ DO NOT return here
    for((nx, ny)  $\in$  (i, j).next):
        index := A[nx, ny] - a
        if(p.next[index]  $\neq$  nil & used[nx * n + ny] = nil):
```

```

used[nx, ny] := True
DFS(A, nx, ny, p.next[index], used, s + A[nx, ny])
used[nx, ny] := False

```

327. Count of Range Sum

Solution: use variable sum to keep track the prefix sum so far, use TreeMap to store it.

$$sum = \sum_0^i nums_i$$

if a subarray $nums[i:j]$ s. t. $\sum_i^j nums_k \in [lower, upper]$, then there must exists a prefix sum:

$$prefix \cdot sum = \left[\sum_0^i nums_i - upper, \sum_0^i nums_i - lower \right]$$

```

int countRangeSum(nums(array), lower(int), upper(int)):
    if(nums == null || nums.length == 0): return 0
    n = nums.length
    result = 0, sumlong = 0
    SortedMap<Long, Integer> smap = null
    TreeMap<Long, Integer> mp
    for(int x: nums):
        sum ← sum + x
        sortedmp ← mp.subMap(sum - upper, sum - lower, true)
        for(int e: sortedmp.values): result := result + e
        mp[sum] := mp[sum] + 1
    return result

```

24. Swap Nodes in Pairs

25. Reverse Nodes in k-group [Facebook](#) [Microsoft](#)

Each time we find k nodes and reverse the order, then recursively make a call to itself dealing with the remaining nodes.

```

ListNode reverseKGroup(ListNode head, int k):
    if(head == nil or k ≤ 1): return head

```



```

cnt = 0, p = head
for(; p ≠ nil & cnt < k; cnt := cnt + 1): p := p.next
★ if there are not enough # of nodes within this group, no need for reversing
if(cnt < k): return head
p = head, tail = head, q = p.next, r = nil, rev = 0
for(; rev < k - 1; rev := rev + 1):
    r := q.next
    q.next := p
    p := q
    q := r
ListNode newhead := p
tail.next = reverseKGroup(q, k)
return newhead

```

237. Delete Node in a Linked List [Microsoft](#) [Apple](#) [Adobe](#)

B-Search & Array & Two Pts

48. Rotate image

50. Pow (x, n) [Google](#) [Facebook](#) [Bloomberg](#) [LinkedIn](#)

```

double myPow(double x, int n):
    double v := x, re := 1.0, M := Double.MaxValue
    int sign := n < 0 → -1: 1:
    long N := (long)n
    while(N > 0):
        if((N & 1) = 1): re := re * v
        N >>= 1
        v *= v
    return sign = -1 →  $\frac{1.0}{re}$ : re

```

56. Merge Intervals [Google](#) [Facebook](#) [Microsoft](#) [Bloomberg](#) [LinkedIn](#) [Twitter](#) [Yelp](#)

Given a collection of intervals, merge all overlapping intervals.

```
 $\mathcal{N} := \text{len}(\text{intervals})$ 
if( $\mathcal{N} = 0$ ): return  $[\phi]$ 
if( $\mathcal{N} = 1$ ): return  $[\text{intervals}[0]]$ 
Interval  $prev := \text{intervals}[0]$ 
for( $i = 1 \rightarrow \mathcal{N} - 1$ ):
     $current := \text{intervals}[i]$ 
    if( $prev.end \geq current.start$ ):  $prev.end = \max(prev.end, current.end)$ 
    else:  $re.add(prev), prev := current$ 
 $re.add(prev)$ 
return  $re$ 
```

Follow Up: Merge Two Sorted Interval List /
Collect All Intersection of Two Sorted Interval List

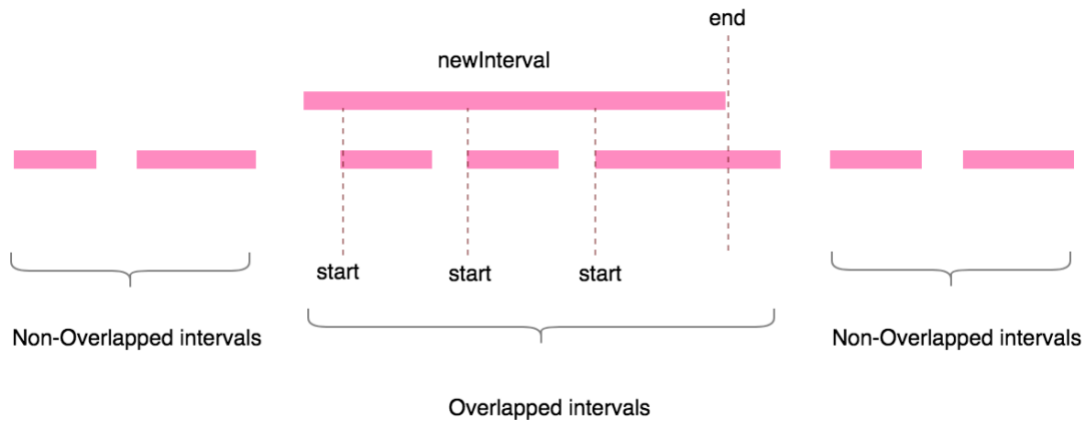
Intuition & Algorithm

Like Merge Two Sorted Arrays

153. Find Minimum in Rotated Sorted Array [Facebook](#)

```
 $l = 0, r = n - 1$ 
while( $l < r$ ):
     $mid = \frac{l+r}{2}$ 
    if( $a[mid] = a[r]$ ):  $r := r - 1$ 
    elif( $a[mid] > a[r]$ ):  $l := mid + 1$ 
    elif( $a[mid] < a[r]$ ):  $r = mid$ 
return  $a_l$ 
```

57. Insert Interval [Google](#) [LinkedIn](#) [Facebook](#)



Intuition & Algorithm

First, scan those non-overlapped intervals. Second, scan those overlapped intervals, Third, scan the remained intervals.

List[Interval] insert(List[Interval] A, Interval new):

n := len(A)

List[Interval] re := [ϕ]

if (n = 0): re.add(new), return re

index := 0

while (index < n & A[index].end < new.start): re.add(A[index++])

while (index < n & new.end ≥ A[index].start):

new = (min(A[index].start, new.start), max(A[index].end, new.end))

index := index + 1

re.add(new)

while (index < n): re.add(A[index++])

return re

154. Find Minimum in Rotated Sorted Array II

l = 0, r = n - 1

while (l < r):

mid = $\frac{l+r}{2}$

if (a[mid] = a[r]): r := r - 1

elif (a[mid] > a[r]): l := mid + 1

elif (a[mid] < a[r]): r = mid

return a_l

461. Hamming Distance [Facebook](#)

```
int hammingDistance(x, y):  
    re := 0  
    for(d = 31 → 0)  
        mask := (1 << d)  
        dx, dy := ((x & mask) >> d), ((y & mask) >> d)  
        if(dx ≠ dy): re := re + 1  
    return re
```

477. Total Hamming Distance [Facebook](#)

Say for any particular bit position, count the number of elements with this bit **ON**. (i.e. this particular bit is 1). Let this count be k . Hence the number of elements with this bit **OFF** (i.e. 0) is $(n - k)$ in an n element array). Thus, at this bit, the total hamming distance is: $k * (n - k)$

```
int totalHammingDistance(int[] A):  
    re := 0  
    bits := [32]  
    for(d = 31 → 0):  
        mask := (1 << d)  
        for(i = 0 → len(A) - 1):  
            dai := ((A[i] & mask) >> d)  
            bits[d] := bits[d] + dai  
        re := re + bits[d] * (len(A) - bits[d])  
    return re
```

277. Find the Celebrity

Intuition & Algorithm

Apply two pointers solver, since the celebrity doesn't know anyone else in the party, and the other guys are not known for the celebrity.

$l = 0, r = n - 1$

```

while( $l \leq r$ ):
    if( $l$  knows  $r$ ):  $l = l + 1$  (because celebrity knows nobody)
    elif( $l$  doesn't know  $r$ ):  $r = r - 1$  (because celebrity should be known by everyone)
check if  $l$  is the celebrity again

```

554. Brick Wall Facebook

There is a brick wall in front of you. The wall is rectangular and has several rows of bricks. The bricks have the same height but different width. You want to draw a vertical line from the top to the bottom and cross the least bricks. If your line go through the edge of a brick, then the brick is not considered as crossed. You need to find out how to draw the line to cross the least bricks and return the number of crossed bricks.

Intuition & Algorithm

Find the position where the most of bricks' edges occurs.

```

int leastBricks(List[List[int]] wall):
    most := 0
    map := {int ⇒ int}
    for(List[int] line: bricks):
        pos := 0
        for( $i = 0 \rightarrow \text{len}(\text{line}) - 1$ ):
            pos := pos + line[i]
            map[pos] := map[pos] + 1
        most := max(most, map[pos])
    return len(wall) - most

```

287. Find the Duplicate Number Google

All number in an array of size $n + 1$ are in the range of $[1: n]$

Solution: slow, fast pointers.

To be noticed, the array companied with such properties forms a cycle if a graph is drawn with index and its corresponding value, for example, an array like $n = 5, [5, 5, 4, 3, 1, 2], \text{len} = 5 + 1, \forall a_i \in [1, 5]$, a circle exists in the path by starting with index $0 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow (5)$.

```
slow = 0, fast = 0
```

```

do
    slow = nums[slow]
    fast = nums[nums[fast]]
while(slow ≠ fast):

begin = 0
while(begin ≠ slow):
    begin = nums[begin]
    slow = nums[slow]
return begin

```

Solution 2: negatives flag

```

for( $i = 0 \rightarrow n - 1$ )
    index = abs(nums[i]) - 1
    if(nums[index] > 0): nums[index] *= -1
    else: return index + 1
return 0

```

Closet & farthest Pair in 2 Sorted Arrays

```

 $l = 0, r = 0, dif = 2^{31} - 1$ 
while( $l < a.length \ \& \ r < b.length$ )
    if( $abs(a_l - b_r) < dif$ ):  $dif = |a_l - b_r|$ 
    if( $a_l \leq b_r$ ):  $l = l + 1$ 
    else:  $r = r + 1$ 
return dif

```

Farthest

```

 $l = 0, r = 0, dif = -2^{32}$ 
while( $l < a.length \ \& \ r < b.length$ )
    if( $abs(a_l - b_r) > dif$ ):  $dif = |a_l - b_r|$ 
    if( $a_l \geq b_r$ ):  $l = l + 1$ 
    else:  $r = r + 1$ 
return dif

```

350. Intersection of Two arrays

Solution: like merge sort

Follow Up: one array, say $array_2$ is too large to fit into memory.

Solution: read $array_1$ into memory using hash-set, and read $array_2$ chunk by chunk into memory, and then find the intersection.

Follow Up: both two arrays are too large to fit into memory.

Solution: using external sort against two arrays (on disk), and each time read top 2 elements from 2 arrays.

75. Sort Colors Uber Didi Labs

for(int x : A):

if ($x = 0$): $A[p_2++] = 2, A[p_1++] = 1, A[p_0++] = 0$

elif ($x = 1$): $A[p_2++] = 2, A[p_1++] = 1$

else: $A[p_2++] = 2$

Time Complexity: suppose $N(x = 0) = N(x = 1) = N(x = 2) = \frac{N}{3}$

, thus, the total amount of operations will be $\frac{N}{3} * 3 + \frac{N}{3} * 2 + \frac{N}{3} = 2 * N$

Another Solution: use 3 delimiters: l, mid, r ,

$[0 \xrightarrow{0} (l - 1)]$; $[l \xrightarrow{1} (mid - 1)]$; $[mid \xrightarrow{\text{unknown}} (r - 1)]$; $[r \xrightarrow{2} (n - 1)]$

while($mid \leq r$)

if ($A[mid] = 0$): *swap*($A[mid++]$, $A[l++]$)

elif ($A[mid] = 1$): $mid := mid + 1$

else: *swap*($A[r--]$, $A[mid]$)

283. Move Zeros Facebook Bloomberg

Given an array \mathcal{A} , write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Intuition & Algorithm

```
void moveZeros(int[]  $\mathcal{A}$ ):
    int wall := 0;
    for( $i = 0 \rightarrow \text{len}(\mathcal{A}) - 1$ ):
        if( $\mathcal{A}[i] \neq 0$ ): swap( $\mathcal{A}, i, \text{wall}$ ), wall := wall + 1
```

260. Single Number III

There are only 2 numbers show up only once, while the remaining numbers show up twice.

Intuition & Algorithm

\oplus all numbers, suppose there two distinct number are x and y , then after \oplus all numbers, it gives out $\text{mask} = x \oplus y$. Since $x \neq y$, we can make use of the trick $e \oplus \sim(e - 1)$ which gives the right-most bit set to be "1".so one of them say x , s. t $(x \oplus \text{mask}) > 0$, and $(y \oplus \text{mask}) < 0$. For all the other numbers except (x, y) , we can use $\text{mask} \oplus \text{nums}_i > 0$ to exactly divide those number into two independent groups.

So, in group I, all elements $\text{nums}[i], (\mathcal{A}[i] \oplus \text{mask}) > 0$, while in group II, all elements $\mathcal{A}[j], (\mathcal{A}[j] \oplus \text{mask}) < 0$. Also, since each number except x occur twice in the group, we can do XOR again inside the group, and the result just gives out the first single number x , and the same thing happen in the second group.

```
xor = nums[0]
for( $i = 1 \rightarrow \mathcal{A}.\text{length} - 1$ ): xor = xor  $\oplus$   $\mathcal{A}[i]$ 
bit = xor  $\oplus$   $\sim(\text{xor} - 1)$ 
rs0 = 0, rs1 = 0
for(int x:  $\mathcal{A}$ ):
    if( $(x \& \text{bit}) > 0$ ): rs0 = rs0  $\oplus$  x
    else: rs1 = rs1  $\oplus$  x
return (rs0, rs1)
```

268. Missing Number

Solution I: XOR

$$e_1 = 1 \oplus 2 \oplus \dots \oplus n$$

$$e_2 = a_0 \oplus a_1 \oplus \dots \oplus a_{n-1}$$

$$result = e_1 \oplus e_2$$

Solution II: Sum

$$result = \frac{n * (n + 1)}{2} - \sum_0^{n-1} a[i]$$

Find Popular Element in an Array

$$popular = frequency(x) \geq \frac{n}{2}$$

190. Reverse Bits

For example, 1010000 → 0000101

for(*i* = 0 → 31):

$$mask = 1 \ll i$$

$$d = (n \& mask) \gg i$$

$$result = result * 2 + d$$

return result

Follow Up: $popular = frequency(x) \geq \frac{n}{4}$

$$if \left(count \left(A \left[\frac{n}{4} \right] \right) > \frac{n}{4} \right) : return A \left[\frac{n}{4} \right]$$

$$if \left(count \left(A \left[\frac{n}{2} \right] \right) > \frac{n}{4} \right) : return A \left[\frac{n}{2} \right]$$

$$if \left(count \left(A \left[\frac{3n}{4} \right] \right) > \frac{n}{4} \right) : return A \left[\frac{3n}{4} \right]$$

Find lower or upper bound of searched item in sorted array

int upper · bound(*target*, *A*):

$$l = 0, r = len(A) - 1, ub = -1$$

while(*l* ≤ *r*):

```

mid =  $\frac{l+r}{2}$ 
if(A[mid] = target):
    ub = mid, l = mid + 1
elif(A[mid] > target): r = mid - 1
else: l = mid + 1

return ub

```

Interview Q. Rotate Array Clock wisely in Place VMware

```

reverse(arr, 0, n - 1):
reverse(arr, 0, k - 1)
reverse(arr, k, n - 1)

```

231. Power of Two

```
return x & (x - 1) == 0
```

136. Single Number

```
re =  $a_0 \oplus a_1 \oplus \dots \oplus a_{n-1}$ 
```

137. Single Number II

All numbers show up 3 times while only one number which shows up only once

```

for(i = 31 → 0):
    mask = 1 << i, d = 0
    for(j = 0 → len(A) - 1):
        d := d + ((A[j] & mask) >> i)
    d := mod(d, 3)
    result := result * 2 + d

```

209. Minimum Size Subarray Sum Facebook

All integers in array are positives, find $sum[l:r] \geq target$

Intuition & Algorithm

Apply two pointers, and always maintain the window sum no exceeds target. When window sum exceeds or equals to target, try to minimize the window size by moving the left pointer.

```
int minSubArray( $\mathcal{S}$ , A):  
     $l = 0, r = 0, \text{minL} = 2^{31} - 1, \text{cur} = 0, \text{re} := \text{null}$   
    for( $r \leq n - 1$ )  
         $\text{cur} := \text{cur} + A[r]$   
        while( $l \leq r \ \& \ \text{cur} \geq \mathcal{S}$ ):  
            if( $\text{cur} - A[l] \geq \mathcal{S}$ ):  $\text{cur} := \text{cur} - A[l]$   
            else: break  
        if( $\text{cur} \geq \mathcal{S}$ ):  $\text{re} = (\text{re} = \text{nil}) \rightarrow (r - l + 1): \text{min}(\text{re}, r - l + 1)$   
    return  $\text{re} = \text{nil} \rightarrow 0: \text{re}$ 
```

Follow Up: Integers are not all positive.

Intuition & Algorithm

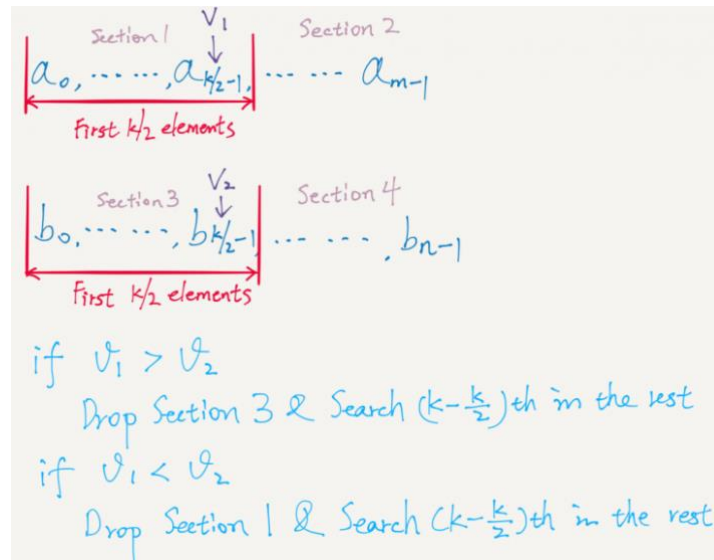
Hash Map

413. Arithmetic Slices

```
 $i = 0, \text{counter} = 0$   
while( $i < n - 1$ ):  
     $j = i + 1$   
    while( $j < n - 1 \ \& \ A_{j+1} - A_j == A_{i+1} - A_i$ ):  $j := j + 1$   
     $\text{le} = j - i + 1$   
     $\text{counter} += (\text{le} - 1) * (\text{le} - 2) / 2$   
     $i = j$   
return counter
```

4. Median of Two Sorted Arrays

Google **Amazon** **Microsoft** **Apple** **Zenefits** **Yahoo** **Adobe** **Dropbox**



Intuition & Algorithm

Compare the two middle elements of A & B , then drop the smaller half.

$find(A, i, B, j, k)$:

★ if one of them is empty, return the k^{th} element of another

$if(i \geq len(A)): return B[j + k - 1]$

$if(j \geq len(B)): return A[i + k - 1]$

★ in case $k = 1$, return the minimum of two arrays

$if(k = 1): return \min(A[i], B[j])$

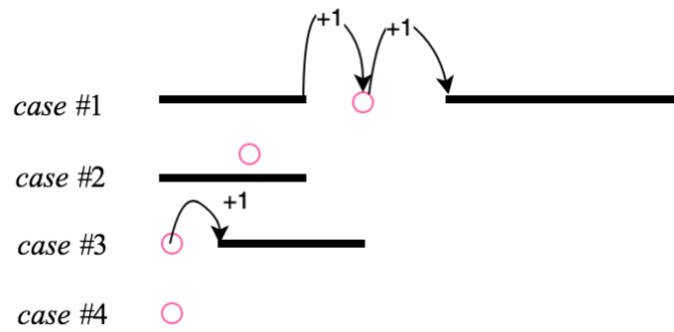
$int\ midA = i + \frac{k}{2} - 1 < len(A) \rightarrow A\left[i + \frac{k}{2} - 1\right]: \infty$

$int\ midB = j + \frac{k}{2} - 1 < len(B) \rightarrow B\left[j + \frac{k}{2} - 1\right]: \infty$

$if(midA < midB): return find\left(A, i + \frac{k}{2}, B, j, k - \frac{k}{2}\right)$

$else: return find\left(A, i, B, j + \frac{k}{2}, k - \frac{k}{2}\right)$

352. Data Stream as Disjoint Intervals



$map[start] \Rightarrow [start, end]$

$addNum(int\ val):$

$if(map[val] \neq nil): return$

$l := map.lowerKey(val)$

$r := map.higherKey(val)$

$if(l \neq null \ \& \ r \neq null \ \& \ map[l].end + 1 = val \ \& \ val + 1 = map[r].start)$

$map[l].end = map[r].end$

$map[r] = nil$ ★ *remove this interval*

$elif(l \neq null \ \& \ val \leq map[l].end + 1):$

$map[l].end = \max(map[l].end, val)$

$elif(r \neq null \ \& \ val = map[r].start - 1):$

$map[val] = map[r].end$

$map[r] = nil$

$else: map[val] = [val, val]$

348. Design Tic-Tac-Toe Google Microsoft

Two players play on a $n * n$ grid, and the diagram below suggests X wins

X	■	■	X	■	O	X	■	O	X	■	O	X	■	O	X	■	O	X	■	O		
■	■	■	→	■	■	■	→	■	■	■	→	■	O	■	→	■	O	■	→	O	O	■
■	■	■	■	■	■	■	■	X	■	■	X	X	■	X	X	■	X	X	X	X	X	X

Algorithm

$int\ row: [2 * n], column: [2 * n], diag: [2], xdiag: [2]$

$when\ place(x, y)\ is\ held\ by\ play\ i \in [0, 1]:$

$row[i, x] = row[i, x] + 1$

$column[i, y] = column[i, y] + 1$

$if(x = y): diag[i] = diag[i] + 1$

```

if( $x + y = n - 1$ ):  $xdiag[i] = xdiag[i] + 1$ 
if( $row[i, x] = n$  or  $column[i, y] = n$  &  $diag[i] = n$  &  $xdiag[i] = n$ ): return win

```

33. Search in Rotated Sorted Array

```

boolean find(arr, target):
    index ← findRotatedPosition(arr)
    if(bin · search(arr, 0, index - 1) & bin · search(arr, index, arr.length)):
        return true
    else: return false

int findRotatedPosition(A)
    l = 0, r = len(A) - 1
    while(l < r):
        mid ←  $\frac{l+r}{2}$ 
        if(A[mid] < A[r]): r := mid
        else: l := mid + 1
    return l

```

34. Search for a Range

Search for the lower & upper boundary of given value

```

int[] search · range(int[] A, int target):
    lb ← lower · boudnary(A, target)
    rb ← upper · boudnary(A, target)
    return new int[] {lb, rb}

int lower · boudnary(A, target)
    l = 0, r = len(A) - 1, lb = -1
    while(l < r):
        mid =  $\frac{l+r}{2}$ 
        if(A[mid] = target): lb = mid, r := r - 1
        elif(A[mid] < target): l := mid + 1
        else: r = mid - 1

```

35. Search insert position

```
l = 0, r = len(A) - 1
while(l < r):
    mid =  $\frac{l+r}{2}$ 
    if(target ≤ A[mid]): r = mid
    else: l = mid + 1
return l
```

240. Search a 2D Matrix II [Google](#) [Apple](#) [Amazon](#)

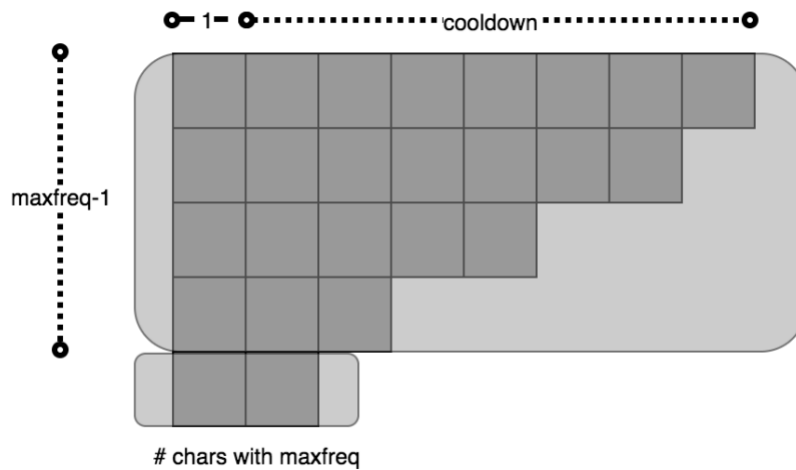
Intuition & Algorithm

Starting searching from the right-upper corner($0, \text{len}(\text{mat}[0])$)

Time Cost: $O(m + n)$

```
r = 0, c = matrix[0].length - 1
while(r < len(mat) & c ≥ 0):
    if(mat[r, c] = target): return true
    elif(target < mat[r, c]): c := c - 1
    else: r := r + 1
return false
```

621. Task Scheduler [Facebook](#)



for example, $(A_3, B_3, C_2, \text{cooldown} = 2) \rightarrow \begin{bmatrix} A & B & C \\ A & B & C \\ A & B & \end{bmatrix}$

Algorithm

$$\text{result} = (\max_{\text{freq}} - 1) * (\text{cooldown} + 1) + \sum_{i=0}^{25} (\text{freq}[\text{task}[i]] = \max_{\text{freq}})$$

767. Reorganize String Uber Google

Algorithm #1

priority queue

$\text{node} \begin{cases} \text{char} \\ \text{freq} \end{cases} \rightarrow \text{PriorityQueue}(\text{descending order of freq})$

Each time take two nodes *first* & *second* (if applicable) out of pq.

Then, if *first.freq* > 0 push it back to pq, the same for *second*

Corner case, if one char whose frequency $\text{freq} > \frac{(n+1)}{2}$, then return false

Solution II: count sort, quite like Q. 621

$\text{cell} \leftarrow \text{array}_{2d}(\text{unique char} \times \text{max} \cdot \text{freq}, \text{default} = \#)$

for($\text{freq} = \text{len}(\text{str}) \rightarrow 1$):

if($\text{map}[\text{freq}] \neq \text{nil}$):

list(char) *ls* $\leftarrow \text{map}[\text{freq}]$

for(char *ch* in *ls*):

for($i = 0 \rightarrow \text{freq}$): *cell*[*rowid*].add(*ch*)


```

        rowid += 1
i = 0, j = 0
for(; j < max · freq; ):
    for(; i < rowid; ):
        if (celli,j == '#'): break
        print(celli,j)

```

3. Longest Substring Without Repeating Characters

```

count = {ϕ}
l = 0
for(r = 0; r < s.length; )
    count[s[r]] := count[s[r]] + 1
    while(l ≤ r & count[s[r]] > 1):
        counts[s[l - -]] -= 1
    result = max(result, r - l + 1):
return result

```

389. Find the difference **Google**

Algorithm

```

count = [256]:int
for(i = 0; i < s.length; ): count[s[i]] := count[s[i]] + 1
for(i = 0; i < s.length; ): count[s[i]] := count[s[i]] - 1
for(c = a → z): if(count[c] ≠ 0): result = c

```

409. Longest Palindrome **Google**

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters.

$$result = \sum (x = letter_i.count, x \& 1 == 0) + \max\{x = letter_i.count, x \& 1 == 1\}$$

11. Container with Most Water **Bloomberg**

Given n vertical lines: $height_0 < height_1 < \dots < height_{n-1}$.

```

while(l ≤ r):
    if(heightl ≤ heightr): result = max(result, (r - l) * heightl), l := l + 1

```

*else: result = max{result, (r - l) * height_r}, r := r - 1*

15. 3Sum & 18. 4Sum

```
Arrays.sort(nums)
for(i = 0; i < n - 1;):
    l = i + 1, j = n - 1
    while(l < r):
        if(al + ar + ai = target):
            l := l + 1, r := r - 1
            while(l < r & a[l] = a[l - 1]): l := l + 1
            while(l < r & a[r] = a[r + 1]): r := r - 1
        elif(ai + al + ar > target): r := r - 1
        else: l := l + 1
```

454. 4Sum II

Solution: hash A + B first, and then check C + D in hash map

26. Remove Duplicates from Sorted Array Facebook Microsoft Bloomberg

```
wall = 1
for(i = 1 → len(A) - 1):
    if(i ≥ 1 & A[i] ≠ A[i - 1]):
        A[wall++] = A[i]
return wall
```

27. Remove Element

```
wall = 0
for(i = 0 → len(A) - 1):
    if(A[i] ≠ val): A[wall++] := A[i]
return wall
```

392. Is Subsequence Uber

```
for(ps = 0, pt = 0; ps < len(s) & pt < len(t);):
```

```

    if(s[ps] = t[pt]): pt += 1
    ps += 1
return pt == len(t)

```

Follow Up: Check a list of string where s_{i-1} is one prefix of s_i

```

for(String t: strList):
    for(i = ps, j = pt; i < len(s) & j < len(t);):
        if(si == tj): j ← j + 1
        i ← i + 1
    if(j == len(t)): ps = i, pt = j

```

204. Count Primes

```

is · prime[i] ← false, 0 ≤ i ≤ n - 1
for(i = 2; i < ⌊√n⌋);
    if(is · prime[i]) result += 1
    for(j = i + i; j < n;): is · prime[j] = false
return result

```

451. Sort Characters by Frequency **Bloomberg**

Solution I: Bucket Sort

Time Complexity: $O(n)$, Space Complexity: $O(n)$

```

List<char>[] freq = new ArrayList[n + 1]
map<char, int> char2freq
for(char c: str):
    counter[c] += 1
for(i = 0; i < 26;):
    freq[counter[i]].add((char)i)
for(i = n; i ≥ 0;):
    if(freq[i].notEmpty) result.addAll(freq[i])

```

692. Top K Frequent Words **Amazon** **Bloomberg** **Uber** **Yelp** **Pocket Gems**

Solution: quite like 451

Follow Up: Get K most frequent words from word stream. Bloomberg

The solution above (451) works well when the number of words is not that large, the strategy here is trading time for space. But in many cases, the number of words is very large, say 10^8 , while the number of different words is relatively small compare to it, say 10^4 , so it is possible to do word count with hash-map, but not possible to allocate such number of buckets for bucket sort. So, an optimized solution is Quick-Select, it takes $O(n)$ time and $O(k)$ space, where n is the number of words in text.

```
for(word w: text): map[w] = map[w] + 1
freq_array, 1d = map.values()
kth · freq = quick · select(freq, 0, len(freq) - 1, len(freq) - k)
for(Map.Entry<word, freq> entry: map.entrySet()):
    if(entry.getValue() ≥ kth · freq):
        result.add(entry.getKey())
    if(result.size() = k): break;
return result
```

76. Minimum Window Substring &

567. Permutation in String &

438. Find All Anagrams in a String Uber

```
minimum · window · substring(s, t):
    for(i = 0 → len(t) - 1):
        need · find[t[i]] := need · find[t[i]] + 1
    l = 0, e = 0
    for(r = 0; r < s.length; ):
        has · found[s[r]] := has · found[s[r]] + 1;
        if(has · found[s[r]] ≤ need · find[s[r]]): e := e + 1
        if(e = len(target)):
            while(need · find[s[l]] = 0 || has · found[s[l]] > need · find[s[l]]):
                if(has · found[s[l]] > need · find[s[l]]):
                    has · found[s[l]] := has · found[s[l]] - 1
                l := l + 1
    return sl,r
```

138. Copy List with Random Pointer Amazon

```
for(Node nd: list):
    nd · copy ← make · copy(nd)
    map(nd) → nd · copy
for(Node nd: list):
    map[nd].next = map[nd.next]
    map[nd].randomNext = map[nd.randomNext]
```

133. Clone Graph Google Facebook Uber Pocket Gems

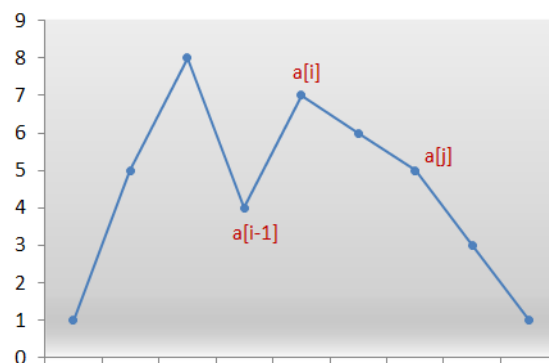
do bfs on graph, and make copy for each of node:

```
for(g: graph.nodes):
    list⟨node⟩ adjs ← g.adjacents
    for(node nd: adjs):
        map[g].adjacentlist.add(map[nd])
```

31. Next Permutation Facebook Google Microsoft Amazon Samsung LinkedIn

Generate next slightly greater number with same digits

Intuition & Algorithm



Find first ℓ s.t. $\mathcal{A}[\ell] < \mathcal{A}[\ell+1]$ from rightmost, then find first r in range of $[\ell+1, n-1]$ s.t. $\mathcal{A}[\ell] < \mathcal{A}[r]$ from right-most. Next, swap $(\mathcal{A}[\ell], \mathcal{A}[r])$. Finally, sort $\mathcal{A}[\ell+1: n-1]$

Next Palindromic Number

Consider three conditions:

-
1. if $a = [9, 9, \dots, 9]$, return $[1, 0, \dots, 0, 1]$
 2. else if a is palindromic except for case 1, increase $a \left\lceil \frac{n}{2} \right\rceil$ and $a \left\lfloor \frac{n}{2} - 1 \right\rfloor$ by 1
 3. else, copy left half $a \left[0 : \frac{n}{2} - 1 \right]$ to the right half $a \left[\frac{n}{2} : n - 1 \right]$, get a^\times

if $a^\times < a$, then $a \left\lceil \frac{n}{2} \right\rceil += 1$, $a \left\lfloor \frac{n}{2} - 1 \right\rfloor += 1$, otherwise, do nothing
-

80. Remove Duplicates from Sorted Array II Facebook

Given a sorted array *nums*, remove the duplicates in-place such that duplicates appeared at most twice and return the new length.

Intuition & Algorithm

```
wall = 1, count := 0
for(i = 1 → len(A) - 1):
    count := (A[i] = A[i - 1]) → count + 1: 0
    if(count < 2):
        A[wall++] := A[i]
return wall
```

442. Find all Duplicates in an Array Pocket Gems

Intuition & Algorithm

Negative Labeling, since $\mathcal{A}[i] \in [1, n]$, we can label $\mathcal{A}[|\mathcal{A}[i]| - 1] \star = -1$

```
re := [], n := len(A)
for(i = 0 → n - 1):
    int index := |A[i]| - 1
    if(index < n): re.add(index + 1)
    else: A[index] ⋆ = -1
return re
```

41. First Missing Positive

448. Find All numbers disappeared in an array

Solution: firstly swap all positive integers to the left, for example, $[3, 4, -1, 1] \rightarrow [3, 4, 1, -1]$, then apply “negative mark” algorithm.

```
for(i = 0; i < len(arr) - 1):
    if(arr[i] > 0): arr[le + +] = arr[i]
for(i = 0 → le - 1):
    idx = |arri| - 1
    if(idx < le & arr[idx] > 0): arr[idx] := arr[idx] * -1
for(i = 0 → le - 1):
    if(arr[i] > 0): return (i + 1)
```

Common Chars in n strings

```
first = 0
for(char c: s[0]): first ← first | (1 << (c - a))
for(i = 1 → n - 1):
    second = 0
    for(char c: s[i]): second := second | (1 << (c - a))
    first = first & second
for(i = 0 → 26):
    if((first & (1 << i)) >> i == 1): re := re + (char)(i + a)
```

318. Maximum Product of Word Lengths

Find the maximum value of $length(word[i]) * length(word[j])$ where the two words do not share common letters.

Intuition & Algorithm

encode string using binary format, like $encode = encode | (1 \ll (str[i] - 'a'))$, $\forall i \in [0, le - 1]$, so checking if two strings share the same characters just need to check $estr[i] \& estr[j] = 0$ or not (in $O(1)$ time). Thus, the overall time complexity is $O(m * n)$

BT & BST & Iterator

302. Smallest Rectangle Enclosing Black Pixels [Google](#)

An image is represented by a binary matrix with 0 as a white pixel and 1 as a black pixel. The black pixels are connected, i.e., there is only one black region. Pixels are connected horizontally and vertically. Given the location (x, y) of one of the black pixels, return the area of the smallest (axis-aligned) rectangle that encloses all black pixels.

Intuition & Algorithm

Apply B-Search to find the left, right, top & bottom boundary of 1.

Time Costs $O(m \star \log(n) + n \star \log(m))$

```
int minArea(int[ ][ ] image, int x, int y):
    top := searchVertical(image, 0, x, searchTop = True)
    bottom := searchVertical(image, x, len(image) - 1, searchTop = False)
    left := searchHorizontal(image, 0, y, searchLeft = True)
    right := searchHorizontal(image, y, len(image[0]) - 1, searchLeft = False)
    return (bottom - top + 1) * (right - left + 1)

int searchVertical(int[ ][ ] image, int top, int bottom, bool searchTop):
    re := searchTop = True → bottom: top
    while(top ≤ bottom):
        mid :=  $\frac{top+bottom}{2}$ 
        s := 0
        for(j = 0 → len(image[mid]) - 1): s := s + image[mid, j]
        if(s ≥ 1):
            re := mid
            if(searchTop): bottom := mid - 1
            else: top := mid + 1
        else:
            if(searchTop): top := mid + 1
            else: bottom := mid - 1
    return re

int searchHorizontal(int[ ][ ] image, int left, int right, bool searchLeft):
    re := searchLeft = True → right: left
```



```

while(left ≤ right):
    mid :=  $\frac{left+right}{2}$ 
    s := 0
    for(i = 0 → len(image) - 1): s := s + image[i, mid]
    if(s ≥ 1):
        re := mid
        if(searchLeft): right := mid - 1
        else: left := mid + 1
    else:
        if(searchLeft): left := mid + 1
        else: right := mid - 1
return re

```

235. LCA of BST & BT

236. LCA of BT

```

lca · bst(TreeNode o, int lb, int rb):
    if(o = nil): return null
    if(lb ≤ o.val ≤ rb): return o.val
    elif(lb > o.val): return lca · bst(o.right, lb, rb)
    else: return lca · bst(o.left, lb, rb);

```

```

lca · bt(TreeNode o, TreeNode p, TreeNode q):
    if(o is null): return null
    if(o = p): return p
    if(o = q): return q
    lo ← lca · bt(o, p, q)
    ro ← lca · bt(o, p, q)
    if(lo ≠ nil & ro ≠ nil): return o;
    if(lo = nil & ro ≠ nil): return ro
    if(lo ≠ nil & ro = nil): return lo
return null

```

Interview Q. **2nd Largest Element in BST** NewsOnChat

Time Complexity: $O(h + k)$,

h: because it traverse from root to the left · most item

k: because it won't stop until finding the k^{st} · item

```
void  $k^{st}$  · largestItem(TreeNode root, Count c):  
    if(root == null or c.val  $\geq$  k):return;  
     $k^{st}$  · largestNode(node.right, c) ★ count # nodes of right subtree  
    c.times = c.times + 1  
    if(c.times == k):  
        print( $k^{st}$  · element is root.val)  
        return  
     $k^{st}$  · largestNode(root.left, c)
```

652. Find Duplicate Subtrees Google

Solution: Preorder + Hash-map

Since same tree structure correspond to the same preorder serialization. I use a hash-map to map the tree preorder serialization to the occurrence times. Once one serialization shows up 2 or more than twice, I add it to the result.

```
String serialize(TreeNode o, Mapstr→int map, List<TreeNode> result):  
    if(o == null):return "#"  
    se = (o.val) + , + serialize(o.left, map, result) + , + serialize(o.right, map, result)  
    if(map.getOrDefault(se, 0) == 1):result.add(o)  
    map[se] = map[se] + 1  
    return se  
main():  
    serialize(o, map, result)  
    return result
```

173. Binary Search Tree Iterator Google Facebook Microsoft LinkedIn

Intuition & Algorithm

Can Refer to <https://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion/>

```
stack(type = TreeNode)  
BSTIterator(TreeNode root):
```

```

node = root
while(node ≠ null):
    stack.push(node)
    node = node.left
boolean hasNext():
    return !stack.isEmpty()
int next():
    node = stack.pop()
    if(node.right ≠ null):
        node = node.right
        while(node ≠ null):
            stack.push(node)
            node = node.left
    return node.val

```

98. Validate Binary Search Tree [Facebook](#) [Microsoft](#) [Amazon](#) [Bloomberg](#)

```

min = -231, max = 231 - 1
check(TreeNode o, int min, int max):
    if(o = nil): return true
    if(o.val ∈ [min, max] & check(o.l, min, o.val) & check(o.r, o.val, max))
        return true
    return false

```

99. Recover Binary Search Tree

Two nodes are mistakenly swapped in BST, asked to find them and swap back

```

void recoverTree(TreeNode R):
    prev := nil, 1st = nil, 2nd = nil
    stack := {ϕ}
    TreeNode p := R
    while(p ≠ nil): stack.push(p), p := p.left
    while(stack.size > 0):
        curr := stack.pop
        if(prev ≠ nil & prev.val ≥ curr.val):

```

```

    if( $1^{st} \neq nil$ ):  $1^{st} = prev$ 
     $2^{nd} = curr$ 
     $prev := curr$ 
    if( $curr.right \neq nil$ ):
         $r := curr.right$ 
        while( $r \neq nil$ ):  $stack.push(r), r := r.left$ 
    swap( $1^{st}, 2^{nd}$ )

```

285. In-order Successor in BST [Facebook](#) [Microsoft](#) [Pocket Gems](#)

```

TreeNode suc := nil
while( $root \neq nil$ ):
    if( $p.val \geq root.val$ ):  $root := root.right$ 
    else:  $suc = root, root := root.left$ 
return suc

```

572. Subtree of Another Tree [Facebook](#) [eBay](#)

Given two non-empty binary trees s and t , check whether tree t has exactly the same structure and node values with a subtree of s

Intuition & Algorithm #1

Two BTs has the same structure & node values **if & only-if** two BTs share the same pre-order serialized representation.

Time Costs: $O(m^2 + n^2 + m * n), m = size(s), n = size(t)$

Intuition & Algorithm #2

For each of node in tree s , check if every subtree is the same as t .

Time Costs: $O(m * n), m = size(s), n = size(t)$

145. Binary Tree Post-Order Traversal

Intuition & Algorithm

```

 $s_1 := \{\phi\}, s_2 := \{\phi\}$ 
 $s_1.push(root)$ 
while( $s_1.size > 0$ ):
    Node  $node := s_1.pop$ 
     $s_2.push(node)$ 
    if( $node.left \neq nil$ ):  $s_1.push(node.left)$ 
    if( $node.right \neq nil$ ):  $s_1.push(node.right)$ 
Postorder :=  $s_2.top \rightarrow s_2.bottom$ 

```

199. Binary Tree Right Side View NewsOnChat

```

rightView =  $\{\phi\}$ 
Q = {root}
while(Q isn't empty):
    n = Q.size
    for(i = 0  $\rightarrow$  n - 1):
        node = Q.poll()
        if(i = n - 1): rightView.offer(node.val)
        if(node.left  $\neq$  null): Q.offer(node.left)
        if(node.right  $\neq$  null): Q.offer(node.right)
return rightView

```

226. Invert Binary Tree

Solution: reverse left-sub tree & right-sub tree first, then swap left & right pointer.

```

invert(TreeNode o):
    if(o is null): return null
    TreeNode lc  $\leftarrow$  invert(o.left)
    TreeNode rc  $\leftarrow$  invert(o.right)
    temp  $\leftarrow$  o.left
    o.left  $\leftarrow$  o.right
    o.right  $\leftarrow$  temp
    return o

```

100. Same Tree

```
same(TreeNode p,TreeNode q):
    if(p = null):return q = null
    if(p.val = q.val & same(p.left,q.left) & same(p.right,q.right)):
        return true
    return false
```

101. Symmetric Tree

```
symmetric(TreeNode p,TreeNode q):
    if(p = null):return q = null
    if(q = null):return p = null
    if(p.val = q.val & symmetric(p.l,q.r)& symmetric(p.r,q.l)):
        return true
    return false
```

297. Serialize & De-serialize BT

Google **Facebook** **Microsoft** **Amazon** **Bloomberg** **Uber** **LinkedIn** **Yahoo**

Solution: preorder

Time Complexity: $O(n)$, Space Cost: $16 \star 2n = 32n$ bits, there are $2n$ nodes(including null node), each node takes 16 bits for storage.

```
void serialize(TreeNode root,StringBuffer sb)
    if(root is nil):sb.append("#"),return;
    sb.append(root.val + ",")
    serialize(root.left,sb)
    serialize(root.right,sb)
```

TreeNode de · serialize(list<str> nodes)

```
if(nodes.empty || nodes.peek == '#'):
    if(nodes.not · empty):nodes.pollFirst
    return null
TreeNode root ← new TreeNode(int(nodes.pollFirst))
root.left ← de · serialize(nodes)
root.right ← de · serialize(nodes)
```

return root

Follow Up: Succinct serialization & de-serialization **Google**

Use two lists, one is called structure, another is called data, both are stored in pre-order.

void serialize(root, structure: list, data: list)

```
if (root is nil):
    structure.add(0), return
structure.add(1)
data.add(root.val)
serialize(root.left, structure, data)
serialize(root.right, structure, data)
```

TreeNode de · serialize(structure, data):

```
if (nodes.empty): return nil
bool b = structure.peekFirst
if (b == 0): structure.pollFirst
else:
    TreeNode root ← new TreeNode(int(data.pollFirst))
    root.left ← de · serialize(structure, data)
    root.right ← de · serialize(structure, data)
    return root
return null
```

102. Binary Tree Level Order Traversal & II (107)

103. Binary Tree Zigzag Level Order Traversal

Solution: BFS

144. Binary Tree Preorder Traversal & Post-order (145)

450. Delete Node in BST

TreeNode deleteNode(TreeNode o, key):

```
if (o is null): return null
o.left = deleteNode(o.left, key)
```

```

o.right = deleteNode(o.right, key)
if(o.val ≠ key): return o
else:
    p = o.right;
    if(p is null): return o.left
    while(p.left is not null): p = p.left
    swap(p, o)
    o.right = deleteNode(o.right, key)
    return o

```

104. Maximum Depth of Binary Tree

Solution: $depth(o) = \max \{depth(o.left), depth(o.right)\} + 1$

110. Balanced BT Bloomberg

```

depth(TreeNode o):
    if(o is null): return 0
    lheight := depth(o.left)
    rheight := depth(o.right)
    if(lheight = -1 or rheight = -1 or abs(lheight - rheight) > 1):
        return -1
    return max(lheight, rheight) + 1
balanced = depth(o) = -1

```

Interview Q. Preorder to BST

```

index = 0, min = -231, max = 231 - 1
TreeNode pre2BST(preorder(type=array, 1d), min, max):
    if(index > preorder.length): return null
    if(preorder[index] ∈ [min, max]):
        root = new TreeNode(preorder[index++])
        root.left = pre2BST(preorder, min, root.val)
        root.right = pre2BST(preorder, root.val, max)

```


return root

323. Number of Connected Components in an Undirected Graph

Solution: union-find

```
CC = N(vertices)
for(edge e: edges):
    o1 = find(e.from)
    o2 = find(e.to)
    if(o1 ≠ o2):
        union(o1, o2)
    CC = CC - 1
```

543. Diameter of Binary Tree [Google](#) [Facebook](#)

Solution: pre-calculate height for each node in BT, then store the $(node, height)$ mapping into hash-map, finally calculate the diameter like this:

$$\mathcal{D}(o) = (h[o.left] + h[o.right]), h[nil] := 0$$
$$re := \max(\mathcal{D}(o), \mathcal{D}(o.left), \mathcal{D}(o.right))$$

Longest Path in N_{ary} Tree [Jingchi.ai](#)

Solution: pre-calculate height for each node in BT, then store the $(node, height)$ mapping into hash-map, finally

```
height: map(Node, height)
int longest · path(Node root):
    if(root == null): return 0
    childNum ← root.chlds.size()
    result = 0
    for(i = 0 → childNum - 1): ★ find LP in root's subtrees
        result = max(result, longest · path(root.chlds[i]))
    mx1 = null, mx2 = null
    for(i = 0 → childNum - 1):
        h ← height[root.chlds[i]]
        if(mx1 = null || h > mx1):
            mx2 = mx1, mx1 = h
```

```

    elif(mx2 = null || h > mx2):
        mx2 = h
    mx1 = mx1 = nil → 0:mx1
    mx2 = mx2 = nil → 0:mx2
    re = max(re, mx1 + mx2 + 1)
return re

int depth(Node root):* height[root] = max(height[child[i]] + 1, ∀i ∈ [0, size(root.chlds)])
    if(root = null): return 0
    childNum ← root.chlds.size()
    maxh = 0
    for(i = 0 → childNum - 1):
        h = depth(root.chlds[i])
        maxh = max(maxh, h)
    height[root] = maxh + 1
    return maxh + 1

main():
    Node root = buildTree()
    depth(root)
    result = longest · path(root)

```

549. Binary Tree Longest Consecutive Sequence II [Google](#)

Follow Up: N-ary Tree Longest Consecutive Sequence

incr: map(Node ⇒ length of longest increasing consecutive sequence end up with this node)
 decr: map(Node ⇒ length of longest decreasing consecutive sequence end up with this node)

```

int longest · consecutive · sequence(Node root):
    if(root == null): return 0
    childNum ← root.chlds.size
    result = 0
    for(i = 0 → childNum - 1):
        result = max(result, longest · consecutive · sequence(root.chlds[i]))
    maxhincr = null, maxhdecr = null

```

```

for( $i = 0 \rightarrow childNum - 1$ ):
    if( $root.child[i].val + 1 = root.val$ ):  $h_{incr} := incr[root.chlds[i]]$ 
    if( $root.child[i].val - 1 = root.val$ ):  $h_{decr} := decr[root.chlds[i]]$ 
    if( $maxh_{incr} = null \parallel h_{incr} > maxh_{incr}$ ):  $maxh_{incr} = h_{incr}$ 
    if( $maxh_{decr} = null \parallel h_{decr} > maxh_{decr}$ ):  $maxh_{decr} = h_{decr}$ 

```

```

 $maxh_{incr} = (maxh_{incr} = null)? 1: maxh_{incr} + 1$ 
 $maxh_{decr} = (maxh_{decr} = null)? 0: maxh_{decr} + 1$ 
 $result = \max(result, maxh_{incr} + maxh_{decr} - 1)$ 

```

```
return result
```

```

int getIncrDepth(Node root):
    if( $root == null$ ): return 0
     $childNum \leftarrow root.chlds.size$ 
     $maxh_{incr} = 0$ 
    for( $i = 0 \rightarrow childNum - 1$ ):
         $h = getIncrDepth(root.chlds[i])$ 
        if( $root.child[i].val + 1 == root.val$ ):  $maxh_{incr} = \max(maxh_{incr}, h)$ 
     $incr[root] = maxh_{incr} + 1$ 
    return  $maxh_{incr} + 1$ 

```

```

int getDecrDepth(Node root):
    if( $root == null$ ): return 0
     $childNum \leftarrow root.chlds.size$ 
     $maxh_{decr} = 0$ 
    for( $i = 0 \rightarrow childNum - 1$ ):
         $h = getDecrDepth(root.chlds[i])$ 
        if( $root.child[i].val + 1 == root.val$ ):  $maxh_{decr} \leftarrow \max(maxh_{decr}, h)$ 
     $decr[root] = maxh_{decr} + 1$ 
    return  $maxh_{decr} + 1$ 

```

Largest Complete Tree ForUsAll

the height of complete tree is: $h = \min(\text{height}(o.l), \text{height}(o.r)) + 1$, to be noticed, it takes the minimal one of left subtree & right subtree plus 1 as its complete tree height, which is different than our node's height computation where $node's\ height = \max(lh, rh) + 1$

the amount of nodes in complete tree with height h is: $size = 1 \ll h - 1$

height · complete(Node o):

if(o is null): return 0

lh = height · complete(o.left)

rh = height · complete(o.right)

$h \cdot complete = \min(lh, rh) + 1$

result = max(result, $h \cdot complete$)

return $h \cdot complete$

$size\ of\ complete\ tree = 1 \ll result - 1$

105. Construct BT from Preorder & In-order Traversal

106. Construct BT from in-order & Post-order Traversal

map(inorder[i] → i)

TreeNode buildBT(preorder, l, r, inorder, p, q)

if(l > r || p > q): return null

index = map[preorder[l]]

root = TreeNode(inorder[pos])

left · size = index - p, right · size = q - index

o.left = build(preorder, l + 1, l + left · size, inorder, p, index - 1)

o.right = build(preorder, l + left · size + 1, r, inorder, index + 1, q)

108. Convert Sorted Array to BST

TreeNode sortedArray2BST(int[] A, int l, int r):

if(l > r): return null;

$m := \frac{l+r}{2}$

TreeNode root ← new TreeNode(A[m])

root.left := sortedArray2BST(A, l, m - 1)

root.right := sortedArray2BST(A, m + 1, r)

return root

109. Convert Sorted List to BST Zenefits

n := 0, p := head

```

while(p ≠ nil): n := n + 1, p := p.next
list2bst(int n):
    if(n ≤ 0 or head = nil): return nil

    l = list2bst( $\frac{n}{2}$ ) ★ build the left · subtree, head has been moved by  $\frac{n}{2}$  steps

    root = TreeNode(head.val)
    root.left = l
    head = head.next

    root.right = list2bst( $n - 1 - \frac{n}{2}$ ) ★ build the right · subtree

    return root

```

255. Verify Preorder Sequence in Binary Search Tree

Algorithm #1

```

verify · preorder(preorder = [array])
    if(root = null): return true
    Stack<int> st = {ϕ}, low = -231
    for(int x: preorder):
        if(x < low): return false
        while(st.size > 0 & x > st.peek()): low = st.pop()
        st.push(x)
    return true

```

Algorithm #2

```

index := 0, size := 0
bool verifyPreorder(int[ ] preorder):
    n := len(preorder)
    mi := -231, mx := 231 - 1
    TreeNode R := build(preorder, mi, mx)
    return size == n

```

```

TreeNode build(int[ ] preorder, int mi, int mx):
    if(index == len(preorder)): return nil
    if(preorder[index] ∈ [mi, mx]):
        R := TreeNode(preorder[index])
        index := index + 1
        R.l := build(preorder, mi, R.val)
        R.r := build(preorder, R.val, mx)
        return R
    return null

```

114. Flatten Binary Tree to Linked List

```

flatten(root):
    if(root is null): return
    flatten(root.left):
    if(prev == nil): head := root
    else: prev.right := root, root.left := prev
    flatten(root.right):

```

116. Populating Next Right Pointers in each Node

117. Populating Next Right Pointers in each Node II Microsoft

```

void connect(Node o):
    if(o is null): return
    o.nextR = null
    while(o != null):
        p = o
        while(p != null):
            if(p.left != null):
                if(p.right != null): p.left.nextR = p.right
                else: p.left.nextR = find · nephew(p)
            if(p.right != nil): p.right.nextR = find · nephew(p)
            p = p.nextR
        if(o.left != nil): o = o.left

```

```

elif(o.right ≠ nil): o = o.right
else: o = find · nephew(o)

```

Node find · nephew(p: Node):

```

q = p.nextR
while(q ≠ nil):
    if(q.left ≠ nil): return q.left
    if(q.right ≠ nil): return q.right
    q = q.nextR
return nil

```

Queue & Stack & Greedy & Heap

155. Minimum Stack

Follow Up: Maximum & Minimum Stack ForUsAll

solution: maintain three variables: *stack*, *min* & *max*

Algorithm

```

push(x):
    if(stack.size = 0): max = x, stack.push(0)
    else: stack.push(x - maxcurrent), maxcurrent = max(x, maxcurrent),
top():
    if(stack.peek ≤ 0): ** x < maxprev, maxcurrent = maxprev
    return x = stack.peek + maxcurrent(max)
    elif(stack.peek > 0): ** x > maxprev, maxcurrent = x
    return maxcurrent(max)
pop():
    result = top()
    if(stack.peek ≤ 0): ** x < maxprev, maxcurrent = maxprev
    *****
    elif(stack.peek > 0): ** x > maxprev, maxcurrent = x, maxcurrent - dif = maxprev
    max = max - stack.peek
    stack.pop()

```

```

    return result
get · max():
    return max

```

232. Implement Queue using Stacks **Microsoft** **Bloomberg**

solution: apply two stacks, one is called “in”, another is called “out”.

When new element comes in, push it in the “in” stack, in case of the pop operation of queue, there are two cases to be handled, if the “out” stack is not empty, then pop the top one, otherwise, push all the elements in the “in” stack and pop the top one of “out” stack.

346. Moving Average from Data Stream **Uber**

```

Deque dq ← {ϕ}
class MovingAvg:
    private int size, window · sum
    public MovingAvg(int size):
        size = sz, window · sum = 0
    public double next(int val):
        dq.addLast(val);
        window · sum += val
        if (dq.size > size):
            dq.pollFirst()
            window · sum -= val
    return  $\frac{\text{window} \cdot \text{sum} \cdot 1.0}{\text{dq.size}}$ 

```

341. Flatten Nested List Iterator **Uber**

Solution: DFS / Stack

```

dfs(List<NestedInteger> list, int idx):
    if (idx == list.size): return;

```

```

if(list[idx].isNumber):result.add(list[idx])
else:dfs(list[idx],0),dfs(list,idx + 1)

```

Stack Solution:

```

List flatten(List<NestedInteger> list):
    List result ← {ϕ}
    for(NestedInteger each: list): stack.push(each)
    while(stack isn't empty):
        NestedInteger top ← stack.pop()
        if(top is a number):
            result.addFirst(top)
        else:
            for(NestedInteger ne: top): stack.push(ne)
    return result

```

225. Implement Stack using Queues

solution:

```

void push(int x):
    queue.offer(x)
    for(i = 0; i < queue.size - 1; ): queue.offer(queue.poll())

```

321. Create Maximum Number Google

```

int[ ] maxNumber(int[ ] nums1,int[ ] nums2,int k):
    m := len(nums1),n := len(nums2),re = [k]
    for(len = 0 → k):
        if(len < m & k - len < n):
            int[ ] A := maxNumberOfKdigits(nums1,len)
            int[ ] B := maxNumberOfKdigits(nums2,k - len)
            re := max(re,merge(A,B))
    return re

```

```

int[ ] maxNumberOfKdigits(A,len):

```

```

re = [len], Stack stack
for(i = 0 → len(A) - 1):
    remains := len(A) - i - 1 ★ [i + 1, len(A) - 1]
    while(A[i] > A[stack.peek] & remains + stack.size ≥ k): stack.pop
    if(stack.size < k) stack.push(i)
e := len - 1
while(~stack.empty): re[e--] := A[stack.pop]
return re

```

```

int[ ] merge(A, B):
    i = 0, j = 0, re = [len(A) + len(B)]
    while(i < len(A) & j < len(B)):
        if(compare(A, i, B, j)): re[e++] := A[i++]
        else: re[e++] = B[j++]
    while(i < len(A)): re[e++] = A[i++]
    while(j < len(B)): re[e++] = B[j++]
    return re

```

```

int compare(A, i, B, j):
    while(i < len(A) & j < len(B) & A[i] = B[j]): i := i + 1, j := j + 1
    if(i < len(A) & j < len(B)): return A[i] - B[j]
    elif(i < len(A) & j = len(B)): return 1
    else: return -1

```

```

int max(int[ ] A, int[ ] B):
    for(i = 0 → len(A)):
        if(A[i] > B[j]): return A
        else: return B
    return A

```

316. Remove Duplicate Letter Google

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appears once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

remove · duplicates(s):

distincts := 0

for(i = 0 → len(s) − 1):

remains[s[i]] := remains[s[i]] + 1

if(remain[s[i]] = 1): distincts := distincts + 1

stack[int] st: ascending order

used: hashset

for(i = 0 → len(str) − 1):

counts[strs[i]] := counts[strs[i]] − 1

** if current char has not shown up in the stack and the remainings > 0*

while(∼st.empty & !used[s[i]] & s[stack.peek] > s[i] & remains[stack.peek] > 0):

used[stack.pop] := false

if(stack.size < distincts):

stack.push(i)

onstack[s[i]] = true

re = ""

while(∼st.empty): re := re + s[st.pop]

return reverse(s)

402. Remove K Digits [Google](#) [Snapchat](#)

Given a non-negative integer num represented as a string, remove k digits from the number so that the new number is the smallest possible.

Intuition & Algorithm

The same idea as **316. Remove Duplicate Letter**.

224. Basic Calculator

227. Basic Calculator II [Uber](#)

** or / > +, −*
) > (

II. only +, −, *, /

i ← 0, le ← exp.length

```

while(i < le & exp[i] == \s) i ← i + 1
if(i == le): return 0
first ← 0
while(i < le & exp[i] ∈ [0,9]): first ← first * 10 + (exp[i + +] - '0')
if(i == le): return first
result = first, update = first
while(i < le):
    while(i < le & exp[i] == \s): i ← i + 1
    if(i == le): break
    nb ← 0, * nb is short for number
    if(exp[i] is ' + '):
        while(i < le & exp[i] isn't a digit): i := i + 1
        while(i < le & exp[i] is a digit): nb ← nb * 10 + (exp[i + +] - '0')
        result ← result + nb, update := +nb
    elif(exp[i] is ' - '):
        while(i < le & exp[i] isn't a digit): i ← i + 1
        while(i < le & exp[i] is a digit): nb ← nb * 10 + (exp[i + +] - '0')
        result ← result - nb, update := -nb
    elif(exp[i] is ' * '):
        while(i < le & exp[i] isn't a digit): i ← i + 1
        while(i < le & exp[i] is a digit): nb ← nb * 10 + (exp[i + +] - '0')
        result := result - update + update * nb, update := update * nb
    elif(exp[i] is ' / '):
        while(i < le & exp[i] isn't a digit): i ← i + 1
        while(i < le & exp[i] is a digit): nb ← nb * 10 + (exp[i + +] - '0')
        result := result - update +  $\frac{update}{nb}$ , update :=  $\frac{update}{nb}$ 
return result

```

150. Evaluate Reverse Polish Notation

20. Valid Parentheses

```

for(i = 0; i < s.length; ):
    if(stack.notempty & stack.peek == ' (' & s[i] == ')'): stack.pop()
    else: stack.push(s[i])

```

Interview Q: Heap Sort Didi Labs

Build the max-heap first, then each time shrink the array by swapping the last element with the first one, and reconstruct the max-heap again.

```
for ( $i = \frac{n}{2} - 1; i \geq 0;$ ):
```

```
    * build the maxheap from bottom to top
```

```
    heapify(array = A, size = n, index = i)
```

```
for( $i = n - 1; i \geq 0;$ ):
```

```
    swap(A, i, 0)
```

```
    heapify(A, size = i, 0)
```

```
void heapify(A, size, i):
```

```
    lchild  $\leftarrow 2i + 1$ , rchild  $\leftarrow 2i + 2$ , largest  $\leftarrow i$ 
```

```
    if(lchild < size & A[lchild] > A[largest]): largest  $\leftarrow$  lchild
```

```
    if(rchild < size & A[rchild] > A[largest]): largest  $\leftarrow$  rchild
```

```
    if(largest  $\neq$  i):
```

```
        swap(A, largest, i)
```

```
        heapify(A, size, largest)
```

295. Find Median from Data Stream

Intuition & Algorithm #1

Maintain two heaps, one max-heap, storing the smaller half, and one min-heap, storing the larger half.

Solution I: max-heap, store the smaller half, min-heap, for the larger half.

```
void add(x):
```

```
    if(min.empty ||  $x \geq$  min.peek):
```

```
        min.offer(x)
```

```
        if(min.size - max.size > 1): max.offer(min.poll)
```

```
    elif(max.empty ||  $x \leq$  max.peek):
```

```
        max.offer(x)
```

```
        if(max.size - min.size > 1): min.offer(max.poll)
```

```
    else:
```

```

min.offer(x)
if(min.size - max.size > 1): max.offer(min.poll)

```

```

int getMedian():
    if(min.size == max.size) return (min.peek() + max.peek()) / 2.0
    else if(min.size > max.size) return min.peek()
    else: return max.peek()

```

Intuition & Algorithm #2

Array-list for storing, quick-select for fetching the median.

239. Sliding Window Maximum

```

for(i = 0; i < n; ++i):
    while(q.size > 0 & a[i] > a[q.tail]): q.pollLast()
    q.addLast(i)
    while(q.size > 0 & q.tail - q.head + 1 > k): q.pollFirst()
    if(i ≥ k - 1): result.add(a[q.head])

```

Follow Up: Sliding Window Max Difference

Solution: it just needs an another deque maintaining an ascending order of integers.

```

for(i = 0; i < n; ++i):
    while(qmx.notEmpty() & a[i] > a[qmx.tail]): qmx.pollLast()
    while(qmi.notEmpty() & a[i] < a[qmi.tail]): qmi.pollLast()
    qmx.addLast(i)
    qmi.addLast(i)
    while(qmx.notEmpty() & qmx.tail - qmx.head + 1 > k): qmx.pollFirst()
    while(qmi.notEmpty() & qmi.tail - qmi.head + 1 > k): qmi.pollFirst()
    if(i ≥ k - 1): result.add(a[qmx.head] - a[qmi.head])

```

407. Trapping Rain Water II

```

for(boundary cell: matrix):
    heap.offer(cell{x, y, height})

```

```

    visited[cell.x][cell.y] = true
while(heap.notempty):
    cell c = heap.pop()
    h = c.height
    for(i = 0 → len(dirs) - 1):
        nx, ny = (c.x, c.y) + dirs[i]
        if(visited[nx, ny] = false & mat[nx, ny] < h):
            visited[nx, ny] = true
            water := water + h - mat[nx, ny]

            heap.push(cell(nx, ny, mat[nx, ny]))

return water

```

252. Meeting Rooms I [Facebook](#)

I: Given a list of (start, end) conference time intervals, check if one can take all intervals.

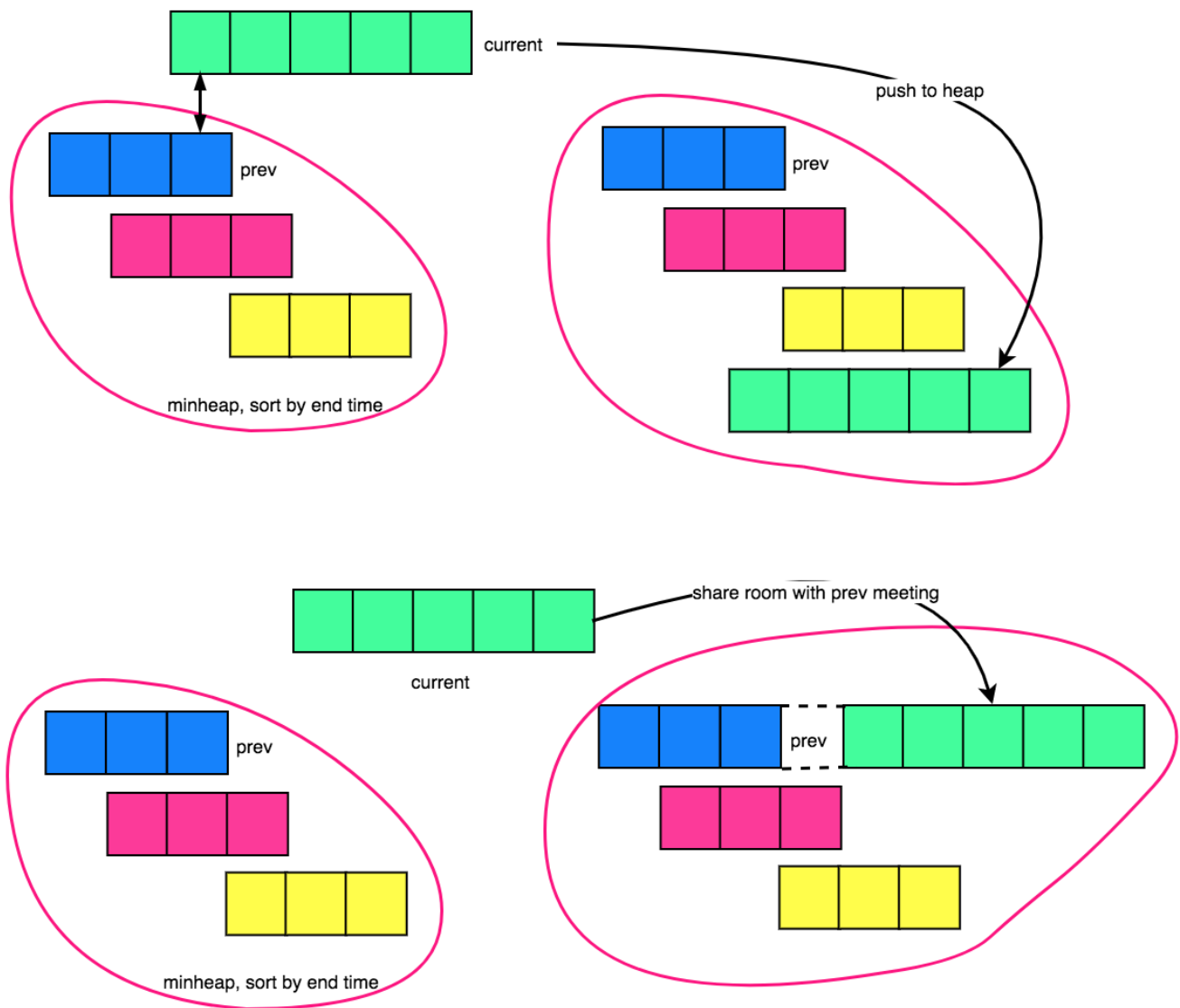
```

sort interval array by start time
for(i = 1; i < intervals.length;):
    if(intervali-1.end > intervali.start): return false
return true

```

253. Meeting Rooms II [Facebook](#)

Find minimum rooms for arranging n meeting intervals.



$\text{sort}(\text{intervals})_{a.start-b.start}$ * only sort by start, no need consideration for end

$\text{min} \cdot \text{heap heap}_{a.end-b.end}$ * only sort by end, no need consideration for start

$\text{heap.offer}(\text{interval}_0)$ * arrange the 1st meeting

for($i = 1 \rightarrow \text{len}(\text{intervals}) - 1$): * arrange meeting which starts earliest.

$\text{interval prev} := \text{heap.poll}$ * check the meeting which ends earliest

$\text{interval curr} := \text{intervals}[i]$

if($\text{prev.end} > \text{curr.start}$): * collision detected, need one more room

$\text{heap.offer}(\text{curr})$

$\text{heap.offer}(\text{prev})$.

else:

$\text{prev.end} = \text{max}(\text{prev.end}, \text{curr.end})$

$\text{heap.offer}(\text{prev})$

return heap.size

Find the point where maximum intervals overlap **Uber**

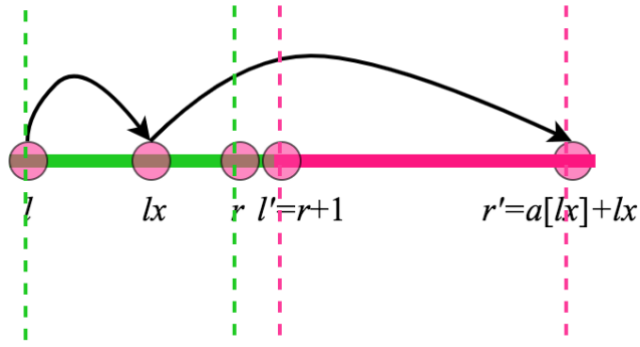
```
Arrays.sort(array = arrive, size = n)
Arrays.sort(array = exit, size = n)
i = 1, j = 0, in = 1
while(i < n & j < n)
    if(arrive[i] ≤ exit[j]):
        in ← in + 1
        if(maxOverlap < in):
            maxOverlap ← in
            map[maxOverlap].add(arrive[i])
        i = i + 1
    else:
        in = in - 1
        j = j + 1
return map[maxOverlap]
```

Intuition & Algorithm #2

```
timeLine[timestamp ⇒ overlapNum]: TreeMap
for(itv: intervals):
    timeLine[itv.start] := timeLine[itv.start] + 1
    timeLine[itv.end] := timeLine[itv.end] - 1
    re = max(timeLine[itv.start], re)
return re
```

55. Jump Game I

45. Jump Game II **Microsoft**



Intuition & Algorithm

Each time select one point $lx \in [l, r]$, which maximize the value of $a[lx] + lx$. And finally update the window from $[l, r] \Rightarrow [r + 1, a[lx] + lx]$.

$jumps = 0, l = 0, r = 0, nr := 0$

while($l \leq r$):

if($r \geq n - 1$): *return jumps*

for($l \leq r; l := l + 1$):

$nr := \max(next_r, l + nums[l])$

$r := nr$

$jumps := jumps + 1$

343. Integer Break

for($factor = 1 \rightarrow \lceil \sqrt{n} \rceil$):

$times := \frac{n}{factor}$

$remain := \text{mod}(n, factor)$

if($remain = 0$): $re = \max\{result, factor^{times}\}$

else: $re = \max(re, factor^{times-1} * (remain + i), factor^{times} * remain)$

397. Integer Replacement

function(n):

if(n is odd):

if $\left(\frac{n+1}{2} \& 1 == 0\right)$: *return function*($n + 1$) + 1

elif $\left(\frac{n-1}{2} \& 1 == 0\right)$: *return function*($n - 1$) + 1

if (n is odd): return function $\left(\frac{n}{2}\right)$

14. Longest Common Prefix [Yelp](#)

Apply Trie algorithm

```
class trie · node{int childNumber, randomChild, next[26]}
TrieNode root ← new TrieNode()
insert(TrieNode o, String str):
    p ← o
    for(char c: str):
        idx ← c - 'a'
        if (p.next[idx] is null):
            p.randomChild = idx
            p.next[idx] = new TrieNode()
            p.childNumber += 1
        p ← p.next[idx]

longest · common · prefix():
    while(true):
        if (p.childNumber ≠ 1): break
        elif (p.childNumber == 1): result += (char)(p.randomChild + 'a')
```

49. Group Anagrams

Solution I: $map[sorted\ word_i] \rightarrow word_i$

Solution II: assign an unique prime number product for string.

187. Repeated DNA Sequence

151. Reverse words in a string

Solution: reverse each word, then reverse the whole string.

6. Zigzag Conversion

214. Shortest Palindrome

Apply KMP algorithm of s and s' reverse

DFS & BFS

360. Valid Sudoku & Sudoku Solver

51. N-Queens & II (52) [Zenefits](#)

78. Subsets (Duplicated Subsets allowed)

90. Subsets II (No Duplicated Subsets allowed) [Facebook](#)

Given a collection of integers that might contain duplicates, $nums$, return all possible subsets.
The solution set must **not contain duplicate subsets**.

Intuition & Algorithm

$List[List[int]]$ subsets($int[] A$):

$sort(A)$

$List[int]$ load := $\{\phi\}$

$List[List[int]]$ re := $\{\phi\}$

int idx := -1

$dfs(A, idx, load, re)$

return re

$void dfs(int[] A, int idx, List[int] load, List[List[int]] re):$

$re.add(new ArrayList(load))$

for($n_x := idx + 1 \rightarrow len(A) - 1$):

*if($n_x > idx + 1 \ \& \ A[n_x] = A[n_x - 1]$): continue \rightarrow **important!!** to avoid duplicates*

$load.add(A[n_x])$

$dfs(A, n_x, load, re)$

$load.remove(load.size - 1)$

79. Word Search

Intuition

DFS + Backtrack

22. Generate Parentheses Google Uber Zenefits

List<String> result = { ϕ }

dfs(l, r, n, s):

if (l = r & l = n): result.add(s)

if (l < n): dfs(l + 1, r, n, s + '(')

if (l < r): dfs(l, r + 1, n, s + ')'

46. Permutation

47. Permutation II

384. Shuffle an Array

dfs(l, r, a[]):

if (l > r): result.add(a), return;

for (k = l \rightarrow r):

swap(a, l, k)

dfs(l + 1, r, a)

swap(a, l, k)

39. Combination Sum

40. Combination Sum II Uber

Intuition & Algorithm

The key idea of avoiding duplicates is *if (i > idx + 1 & A[i] = A[i - 1]): continue*

List<List<int> > combination = sum(int[] A, int target):

List<List<int> > result = new ArrayList<> ()

Arrays.sort(A)

dfs(A, result, candidate = [ϕ], sum = 0, target, idx = -1)

return result

```

dfs(A, result, load, sum, target, idx)
    if(sum > target): return;
    if(sum == target): result.add(load), return
    for(i = idx + 1 → len(A) - 1):
        if(i > idx + 1 & A[i] == A[i - 1]): continue
        load.add(A[i])
        dfs(A, result, load, sum + A[i], target, i)
        load.remove(load.size - 1)

```

54. Spiral Matrix & II (59) Google Microsoft Uber

```

while(true):
    for(j = left; j ≤ right; ): print(a[top, j]) ★ scan the top
    top := top + 1 ★ remove the top, if(left > right || top > bottom): break;

    for(i = top; i ≤ bottom; ): print(a[i, right]) ★ scan the right
    right := right - 1 ★ remove the right, if(left > right || top > bottom): break;

    for(j = right; j ≥ left; ): print(a[bottom, j]) ★ scan the bottom
    bottom := bottom - 1 ★ remove the bottom, if(left > right || top > bottom): break;

    for(i = bottom; i ≥ top; ): print(a[i, left]) ★ scan the left
    left := left + 1 ★ remove the left, if(left > right || top > bottom): break;

```

112. Path Sum I

113. Path Sum II

129. Sum Root to Leaf Numbers

```

boolean path · sum(TreeNode o, int sum):
    if(o is null): return false
    if(o is a leaf): return sum == o.val
    if(path · sum(o.l, sum + o.val) || path · sum(o.r, sum + o.val)):
        return true
    return false

```

II. Solution: DFS + backtrack

```
dfs(TreeNode o, List<int>candidate, int s, int sum):  
    if(o is a leaf):  
        if(s == sum): result.add(candidate)  
        return  
    if(o.left != nil):  
        candidate.add(o.left.val)  
        dfs(o.left, candidate, s + o.left.val, sum)  
        candidate.remove last  
    if(o.right != nil):  
        candidate.add(o.right.val)  
        dfs(o.right, candidate, s + o.right.val, sum)  
        candidate.remove last
```

400. N-th Digit

43. Multiply Strings Uber Facebook Twitter

```
reverse(a), reverse(b)  
re = [len(a) + len(b)]  
for(i = 0 → len(a) - 1):  
    for(j = 0 → len(b) - 1):  
        re[i + j] := re[i + j] + a[i] * b[j]  
for(i = 0 → len(re) - 1):  
    re[i + 1] := re[i + 1] +  $\frac{re[i]}{10}$   
    re[i] := mod(re[i], 10)  
i := len(re) - 1  
while(i ≥ 1 & re[i] = 0): i := i - 1  
for(; i ≥ 0; ): result ← result.append(arri)
```

50. Pow(x, n)

$x^{11} = x^8 * x^2 * x^1$

```

pow(x, n):
    if(n = 0): return 1
    temp ← x
    while(n > 0):
        if(n & 2 = 1): re *= temp
        n =  $\frac{n}{2}$ 
        temp *= x (temp = x1, x2, x4, x8)

```

372. Super Pow

$$a^{1234} = (a^{1000})^1 (a^{100})^2 (a^{10})^3 (a^1)^4$$

```

super · pow(a, b[])
    result = 1, a = mod(a, 1337)
    for(i = b.length - 1; i ≥ 0; ):
        result = result * mod(fastRow(a, b[i]), 1337)
        a = fastRow(a, 10)

```

149. Max Points on a Line

```

result = 0
for(i = 0; i < p.length; ):
    same = 0, vertical = 0, e = 1
    for(j = 0; j < p.length; )
        if(i = j): continue
        if(pi.x = pj.x & pi.y = pj.y): same += 1
        elif(pi.x = pj.x) vertical += 1
        else: map  $\left[ \frac{p_i.y - p_j.y}{p_i.x - p_j.x} \right] += 1$ 

    e = max(same + max(map  $\left[ \frac{p_i.y - p_j.y}{p_i.x - p_j.x} \right]), vertical)$ 

    result = max{result, e}

```

342. Power of Four **Uber**


```

is · power · 4(int num):
    double x = log4(num)
    double y = Math.floor(x)
    if (|x - y| < 10-6): return true
    return false

double log4(int num):
    return (log10(num))/log10(4)

```

Solution II:

$$4^n - 1 = (4 - 1)(4^{n-1} + 4^{n-2} + \dots + 4^1 + 1) = 3K$$

```
return num & (num - 1) == 0 and num > 0 and (num - 1)%3 == 0
```

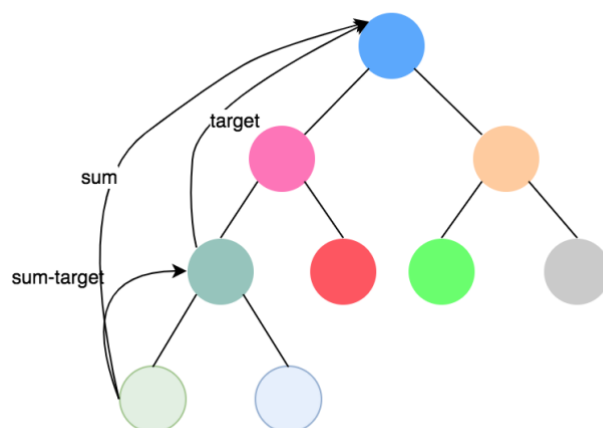
326. Power of Three

```
re = n > 0 & n & (n - 1) = 0 & mod(n - 1, 3) = 0
```

Hash Map & Tree Map & BIT

437. Path Sum III

You are given a binary tree in which each node contains an integer value. Find the number of paths that sum to a given value. The path does not need to start or end at the root or a leaf, but it must go downwards (traveling only from parent nodes to child nodes).



Intuition & Algorithm

The same idea as “count how many subarrays that sum to target”, use $sum_{current}$ to keep track of the sum from root node to current node, and $map[sum_{current}, times]$ to record the number of times that the sum of root nodes down to current node shows up.

```

map[0] = 1
re = dfs(o, map, sumcurrent = 0, target)
dfs(TreeNode o, Map<int, int> map, int sumcurrent, int target):
    if(o is null): return 0
    sumcurrent := sumcurrent + o.val
    re := re + map[sumcurrent - target]
    map[sumcurrent] := map[sumcurrent] + 1
    re := re + dfs(o.left, map, sumcurrent) + dfs(o.right, map, sumcurrent)
    map[sumcurrent] := map[sumcurrent] - 1
    return result

```

Number of subarrays having sum exactly equal to k [GeeksForGeeks](#)

```

int findSubarraySum(int[ ] A, int T):
    map: Presum ⇒ counts
    re := 0
    s := 0
    map[0] := 1
    for(i = 0 → len(A) - 1):
        s := s + A[i]
        re := re + map[s - T]
        map[s] := map[s] + 1
    return re

```

1. Two Sum

$O(n)$ solution: hash · map

$O(\log(n) * n)$ solution: sort + 2 pointers

Interview Q. [Uber](#)

Implement a hash-map which supports 3 operations: put, get & putAll(value), to be noticed, putAll(value) updates all KV pairs with the new value.

```

public class MyMap<K,V>:
    private Map<K, V> map

```

```

private Set<K> nonAffectedkeys
private V val = null
public void put(K key, V value):
    keys.add(key)
    map[key] = value
public T get(K key):
    if(nonAffectedkeys.contains(key)): return map[key]
    else return val
public void putAll(V value):
    keys.clear() | keys = new HashSet
    val = value

```

Interview Q. Uber

Given a hash-map, like “apple”:10, “banana”:50, “peach”:30

The value part suggests the relative weight, this question ask you to randomly output a key in accordance with the corresponding weight.

```

rid = rand.nextInt(range =  $\sum \text{map.values}$ )
pos ← 0, id ← 0
it ← map.entrySet().iterator(), result = ""
while(pos < rid & it.hasNext()):
    KV ← it.next(), result = KV.getKey(), weight ← KV.getValue()
    pos ← pos + weight
return result

```

The above algorithm takes $O(n)$ time

Solution II:

If the input is given in a array format (or we can export the KV pairs in hash-map into an array), we can search the target using binary search $O(\log n)$

```

arr = ["apple": 10, "banana": 50, "peach": 30]
sum = [10,60,90]
l = 0, r = arr.length - 1
while(l < r):
    mid = (l + r) >> 1

```

```

    if( $sum_{mid} - sum_l + arr[l].weight < rid$ ):  $l = mid + 1$ 
    elif( $sum_{mid} - sum_l + arr[l].weight == rid$ ): return  $arr[mid].key$ 
    else:  $r = mid$ 
return  $arr[l].key$ 

```

525. Contiguous Array Facebook

Given a 01 array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

Intuition & Algorithm

Translate the $A[i] = 0 \rightarrow -1$, then this problem is to find such a longest subarray whose sum equals to zero.

```

 $map := \{prefixsum \Rightarrow index\}$ 
 $map[0] := -1$ 
 $s := 0, re := 0$ 
for( $i = 0 \rightarrow n - 1$ ):
     $s := s + (A[i] < 0) \rightarrow -1: 1$ 
    if( $s = 0$ ):  $re := \max(re, i)$ 
    if( $map[s] \neq nil$ ):  $re := \max(re, i - map[s])$ 
    if( $map[s] = nil$ )  $map[s] = i$ 
return  $re$ 

```

325. Maximum Size Subarray Sum Equals k Facebook Palantir

```

 $mp: (type = (int \rightarrow int), default: mp[0] = -1)$ 
for( $i = 0 \rightarrow n - 1$ ):
     $sum = sum + arr[i]$ 
    if( $mp[sum - k] \neq null$ ):  $result = \max(result, i - mp[sum - k] + 1)$ 
    if( $mp[sum] == null$ ):
        * if sum occurs more than once, only record the leftmost index associated with it
         $mp[sum] = i$ 

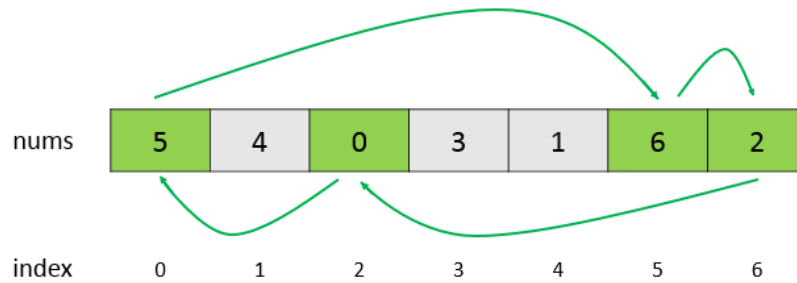
```

Follow Up: if it is asked to find the minimum size subarray, then just remove the blue line of code above.

565. Array Nesting Apple

A zero-indexed array A of length N contains all integers from 0 to $N - 1$. Find and return the longest length of set S , where $S[i] := (A[i], A[A[i]], A[A[A[i]]], \dots)$ subjected to the rule below.

i.e. $A = [5, 4, 0, 3, 1, 6, 2]$, $re = (A_0 = \mathbf{5} \rightarrow A_5 = 6 \rightarrow A_6 = 2 \rightarrow A_2 = 0 \rightarrow A_0 = \mathbf{5})$ form a cycle with length of 4.



Elements 5, 0, 6, 2 are added to the current set, if we start with any of the index from 0, 2, 5, 6

Intuition & Algorithm

```

int arrayNesting(int[ ] A):
    bool[ ] used := [len(A)]
    re := 0
    for(i = 0 → len(A) - 1):
        if(used[i] = False):
            start := A[i], len := 1
            * Mark all number is a circle as visited
            do: start := A[start], len := len + 1, used[i] := True
            while(start ≠ A[i])
            re := max(re, len)

```

return re

17. Letter Combinations of a Phone Number [Uber](#)

Follow Up: Given a phone number list & word dictionary, find the phone number which maps the most number of words.

map: 2: {a, b, c}, 3: {d, e, f}, 4: {g, h, i}, 5: {j, k, l}, 6: {m, n, o}, 7: {p, q, r, s}, 8: {t, u, v}, 9: {w, x, y, z}

```
for(word w: wordlist):
    for(char ch: w): phone · number ← phone · number.append(map[ch])
    word2num[w] ← phone · number
for(string word: word2num.keySet()):
    phone · number ← word2num[word]
    num2wordlist[phone · number].add(word)
    if(num2wordlist.size > maxlen):
        maxlen = num2wordlist.size
    result = phone · number
return result
```

200. Number of Islands [Google](#) [Facebook](#) [Microsoft](#) [Amazon](#) [Zenefits](#)

Given a 2d grid map of '1's (land) and '0's (water), count 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.

Algorithm

Nothing trivial, just do DFS by starting from each island (1)

```
int number · island(grid2d):
    for(cell c: grid2d):
        if(c == '1'):
            visited.add(c)
            dfs(c, visited)
            island += 1
```

305. Number of Islands II [Google](#)

Intuition & Algorithm

Union-Find, for each of newly added island, we increase the number of islands by 1. Then, find 4-directionally adjacent island and union them, if the union two islands have already in the same group, do nothing. Otherwise, decreasing the number of islands by 1.

```
List[int] numIslands(int m, int n, int[ ][ ] positions)
    re = [ϕ], islands = 0, marked = {ϕ}
    uf = UnionFind(m * n)
    for(i = 0 → len(positions) - 1):
        islands := islands + 1
        x, y = positions[0], positions[1]
        id = x * n + y
        marked.add(id)
        for(k = 0 → len(dirs) - 1):
            nx, ny = x + dirs[i, 0], y + dirs[i, 1]
            if(nx ∈ [0, m - 1] & ny ∈ [0, n - 1] & marked[nx * n + ny] = true):
                nextId = nx * n + ny
                bool b = uf.union(id, nextId)
                if(b): islands := islands - 1
    re.add(islands)
```

128. Longest Consecutive Sequence

Solution I: hash-map

```
for(int x: nums): set.add(x)
for(int x: nums):
    if(set[x] ≠ null): continue
    l = x - 1
    while(set[l] ≠ null): set[l] = null, l := l - 1
    r = x + 1
    while(set[r] ≠ null): set[r] = null, r := r + 1
    set[x] = null
    result = max(result, r - l)
```

find(i):

if(*fa*[*i*] ≠ *i*): *fa*[*i*] = *find*(*fa*[*i*])

return fa[*i*]

connect(i, j):

if((*o_i* = *find*(*i*)) ≠ (*o_j* = *find*(*j*))):

if(*rank*[*o_i*] > *rank*[*o_j*]): *fa*[*o_j*] = *o_i*

elif(*rank*[*o_i*] < *rank*[*o_j*]): *fa*[*o_i*] = *o_j*

else: *fa*[*o_i*] = *o_j*, *rank*[*o_i*] += 1

644. Maximum Average Subarray II Google

Find a subarray with a length ($\geq k$) whose average is the largest.

Solution: binary guess or Simulated Annealing algorithm. Since certainly, such an average satisfying the condition must be in the range of $[\min, \max]$, so we can continuously enumerate avg in $[\min, \max]$, and then check if there is a subarray.

$$\sum_i^j A[x] \ (j - i + 1 \geq k)$$

$$\text{s.t. } \frac{\sum_i^j A[x]}{(j-i+1)} \geq \text{avg} \text{ or } \sum (arr[k] - \text{avg}) \geq 0.$$

boolean check(double avg_{est}, nums, k):

sum = 0.0

for(*i* = 0 → *k* − 1): *sum* = *sum* + *nums*[*i*] − *avg_{est}*

if(*sum* ≥ 0.0): *return true*

pre_{sum} = 0.0, *mi_{presum}* = 0.0

for(*i* = *k* → *n* − 1):

sum = *sum* + (*nums*[*i*] − *avg_{est}*)

pre_{sum} = *pre_{sum}* + (*nums*[*i* − *k*] − *avg_{est}*)

min_{presum} = *min*(*min_{presum}*, *pre_{sum}*)

if(*sum* − *mi_{presum}* ≥ 0): *return true*

return false

findMaxAverage(nums, k):

mi := *min*(*nums*[*i*]), *mx* := *max*(*nums*[*i*]), $\forall i \in [0, \text{len}(\text{nums}) - 1]$

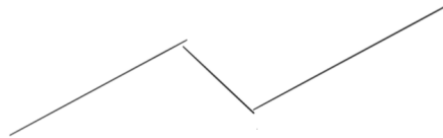

```

if(mi == mx):return mx
error = 231 - 1,prevest = 0.0
while(error > 10-6):
    avgest =  $\frac{mi+mx}{2}$ 
    if(check(avgest,nums,k)):mi = avgest
    else:mx = avgest
    error = |avgest - prevest|
    prevest = avgest
return mx

```

665. Non-decreasing Array Google

Check if an array can be non-decreasing by modifying at-most 1 element.



the above picture suggests an example of such array that making an array non-decreasing by modifying at-most one element.

So we just need to find such an position i such that $nums[i] > nums[i + 1]$, and then I make $nums[i]$, $nums[i+1]$ equal by either $nums[i] = nums[i + 1]$ (case i) or $nums[i + 1] = nums[i]$ (case ii). if in either of cases, $nums[0:i]$ is in non-decreasing order and $nums[i + 1:n - 1]$ is also in non-decreasing order. Then return true, return false otherwise.

670. Maximum Swap Facebook

Intuition & Algorithm

At each digit of the input number in order, if there is a larger digit that occurs later, we know that the best swap must occur with the digit we are currently considering. We will compute $last[d] = i$, the index i of the last occurrence of digit d (if it exists).

Afterwards, when scanning the number from left to right, if there is a larger digit in the future, we will swap it with the largest such digit; if there are multiple such digits, we will swap it with the one that occurs the latest.

```

int maxSwap(int x):
    char[ ] A = String(x).toCharArray
    int[ ] index = [10]
    for(i = 0 → len(A) - 1): index[A[i] - 0] = i

    for(i = 0 → len(A) - 1):
        for(d = 9 → A[i] - '0' + 1):
            if(index[d] > i):
                swap(A, i, index[d])
            return int(A)

    return num

```

Heap-Sort Didi Labs

```

heapify(arr, n, i):
    largest ← i
    l ← largest * 2
    r ← largest * 2 + 1
    if(l < n & arrl > arrlargest): largest ← l
    if(r < n & arrr > arrlargest): largest ← r
    if(largest ≠ i)
        swap(arr, i, largest)
        heapify(arr, n, largest)

```

heap · sort(arr)

build · max · heap $\left\{ \begin{array}{l} \text{for } (i = \frac{n}{2} - 1; i \geq 0;): \\ \quad \text{heapify}(arr, n, i) \end{array} \right.$

for(i = n - 1; i ≥ 0;)

swap(arr, 0, i) → *remove the top*

heapify(arr, i, 0) → *maintain max · heap for arr₀ⁿ⁻², arr₀ⁿ⁻³, ..., arr₀⁰)*

K-dimensional Tree

common questions:

range query (find all points inside the given range $R: [x_1, x_2] \times [y_1, y_2]$)

269. Alien Dictionary

examples: input: words[] = {"baa", "abcd", "abca", "cab", "cad"}

output: Order of characters is 'b', 'd', 'a', 'c'

graph g $\rightarrow \{\phi\}$

for(word_i, word_j in words):

$idx \leq \{word_i[id] \neq word_j[id], 0 \leq id < \min(\text{len}(word_i, word_j))\}$

add directed edge {word_i[idx], word_j[idx]} *into g*

do topological sort on g

271. Closest Binary Search Tree Value & 272

result = $2^{31} - 1$, *dif* = $2^{31} - 1$

while(root \neq null):

if (|root.val - target| < *dif*):

dif = |root.val - target|

result = root.val

if (target < root.val): root = root.left

elif (target > root.val): root = root.right

else: return root.val

return result

272. Find k Closet Values

List(int) closetKvalues(TreeNode root, target):

List(int) result = { ϕ }

findLeft(root.left, target, lc(type = stack))

findRight(root.right, target, rc(type = stack))

while (result.size() < k):

if (lc.isEmpty()): result.add(rc.pop())

elif (rc.isEmpty()): result.add(lc.pop())

elif (|lc.peek() - target| < |rc.peek() - target|): result.add(lc.pop())

```

        else: result.add(rc.pop())
    return result

```

```

findLeft(TreeNode root, target):
    if(root == null): return
    findLeft(root.right, target, lc)
    lc.push(root.val)
    if(target < root.val): return
    findLeft(root.left, target, lc)

```

Solution 2. flatten BST into sorted list, then find k closet node on list

```

static TreeNode head = null
static TreeNode prev = null
flatten(TreeNode root):
    if(root == null): return
    flatten(root.left)
    if(head == null):
        head = root
        prev = root
    else:
        prev.right = root
        root.left = prev
        prev = root
    flatten(root.right)

```

```

List<int> findkclosest(int target, int k):
    result = {ϕ}
    TreeNode p = findNodeClosest2Target(head, target)
    if(p.left == null): result.add(p → p.right + k - 1)
    elif(p.right == null): result.add(p → p.left - l + 1)
    else:
        l = p.left, r = p.right, result.add(p.val)
        while(result.size() < k):
            if(|l.val - target| ≤ |r.val - target|):

```

```

        result.add(l.val), l = l.left
    else: result.add(r.val), r = r.right
return result

```

Graph & Topo-sort & Union-Find

127. Word Ladder & II (126) Uber Zillow

```

Queue Q ← {Node(beginword, 1)}
worddict.add(endword)
while(Q.size > 0):
    Node top ← Q.poll
    word ← top.word, level ← top.level
    if(word == endword) return true
    for(i = 0 → len(word) - 1):
        for(c = a → z):
            if(word[i] ≠ c):
                temp = word[i], word[i] = c, strnew = String(word[i])
                if(worddict[strnew] ≠ null):
                    Q.offer(Node(strnew, level + 1))
                    worddict.remove(strnew)
                wordi = temp #backtrack
return false

```

Interview Q. Bipartite graph check Uber

```

queue.offer(node{src, color = 0})
while(queue.is not empty):
    node nd = queue.pop
    u = nd.vertex, color = nd.color
    if(matrix[u][u] == 1): return false
    for(v: u.adjacent nodes):
        if(color[v] == 1 - color): continue
        elif(matrix[u][v] == none): queue.offer(node{v, 1 - color})

```

```

        elif(matrix[u][v] == red):return false
return true

```

433. Minimum Genetic Mutation Twitter

Determine the minimum number of mutations needed to mutate from "start" to "end". If there is no such a mutation, return -1.

```

int minMutation(String start, String end, String[] bank):
    if(len(bank) == 0):return - 1
    Queue[String] que = {ϕ}, Set[String] genebank = {bank}, Set[String] used = {ϕ}
    level = 0

    while(que.size > 0):
        size = que.size
        used.addAll(que) * Mark all string on current level as visited
        for(i = 0 → size - 1):
            String gene = que.poll
            if(gene == end): return level
            char[] chars = gene.toCharArray
            for(j = 0 → len(chars) - 1):
                for(k = 0 → len([A,C,G,T]) - 1):
                    if(chars[j] != elements[k]):
                        swap(chars[j], elements[k])
                        String genex = String(chars)
                        if(geneBank[genex] != nil & used[genex] != nil):
                            que.offer(genex), used.add(genex)
                        swap(chars[j], elements[k])

            level := level + 1
    return - 1

```

207. Course Schedule

210. Course Schedule II Facebook Zenefits

There are a total of n courses you have to take, labeled from 0 to n-1. Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as

a pair: $[0,1]$. Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses. There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

Intuition & Algorithm

Topological sort

```

in[vi] = 0, vi ∈ g.vertex
for(edge e: g.edges): in[e.to] += 1
Queue Q = {ϕ}
for(vertex vi ∈ g.vertex)
    if(in[vi] = 0): Q.add(vi)
while(Q is not empty):
    v = Q.poll()
    result.add(v)
    for(vj ∈ v.adjacent):
        in[vj] := in[vj] - 1
        if(in[vj] = 0): Q.offer(vj)
if(result.size ≠ g.vertex.number): "no solution"

```

310. Minimum Height Trees Google

For an undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

Intuition & Algorithm

Topological Sorting on an undirected graph, we keep removing the node whose indegree is 1, that is, the leaf node. We update minHeightTrees in each level, the last one remained will be the list containing roots.

```

List[int] findMinHeightTrees(int n, int[ ][ ] edges):
    if(len(edges) = 0): return [0]
    g: Map[int, Set[int]], ind = [n], List[int] re = [ϕ]

```

```

for(int[ ] e: edges):
    ind[e[0]] := ind[e[0]] + 1, ind[e[1]] := ind[e[1]] + 1
    g[e[0]].add(e[1]), g[e[1]].add(e[0])

Queue[int] que = [v | v, ind[v] = 1]
while(que.size > 0):
    sz = que.size
    re.clear, re.addAll(que)
    for(i = 0 → sz - 1):
        node = que.poll
        for(anode: g[node]):
            ind[anode] := ind[anode] - 1
            if(ind[anode] = 1):
                que.offer(anode)

return re

```

Math & Geometry & Number Theory

365. Water and Jug Problem (Number Theory)

You are given two jugs with capacities x and y liters. There is an infinite amount of water supply available. You need to determine whether it is possible to measure exactly z liters using these two jugs. For example, $x = 3, y = 5, z = 4 \rightarrow \text{can measure}$; $x = 2, y = 6, z = 5 \rightarrow \text{can't measure}$

Solution: this is a kind of Euclidean problem, where I can assume you did m times operation on jug_x , n times operations on jug_y to measure z liters: $m * x + n * y = \text{gcd}(x, y)$, so z can be measured if & only if $\text{mod}(z, \text{gcd}(x, y)) = 0$.

```

canMeasure(x, y, z):
    if(z = 0): return true
    if(x + y < z): return false
    if(x > y): swap(x, y)
    if(z mod gcd(x, y) = 0): return true
    return false

```



```
gcd(x, y):
    if (y = 0): return x
    else: return gcd(y, x % y)
```

367. Valid Perfect Square [LinkedIn](#)

Given a positive integer, determine if the given number is perfect square or not.

Intuition & Algorithm

```
bool isPerfectNum(int x):
    if (num ≤ 0): return false
    low := 1, high := x
    while (low < high):
        mid :=  $\frac{low+high}{2}$ 
        if ( $\frac{x}{mid} > mid$ ): low := mid + 1
        else: high := mid
    return low2 = x
```

168. Excel Sheet Column Title [Facebook](#) [Microsoft](#) [Zenefits](#)

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example, $1 \rightarrow A, 2 \rightarrow B, \dots, 26 \rightarrow Z, 27 \rightarrow AA, 28 \rightarrow AB$

```
string convertToTitle(int n):
    re := ""
    while (n > 0):
        r := mod(n, 26)
        if (r = 0): re := "Z" + re
        else: re := (char)('A' + r - 1) + re
        n :=  $\frac{n}{26}$ 
    if (r = 0): n := n - 1
```

469. Convex Polygon [Google](#)

boolean positive = false, negative = false

n = poly.size

for(int B = 0; B < n; B += 1) :

A = (B - 1 + n) % n, C = (B + 1 + n) % n

BA = (poly[A].x - poly[B].x, poly[A].y - poly[B].y)

BC = (poly[C].x - poly[B].x, poly[C].y - poly[B].y)

if(BA × BC < 0):negative = true

else:positive = true

if(positive & negative):return false

return true

Area of Polygon

for(int B = 1; B < poly.size; B += 1) :

A = (B - 1), C = B + 1

BA = (poly[A].x - poly[B].x, poly[A].y - poly[B].y)

BC = (poly[C].x - poly[B].x, poly[C].y - poly[B].y)

area += $\frac{BA \times BC}{2}$

Intersection point of ray & triangle-3d.

Triangle-3d can be presented as: $Ax + By + Cz + D = 0$ (1)

Ray can be presented as: $P = O + t \cdot R$ (2).

Supposing the intersection point lies on the triangle is (P_x, P_y, P_z) , so we have:

$$AP_x + BP_y + CP_z + D = 0$$

$$A(O_x + tR_x) + B(O_y + tR_y) + C(O_z + tR_z) + D = 0$$

$$(AO_x + BO_y + CO_z) + D = -t(AR_x + BR_y + CR_z)$$

because the normal vector of plane (based on triangle) is $N(A, B, C)$, and $(AO_x + BO_y + CO_z) = N \cdot O$, and $(AR_x + BR_y + CR_z) = N \cdot R$, $t = \frac{N \cdot O + D}{N \cdot R}$, so the intersection point is $P = O + \frac{N \cdot O + D}{N \cdot R} R$

Fibonacci progression

Solution: $O(\log N)$ time

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{\frac{n}{2}} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{\frac{n}{2}} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{(n\&1)}$$

$$F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

```
power(int F[], int n):
    if (n ≤ 1) return;
    power(F, n/2):
    multiply(F, F)
    if (n&1 == 1) multiply(F, M)
```

Check if one number n is a Fibonacci number

return $\sqrt{5n^2 + 4} \in \mathbb{Z}^+ \parallel \sqrt{5n^2 - 4} \in \mathbb{Z}^+$

Find Path In a Maze

solution: A* algorithm.

```
class cell{int r, c, g(distsource,current), h(distcurrent,goal), f(= g + h)}
```

maintain 2 sets: open(f, cell), closed(cell)

intialize set open with open = {0, source}, closed = {}.

while(open is not empty):

 cell ← open.pollfirst(pop the cell with the smallest f)

 add cell into closed

 for(neighbor of cell):

 if(neighbor ∉ {closed ∪ block}):

 if(neighbor == goal): found = true, return;

 g_{new} = cell.g + 1, h_{new} = dis(neighbor, goal), f_{new} = g_{new} + h_{new}

 if(neighbor.f > f_{new}): open.update(neighbor, f_{new})¹

if(! found): "not found"

note¹: if neighbor already exists in the open, update it with smallest f_{new};

: otherwise, insert such pair into open

String & Easy

166. Fraction to Recurring Decimal

fractionToDecimal(int n, int d) :

```
StringBuilder sb = new StringBuilder();
if((n > 0 & d < 0) || (n < 0 & d > 0)): sb.append("-")
long num = (long)|n|, den = (long) |d|

sb.append(Long.toString( $\frac{num}{den}$ )) the integer part

num %= den
if(num > 0) sb.append('.')
map<long, int> ls → map remainder to decimal index

p = 0, ls[num] = sb.length() (i.e.  $\frac{20}{3} = 6.\dots$ , remainder = 2), ls[2] = len('6.') = 2

while(num > 0):

    num := num * 10, sb.append( $\frac{num}{den}$ ), num := mod(num, den)

    if(ls[num] ≠ null)
        idx := ls[num])
        sb.insert(idx, '(')
        sb.append(')')
        break
    else: ls[num] = len(sb)

return sb.toString()
```

29. Divide two Integers

526. Beautiful Arrangement

dfs(N, index, used, count)

```
if(index > N):
    count = count + 1, return
for(i = 1 → N):
    used[i] = true
```

dfs(N, index + 1, used, count)

used[i] = false

482. License Key Formatting

just familiar with several functions:

String: replace(some str, another str), i. e. replace("-", "")

String: toUpperCase()

12. Integer to Roman & Roman to Integer (13) Oscar Health

unit: ["", I, II, III, IV, V, VI, VII, VIII, IX]

tens: ["", X, XX, XXX, XL, L, LX, LXX, LXXX, XC]

hundreds: ["", C, CC, CCC, CD, D, DC, DCC, DCCC, CM]

thousands: ["", M, MM, MMM]

$$\begin{aligned} result = thousands \left[\frac{x}{1000} \right] + hundreds \left[\left(\frac{x}{100} \bmod 10 \right) \right] + tens \left[\bmod \left(\frac{x}{10}, 10 \right) \right] \\ + unit[\bmod(x, 10)] \end{aligned}$$

65. Valid Number

For example, $-12.35789e^{+456}$

*regex = $[-+]?(\backslash d + \backslash . ? | \backslash . \backslash d +) \backslash d * (e[-+]? \backslash d +)?$*

Interview Q. Given a word dictionary & a sentence

For example, *dict* = [this, hi, his, is, word], *sentence* = thisisaword

Output = [this, hi, his, is, is, word]

Solution: grab all substring of sentence at first, and check each substring exists in dictionary or not.

Optimization: avoid checking too-long substring.

648. Replace Words Uber

Replace words in a string with shortest prefix in dict.

String replaceWords(List<String> dict, String sentence):

```

String[] words ← sentence.split(" ")
buildTrie(dict)
result ← ""
for(i = 0 → words.length):
    TrieNode p ← root
    j ← 0
    for(; j < words[i].length();):
        if(p.next[words[i][j] - 'a'] ≠ null):
            p ← p.next[words[i][j] - 'a']
            if(p.isWord): break
        else: break
    if(p.isWord): result ← result + words[i].substring(0, j + 1)
    else: result ← result + words[i]
    if(i < words.length - 1): result ← result + " "

```

412. Fizz Buzz

415. Add Strings

```

char[] a ← new StringBuffer(num1).reverse().toString().toCharArray()
char[] b ← new StringBuffer(num2).reverse().toString().toCharArray()
len ← a.length > b.length ? a.length : b.length
result ← array1d(size = len + 1)
for(i = 0 → len):
    ai ← i < a.length ? ai : 0
    bi ← i < b.length ? bi : 0
    resulti ← ai + bi
for(i = 0 → len - 2):
    resulti+1 ← resulti+1 + resulti / 10
    resulti = resulti mod 10

```

```

i ← len − 1, output = ""
while(i ≥ 1 & resulti == 0) i ← i − 1
for(; i ≥ 0;): output ← output + resulti
return output

```

695. Max Area of Island

Solution: simple dfs

Note: cut branches of dfs tree by turn the “island” to water during dfs.

<https://leetcode.com/submissions/detail/140076776/>

67. Add Binary Facebook

```

char[] ca ← a.reverse().toCharArray()
char[] cb ← b.reverse().toCharArray()
i = 0, j = 0
len = max(ca.length, cb.length), int[] re = [len]
for(i = 0; i < len;):
    int ai = i < ca.length? ca[i] − '0': 0;
    int bi = i < cb.length? cb[i] − '0': 0;
    rei = ai + bi
for(i = 0 → len − 2):
    re[i + 1] := re[i + 1] +  $\frac{re[i]}{10}$ 
    re[i] := mod(re[i], 10)
i = len − 1
for(; i ≥ 1 & re[i] = 0; i := i − 1); ⇒ remove unnecessary leading zeros
result = ""
for(; i ≥ 0; -- i): result.append(re[i] + '0')
return result

```

38. Count and Say Facebook

```

StringBuilder rs ← new StringBuilder("1")

```

```

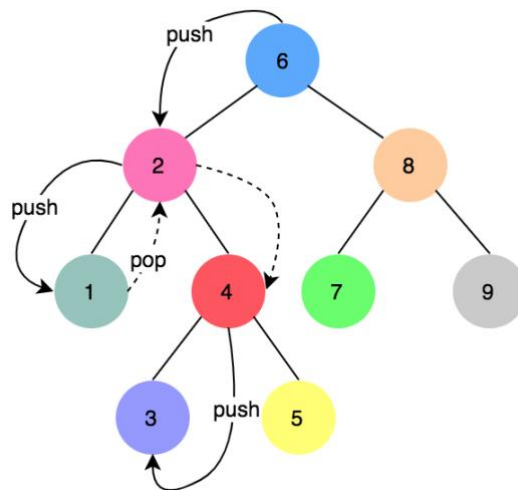
i = 1
while(i < n):
    StringBuilder temp ← ""
    l = 0
    while(l < rs.length()):
        r = l + 1
        while(r < rs.length() & rs[l] == rs[r]): r ← r + 1
        count = r - l
        temp.append(count)
        temp.append(rs[l])
        l = r
    rs = temp
    i ← i + 1
return rs

```

463. Island Perimeter

94. Binary Tree In-order Traversal Microsoft

Non-recursive Solution



```

re = {ϕ}
Stack[TreeNode] st = {ϕ}
TreeNode node := root
while(node ≠ nil):

```



```

    st.push(node), node := node.left
while(st.size > 0):
    node := stack.pop
    re.add(node.val)
    if(node.right ≠ nil):
        node := node.right
    while(node ≠ nil):
        stack.push(node)
        node := node.left
return re

```

202. Happy Number

217. Contains Duplicate

219. Contains Duplicate II

```

for(int x: nums)
    if(set.contains(x)): return true
    set.add(x)
return false

```

Find if there exists 2 indexes $i, j, j - i \leq k, \text{nums}_i == \text{nums}_j$

```

map(int: int)
for(int i = 0; i < nums.length;)
    if(map[numsi] ≠ null):
        index ← map[numsi]
        if(i - index ≤ k): return true
    map[numsi] = i
return false

```

83. Remove Duplicates from Sorted List & II (82)

```

ListNode deleteDuplicates(ListNode head)

```

```

if(head == null):return null
ListNode p ← head
while(p ≠ null):
    if(p.next == null):break;
    ListNode q ← p.next
    while(q ≠ null & q.val == p.val):q ← q.next
    p.next ← q
    p ← q
return head

```

II. Remove all duplicated number only remain single number

```

ListNode deleteDuplicates · 2(ListNode head)
    ListNode dummy ← new ListNode(-1)
    dummy.next ← head
    ListNode p ← dummy, q = dummy.next
    while(q ≠ null):
        while(q.next ≠ null & q.next.val == q.val) q = q.next
        if(p.next == q):
            p = p.next, q = q.next
        else:
            p.next = q.next
            q = p.next

```

242. Valid Anagram

```

count(type = array, size = 26, default = 0)
if(s.length() ≠ t.length()):return false
for(i = 0 → s.length() - 1):
    counter[s[i]] += 1
    counter[t[i]] -= 1
for(i = 0 → 25):
    if(counter[i] ≠ 0):return false
return true

```

147. Insertion Sort List

148. Sort List

```
ListNode sortList(ListNode head):
    if(head == nil or head.next == nil): return head
    ListNode prev = nil, slow = head, fast = head
    while(fast != nil and fast.next != nil):
        prev = slow
        slow = slow.next
        fast = fast.next.next
    prev.next = nil * very important, to cut down one list into two halves
    ListNode left = sortList(head)
    ListNode right = sortList(slow)
    return merge(left, right)

merge(left, right):
    ListNode head = (-1), r = head
    while(left != nil and right != nil):
        if(left.val ≤ right.val):
            r.next = left, r = r.next
        else: r.next = right, r = r.next
    while(left != nil): r.next = left, left = left.next, r = r.next
    while(right != nil): r.next = right, right = right.next, r = r.next
    return head.next
```

387. First Unique Character in a String

```
for(i = 0 → str.length - 1):
    counter[str[i] - 'a'].freq += 1
    if(counter[str[i] - 'a'].freq == 1): counter[str[i] - 'a'].firstIdx = i
for(i = 0 → 25):
    if(counter[i].freq == 1 & counter[i].firstIdx < result):
        result = counter[i].firstIdx
return result
```

290. Word Pattern I & II (291) **Uber**

Given a *Pattern* and a string *str*, find if *str* follows the same pattern

```
words[] ← str.split('\s')
if(words.size ≠ pattern.length): return false
map⟨char, string⟩ char2str
set⟨string⟩ visited
for(i = 0 → pattern.length; ):
    ch ← pattern[i]
    ss ← words[i]
    if(char2str[ch] ≠ nil & char2str[ch] ≠ ss): return false
    if(char2str[ch] is nil & visited[ss] is null): return false
    char2str[ch] → ss
    visited.add(ss)
```

205. Isomorphic Strings

Q. check two strings *s* & *t* are isomorphic strings

Solution: like solution of 290 & 291, use hash-map & hash-set

```
map = array1d(int, default = 0)
visited = array1d(boolean, default = false)
for(i = 0 → s.length() - 1):
    if(map[si] ≠ null & map[si] == ti): return false
    if(map[si] == null & !visited[ti]): return false
    map[si] = ti
    visited[ti] = true
```

II. No empty space between words in a string

Solution: DFS, time $O(n * (n - 1) * \dots * 1 = n!, n \text{ factorial})$

```
boolean wordPatternMatch(pattern, str):
    if(pattern.length is 0 & str.length is also 0): return true
    if(pattern.length is 0): return false
    map⟨char, str⟩ char2str, set⟨str⟩ vis
    return check(pattern, 0, str, 0, char2str, vis)
```

```

boolean check(pattern, i, str, j, char2str, vis):
    if(i == pattern.length): return j == str.length
    for(k = j + 1 → str.length):
        substr ← str.substring(j, k)
        if(char2str[pattern[i]] == nil & vis[substr] == nil):
            char2str[pattern[i]] ← substr
            vis.add(substr)
            if(check(pattern, i + 1, str, k, char2str, vis)):
                return true
            char2str[pattern[i]] ← nil
            vis.remove(substr)
        elif(char2str[pattern[i]] ≠ nil & char2str[pattern[i]] == substr):
            if(check(pattern, i + 1, str, k, char2str, vis)):
                return true
    return false

```

66. Plus One

369. Plus One Linked List Google

Idea: reverse the linked list, plus one on first node and do carry-over on each node whose value exceeds 10, finally reverse the linked list again.

<https://github.com/fu11211129/Algorithm/blob/master/list/google/PlusOneLinkedList.java>

169. Majority Element Zenefits Adobe

Solution: voting algorithm.

```

re := A[0], freq := 1
for(i = 1 → len(A) - 1):
    if(A[i] = re): freq := freq + 1
    else: freq := freq - 1
    if(freq = 0): re = A[i], freq := 1
return re

```

86. Partition List

88. Merge Sorted Array Facebook Microsoft Bloomberg

Given two sorted integer arrays A and B , merge B into A as one sorted array.

Intuition

Scan from the back

```
merge(int[ ] A, m, int[ ] B, n):
```

```
     $i = m - 1, m := m - 1, n := n - 1$ 
```

```
    while( $m \geq 0 \ \& \ n \geq 0$ ):
```

```
        if( $A[m] \geq B[n]$ ):  $A[i] := A[m], i := i - 1, m := m - 1$ 
```

```
        else:  $A[i] := B[n], i := i - 1, n := n - 1$ 
```

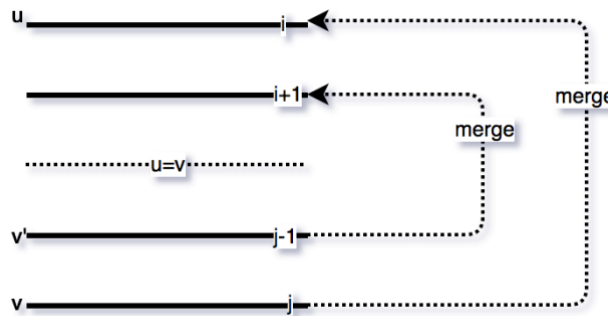
```
    while( $n \geq 0$ ):  $A[i] := A[n], i := i - 1, n := n - 1$ 
```

23. Merge k Sorted Lists Facebook

Intuition

Initialize two pointers, one points to the first list, another point to the last one. Each time merge two lists together into first one.

Algorithm #1



$$\frac{k}{2} * 2n + \frac{k}{4} * 4n + \dots + \frac{k}{2^c} * (2^c n) = O(\log k * kn), n = \frac{\sum_{i=0}^{k-1} list[i].size}{k}$$

```
 $u = 0, v = list.length - 1$ 
```

```
while( $u \neq v$ ):
```

```
     $i = u, j = v$ 
```

```
    while( $i < j$ ):
```

```

    list[i] = merge(list[i], list[j])
    i ← i + 1, j ← j - 1
v = j
return list[0]

```

Algorithm #2

Min Heap, Time Cost: $O(\log k \star kn)$ & Space Cost: $O(nk)$

412. Fizz Buzz

```

result ← {ϕ}
for(i = 1 → n):
    if(mod(i, 3) == 0 & mod(i, 5) ≠ 0): result.add('fizz')
    elif(mod(i, 3) ≠ 0 & mod(i, 5) == 0): result.add('buzz')
    elif(mod(i, 15) == 0): result.add('fizzbuzz')
    else: result.add((char)(i + '0'))

```

758. Bold Words in String Google

Solution I: Time complexity: $O(n \star m \star k)$, $m = \text{dict.size}$, $k = \text{avg}(\text{word}[i].\text{length})$

```

its(Interval) = {}
for(i = 0 → n - 1):
    for(word: dict):
        if(s.startsWith(word, i)): its.add(new Interval(i, i + word.length - 1))
its = merge · intervals(its)
for(Interval it: its):
    for(i = it.start → it.end): bold[i] = true
for(i = 0 → n - 1):
    if(bold[i] and (i == n - 1 or !bold[i + 1])): result ← result + "</b>"
    result ← result + si
    if(bold[i] and (i == n - 1 or !bold[i + 1])): result ← result + "</b>"
return result

```

Optimization: since for real English words, there is no word with a length greater than 10, so we do not need to get all substring of $s[i:j]$, instead, just need to retrieve the $s[i:i] \rightarrow s[i:i+9]$, then the time complexity can be reduced from $O(n^3) \rightarrow O(nL^2)$ where $L = \max(\text{len}(\text{word}_i))$

for($i = 0 \rightarrow n - 1$):

for($l = \min(n - i, kMaxWordLen = 10) \rightarrow 1$):

if($\text{dict}[s_{i,l}] \neq \text{null}$):

for($k = i \rightarrow i + l$): $\text{bold}[k] = \text{true}$
