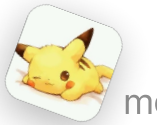
Computer Programming 2 Lab

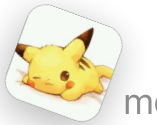
2023-03-22

Mozix Chien



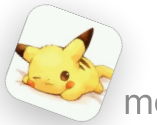
Outline

- Time Complexity
- Stack
  - Rails Problem
- Queue
  - Knight Problem
  - Binary Tree Level Order Traversal
  - Meow catch mouse



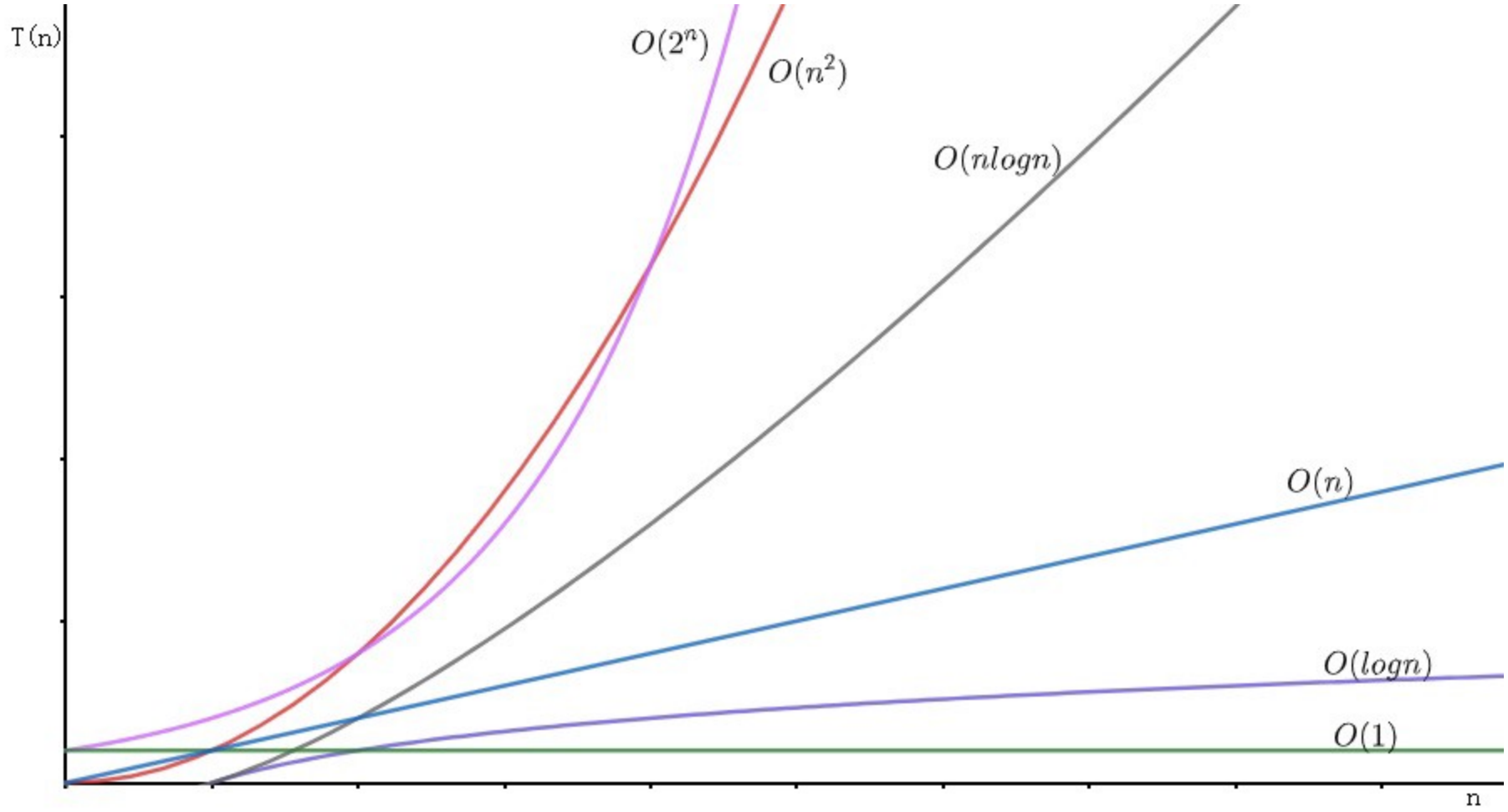
Outline

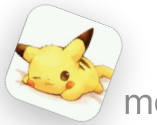
- Priority Queue (Heap)
  - Kth Largest Element in an Array
- Union Find Tree
  - Redundant Connection
- Homework & Exercise
  - HW
  - EX



Time Complexity

Time Complexity





Time Complexity

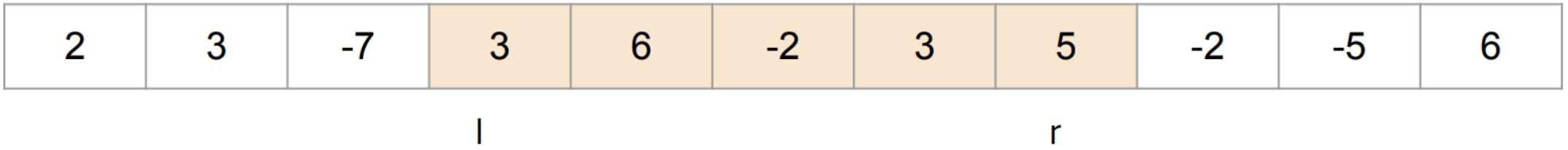
Maximum Subarray

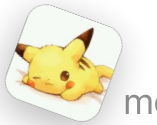
Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

- A subarray is a contiguous part of an array.

Solution 1 $O(n^3)$

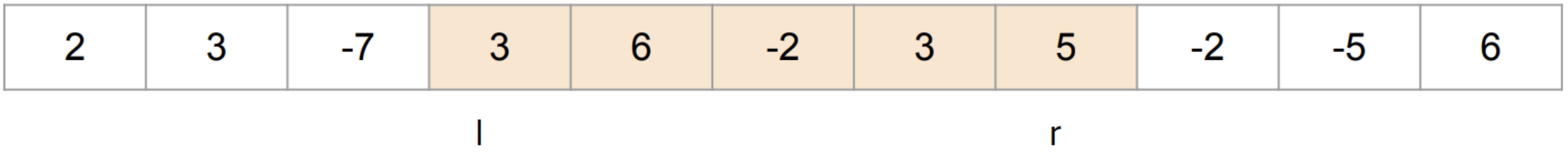
```
nums = [2, 3, -7, 3, 6, -2, 3, 5, -2, -5, 6]
n = len(nums)

ans = 0
for i in range(n):
    for j in range(i, n):
        sum = 0
        for k in range(i, j):
            sum += nums[k]
        if sum > ans:
            ans = sum
```

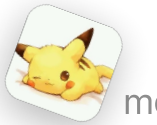




Solution 2 $O(n^2)$

```
nums = [2, 3, -7, 3, 6, -2, 3, 5, -2, -5, 6]
n = len(nums)

ans = 0
for i in range(n):
    sum = 0
    for j in range(i, n):
        sum += nums[k]
        if sum > ans:
            ans = sum
```

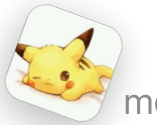




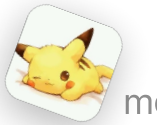
Solution 3 $O(n)$

```python
nums = [2, 3, -7, 3, 6, -2, 3, 5, -2, -5, 6]
n = len(nums)

ans, sum = 0, 0
for i in range(n):
    if sum > 0:
        sum += nums[i]
    else:
        sum = nums[i]
    ans = max(ans, sum)
```

	2	3	-7	3	6	-2	3	5	-2	-5	6
sum	2	5	-2	**3**	9	7	10	15	13	8	14
ans	**2**	**5**	5	5	**9**	9	**10**	**15**	15	15	15



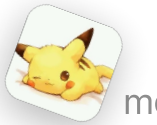
Stack



Stack

Using STL in C++

```
#include <stack>

stack<int> stk;
stk.push(2);    // push 2 into stk
stk.top();      // return 2
stk.size();     // return 1
stk.pop();      // remove the top of stack
stk.empty();    // return true
```

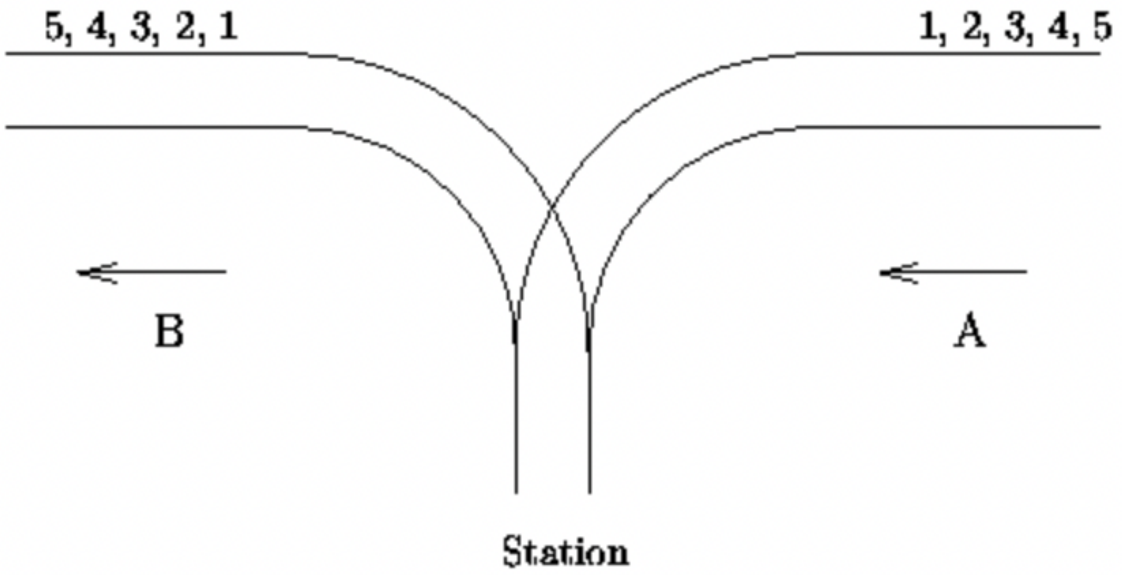


Stack

Rails Problem

Description

Given you a sequence of train cars, and you have to decide whether you can rearrange them into a valid train track.



Output

For each test case, print `Yes` if you can rearrange the cars into a valid train track, otherwise print `No`.

Input

The first line contains an integer n $(1 \leq n \leq 100)$

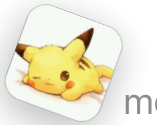
The following is n integers $a_1, a_2, \cdots, a_n$ $(1 \leq a_i \leq n)$, which is standing for the number of cars in each train.

Sample Input

```
7
4 5 3 7 6 2 1
```

Sample Output

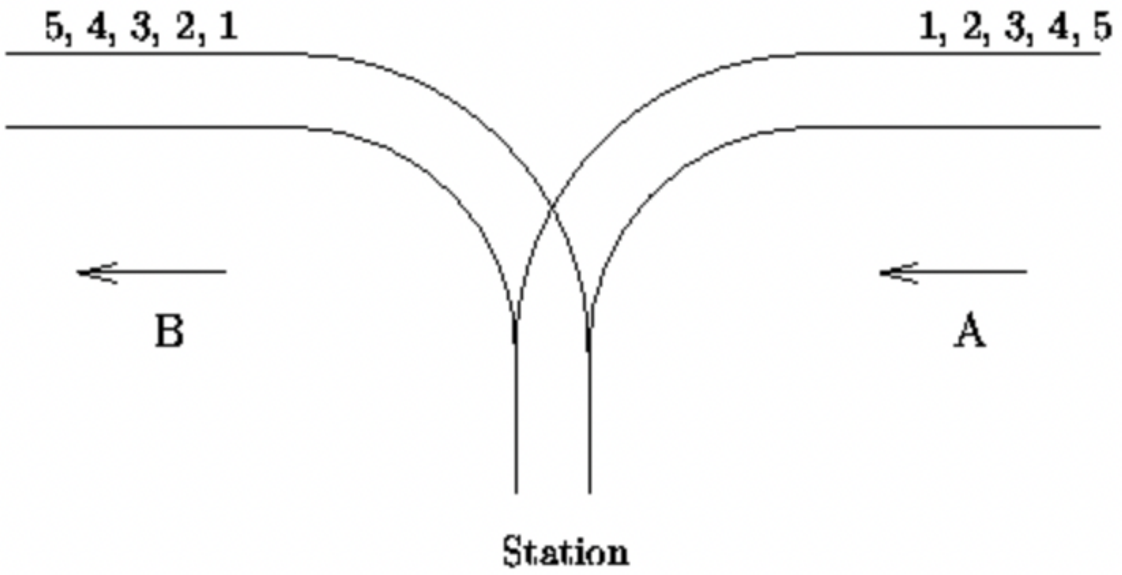
```
Yes
```

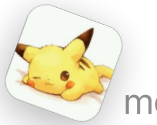


Stack

Rails Problem

- For the final valid train track, the cars should enter the track **IN ORDER**
- So that we can see that the cars can only pick from **A** or **Station**.
- Thus, we can do a simulation of the process. (But how to do it simply?)

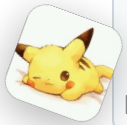




Stack

Rails Problem

```
int arr_A[1000000];
int station[1000000];
int A_ptr = 0;
int station_ptr = 0;

while(A_ptr < n){
    if(arr_A[A_ptr] == station[station_ptr]){
        // pick cars from A if A's car fit the requirement.
        A_ptr++;
        ...
    }else{
        // pick cars from station if station's car fit the requirement.
        ...
    }
    // or push cars into station if we can't find any car to fit the requirement.

    // or we can't find any car to fit the requirement.
}
```

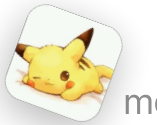


Stack

Rails Problem

```
stack<int> stkA;
stack<int> stkStation;

while(n && (stkA.top() == n || stkStation.top() == n)){
    while(!stkA.empty() && stkA.top() == n){
        stkA.pop(), n--;
    }
    while(!stkStation.empty() && stkStation.top() == n){
        stkStation.pop(), n--;
    }
    while(!stkA.empty() && stkA.top() != n){
        stkStation.push(stkA.top());
        stkA.pop();
    }
}
```

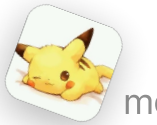
Queue



Queue

Using STL in C++

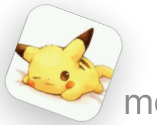
```
#include <queue>

queue<int> q;
stk.push(2);    // push 2 into q [ 2 ]
stk.push(5);    // push 5 into q [ 2 5 ]
stk.front();    // return 2
stk.size();     // return 2
stk.pop();      // remove the first elm of queue [ 5 ]
stk.empty();    // return false
```



Queue

BFS(Breadth First Search)

- We usually use Queue to implement BFS method.
- We can also use Queue to implement a **Level Order Traversal** of a tree.

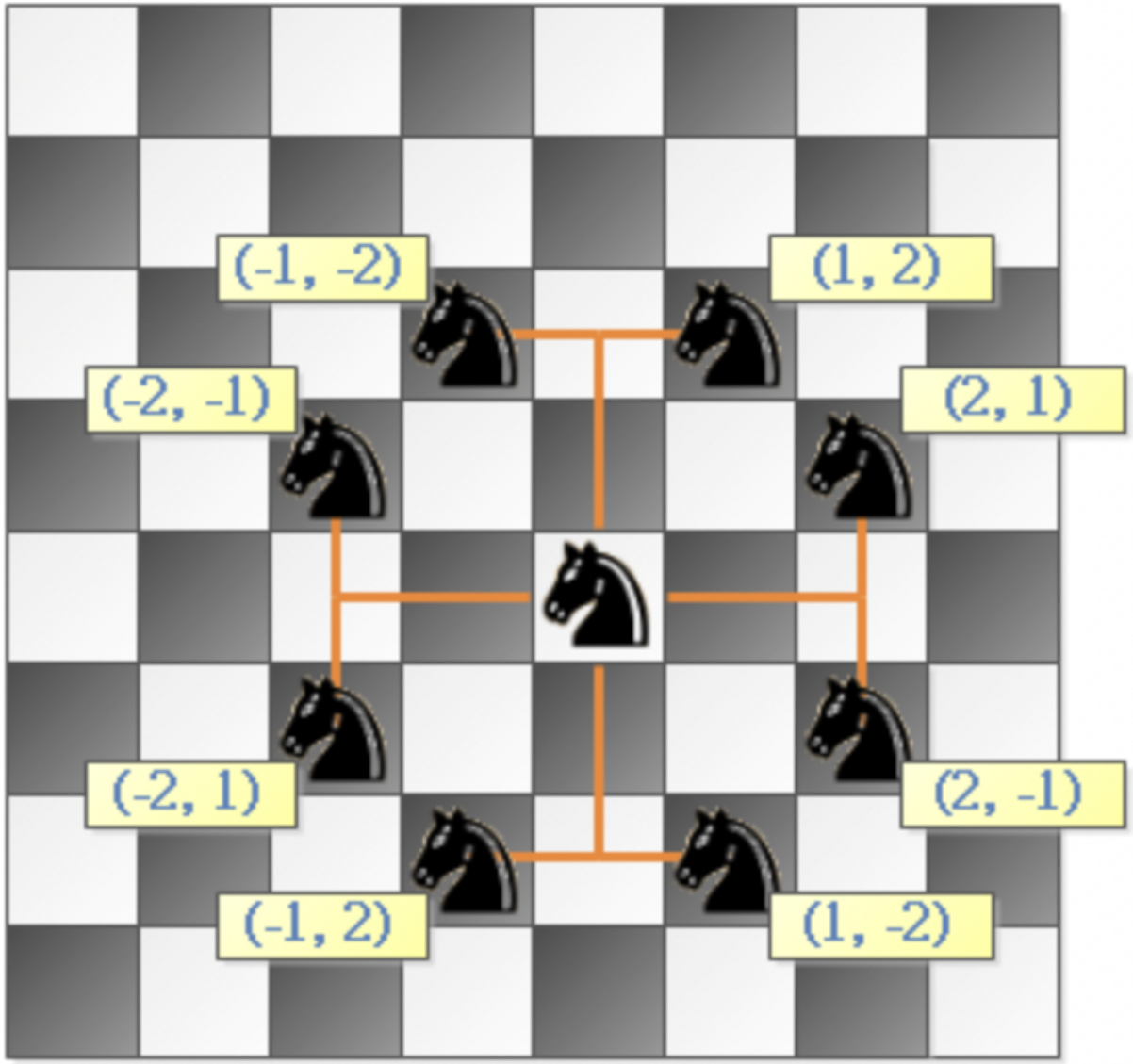


Queue

Knight Problem

Description

Given you the start point and the end point of a knight in a chessboard, and you have to find the shortest path number from the start point to the end point.



Input

The first line contains 2 integers, which indicate the location of the start point.
The second line contains 2 integers, which indicate the location of the end point.

Output

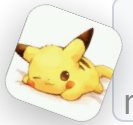
the shortest path number from the start point to the end point. or `NO ANSWER` if there can no be reach in 7 path.

Sample Input

```
4 3
7 6
```

Sample Output

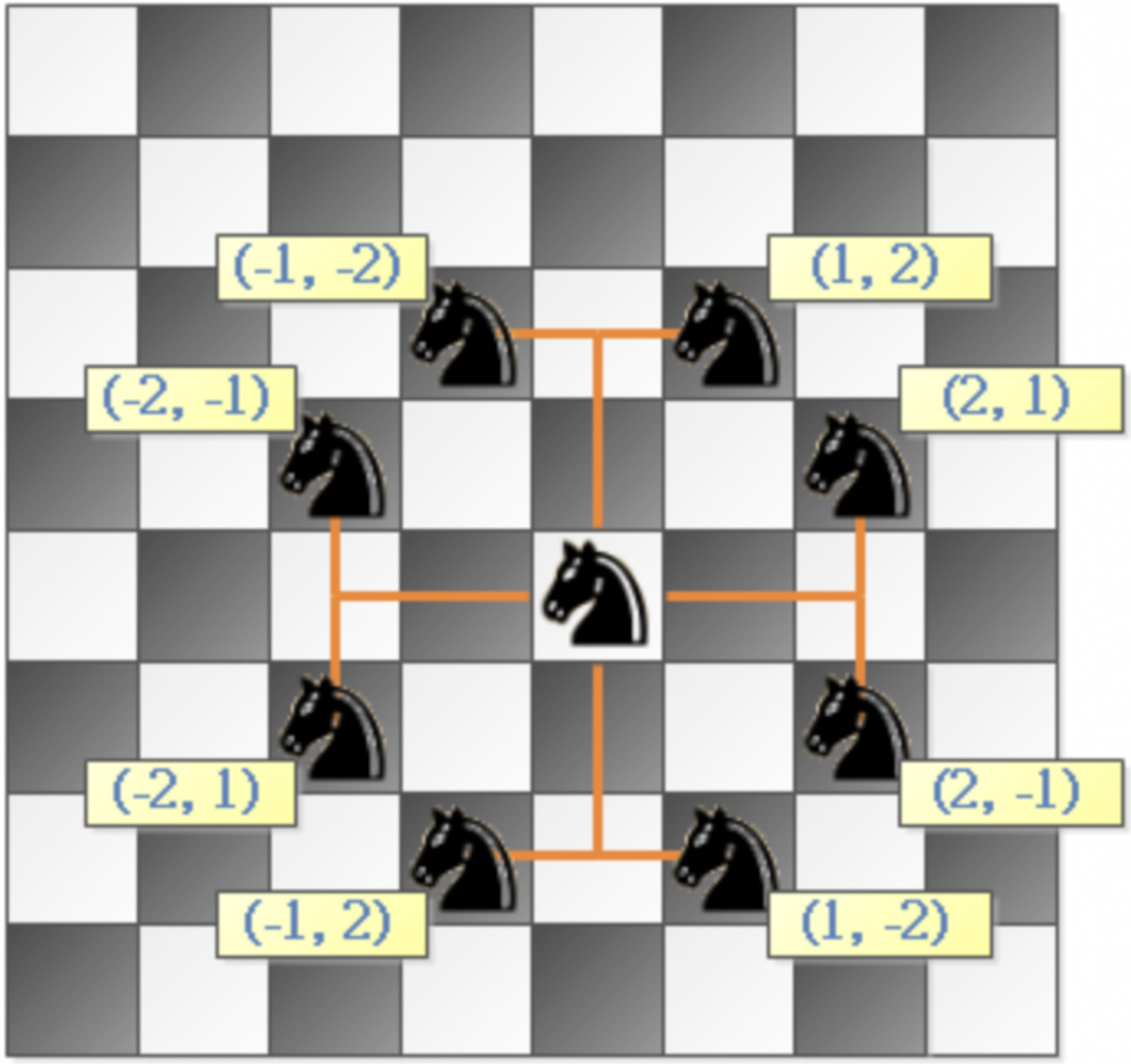
```
2
```

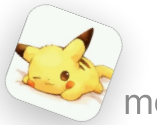


Queue

Knight Problem

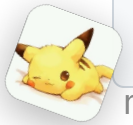
- We can first notice that there is no possible to output `NO ANSWER`
- Because of the limit of the path number is 7, even if we try all the possible route, paths we tried will not be too large.
- Thus, we can try Brutal Force with BFS to solve this problem.





Knight Problem

```
int dx[8] = {1, 1, 2, 2, -1, -1, -2, -2};
int dy[8] = {2, -2, 1, -1, 2, -2, 1, -1};

void bfs(int x, int y){
    queue<pair<int, int>> q;
    q.push({x, y});
    while(!q.empty()){
        pair<int, int> cur = q.front();
        q.pop();
        for(int i = 0; i < 8; i++){
            int nx = cur.first + dx[i];
            int ny = cur.second + dy[i];
            if(nx < 0 || nx >= 8 || ny < 0 || ny >= 8) continue;
            if(vis[nx][ny]) continue;
            vis[nx][ny] = vis[cur.first][cur.second] + 1;
            q.push({nx, ny});
        }
    }
}
```

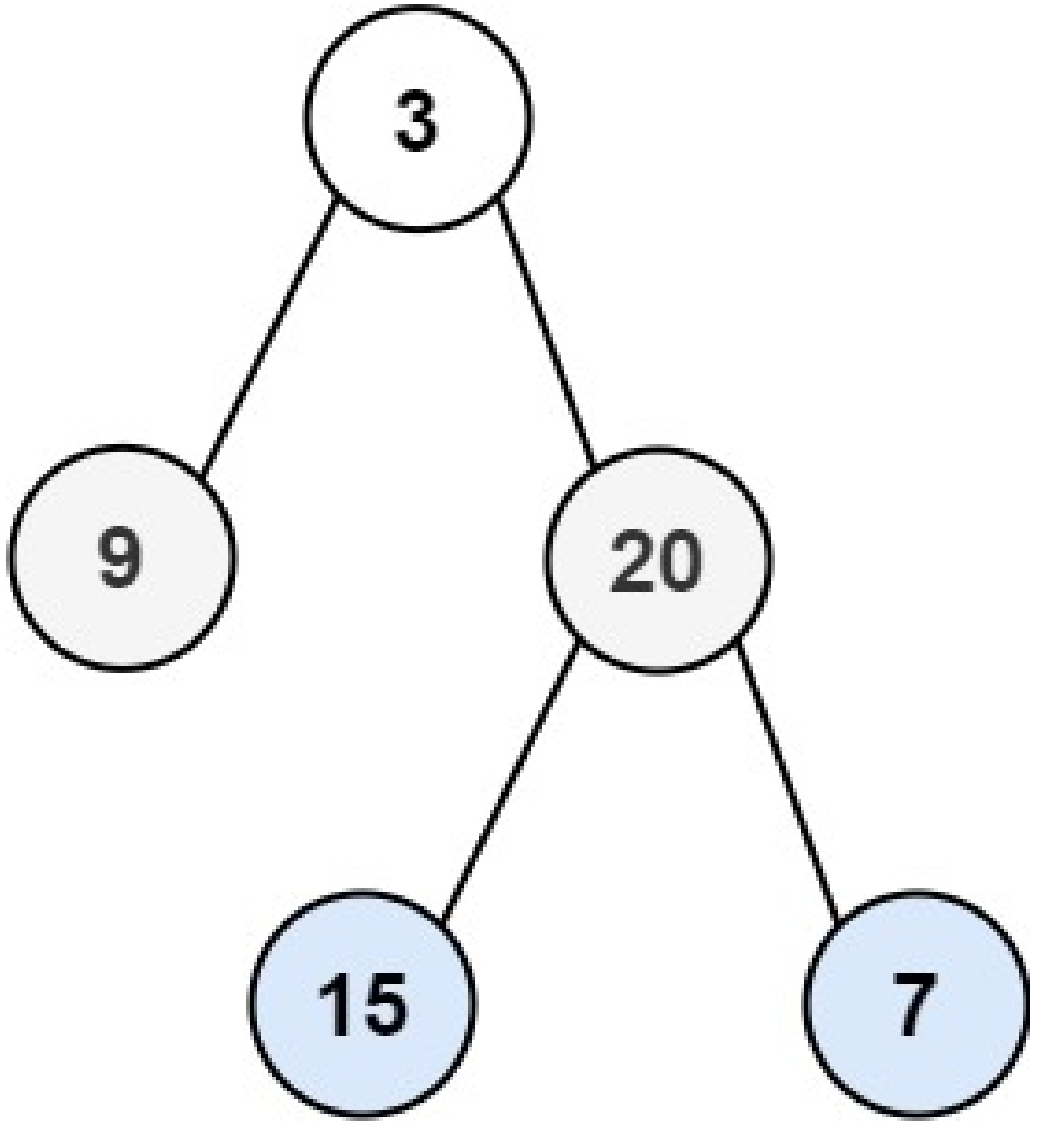


Queue

Binary Tree Level Order Traversal

Description

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).



Queue

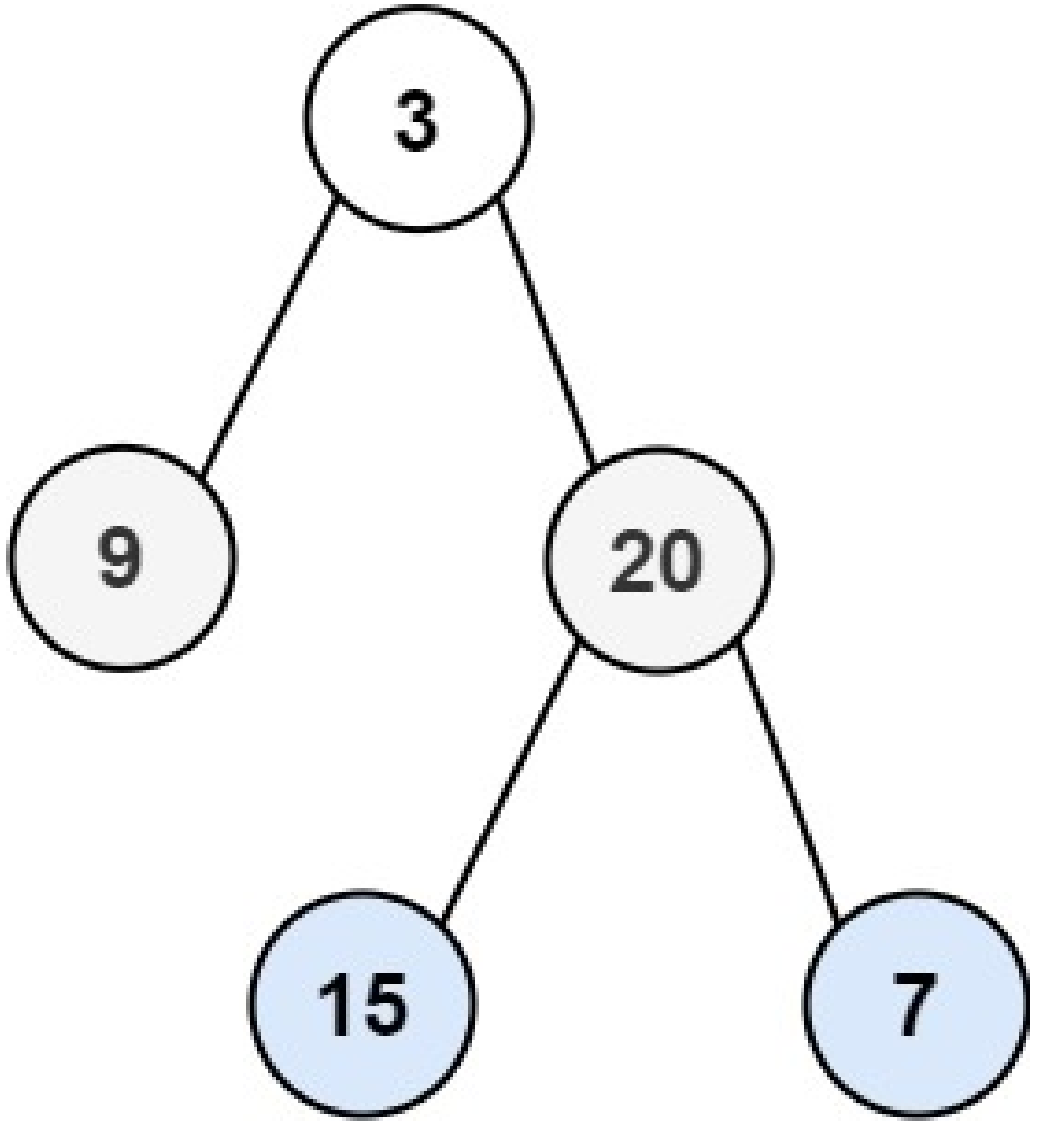
Binary Tree Level Order Traversal

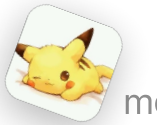
Input:

```
root = [3,9,20,null,null,15,7]
```

Output:

```
[[3],[9,20],[15,7]]
```

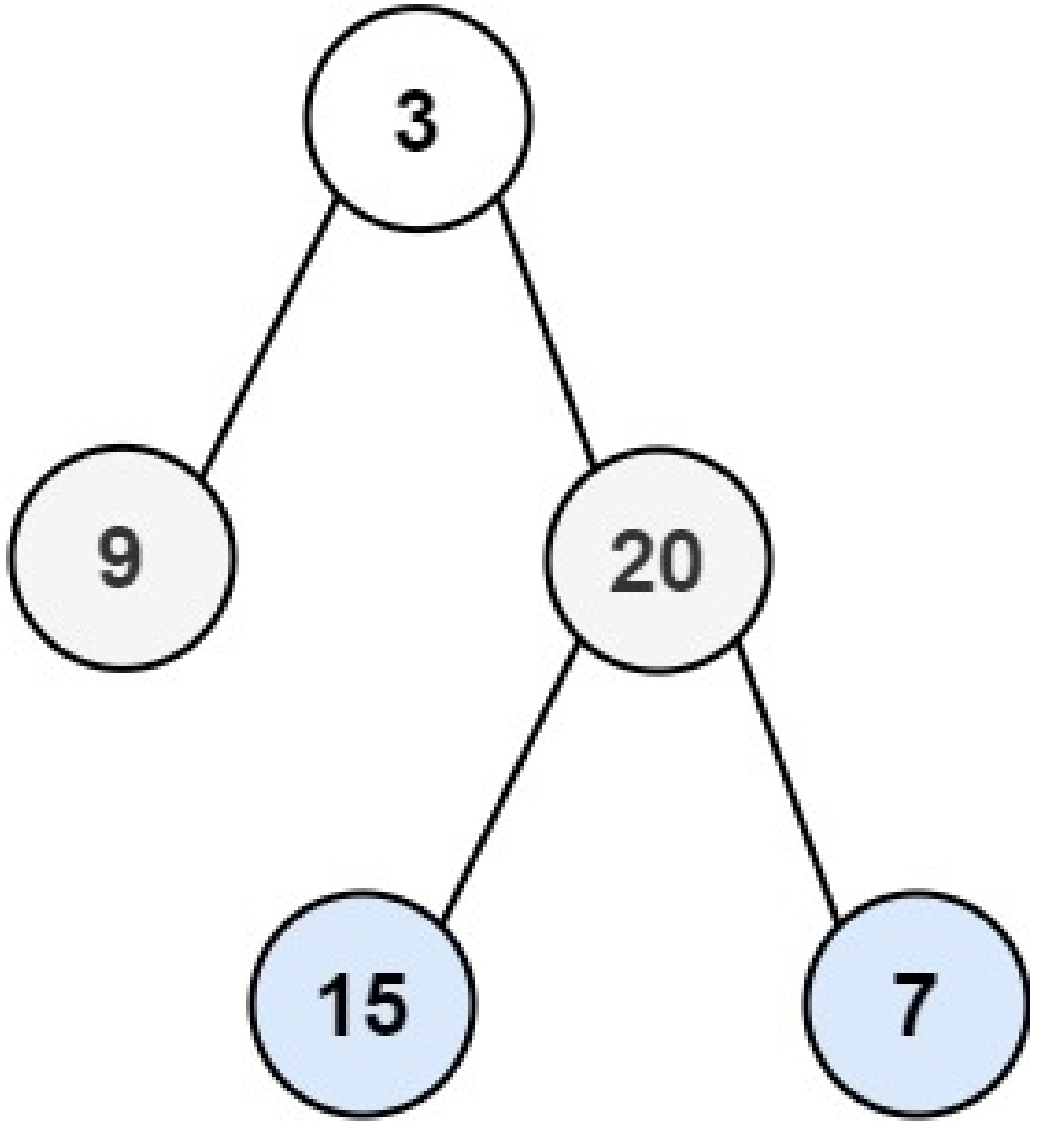


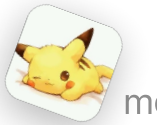


Queue

Binary Tree Level Order Traversal

- We can see the root of the tree as the start point as `Knight Problem`.
- Then, we can just use BFS to solve this problem.

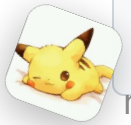




Queue

Binary Tree Level Order Traversal

```
vector<vector<int>> res;
if (!root) return res;

queue<TreeNode*> q{{root}};
while (!q.empty()) {
    vector<int> oneLevel;
    for (int i = q.size(); i > 0; --i) {
        auto t = q.front(); q.pop();
        oneLevel.push_back(t->val);
        if (t->left) q.push(t->left);
        if (t->right) q.push(t->right);
    }
    res.push_back(oneLevel);
}
return res;
```

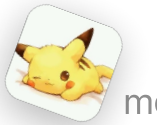


Queue

Meow catch mouse

Description

Given you a graph included the mouse and the cat, and you have to find the shortest path number from the start point to the end point.



Queue

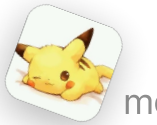
Meow catch mouse

Sample Input

```
##########
#K......@#
##########
```

Sample Output

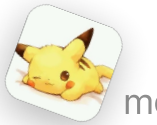
```
7
```



Queue

Meow catch mouse

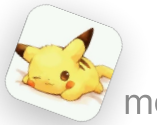
- As we have done above, we can see `K` as the start point, and `@` as the end point.
- Trying to use BFS to solve this problem.



Priority Queue (Heap)

堆積的機制

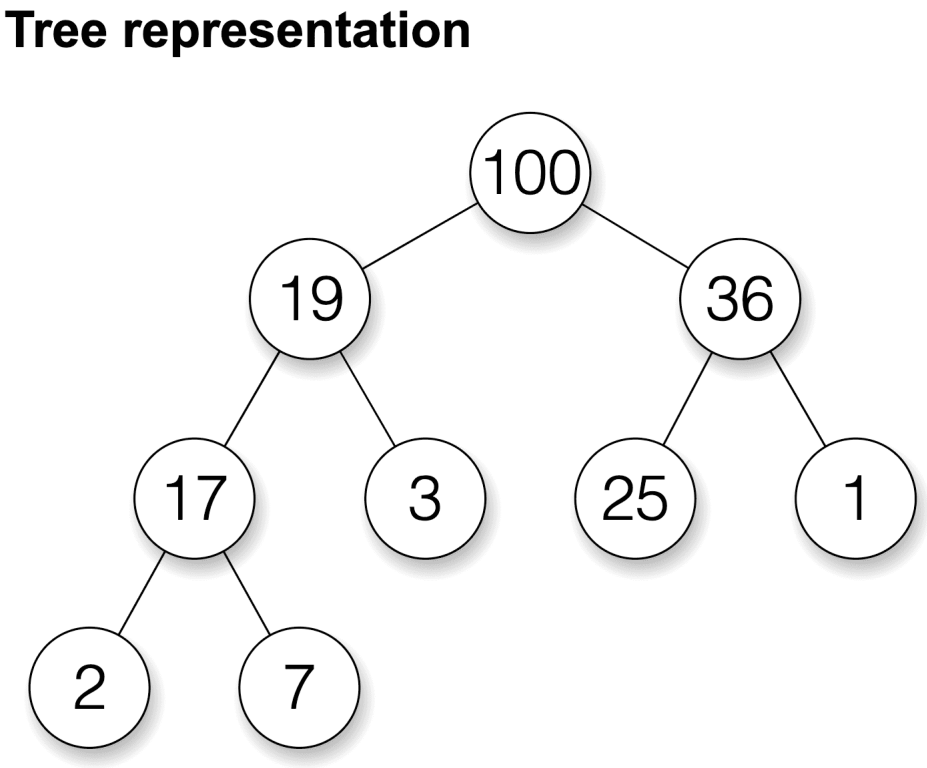
1. 可以追加數值
2. 可以取出最大(小)值
    - 大頂堆
    - 小頂堆
3. 可以刪除最大(小)值



Priority Queue (Heap)

二元堆積

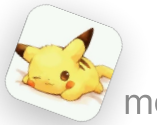
- 將堆積以二元樹實踐
- 只在乎二元樹的 root



Priority Queue (Heap)

堆積實作

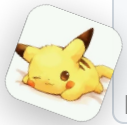
1. 節點編號由 0 開始
    - 左子節點的編號為 2*n
    - 右子節點的編號為 2*n+1
2. 考慮新增節點時的交換比較
3. 考慮刪除節點時的交換比較



Priority Queue (Heap)

堆積 STL

```cpp
#include <queue>

int main(void){
    priority_queue<int> pq; // 預設大頂堆
    priority_queue<int, vector<int>, greater<int>> spq // 小頂堆
    priority_queue<int, vector<int>, cmp> spq // 自定義堆積

    pq.push(5);
    pq.push(1);
    pq.push(3);

    while(!pq.empty()){
        cout << pq.top() << endl;
        pq.pop();
    }
}
```

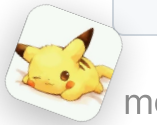


Priority Queue (Heap)

自定義堆積 in STL

- 使用 struct
- overload operator()

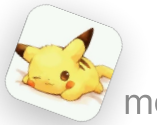
```
struct cmp {
    bool operator()(node a, node b) {
        /*
        priority_queue優先判定為!cmp，所以「由大排到小」需「反向」定義
        實現「最小值優先」
        */
        return a.x < b.x;
    }
};
```



Union Find Tree

Union Find Tree

- 劃分群組進行管理的資料結構
- find(a, b): 查詢 a, b 是否為屬於同一個群組
- union(a, b): 合併 a, b 所屬的群組



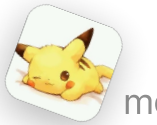
Union Find Tree

初始化

- 每個點初始時都指向自己 (自己獨立是一個集合)

合併

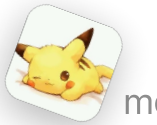
- 將兩集合樹根相連即可做到合併



Union Find Tree

判斷

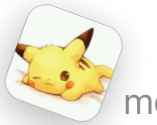
- 判斷時，回朔點的樹根
  - 如果樹根相同，則為同一個集合
  - 如果樹根不同，則為相異的集合(可以做合併)



Union Find Tree

注意事項

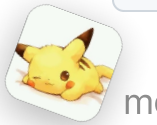
- 樹歪斜，導致執行效率差
    i. 樹高小的連上樹高大的上
    ii. 回朔時，同步更新所有點的根節點，藉此減少樹高



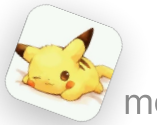
Union Find Tree

Implementation

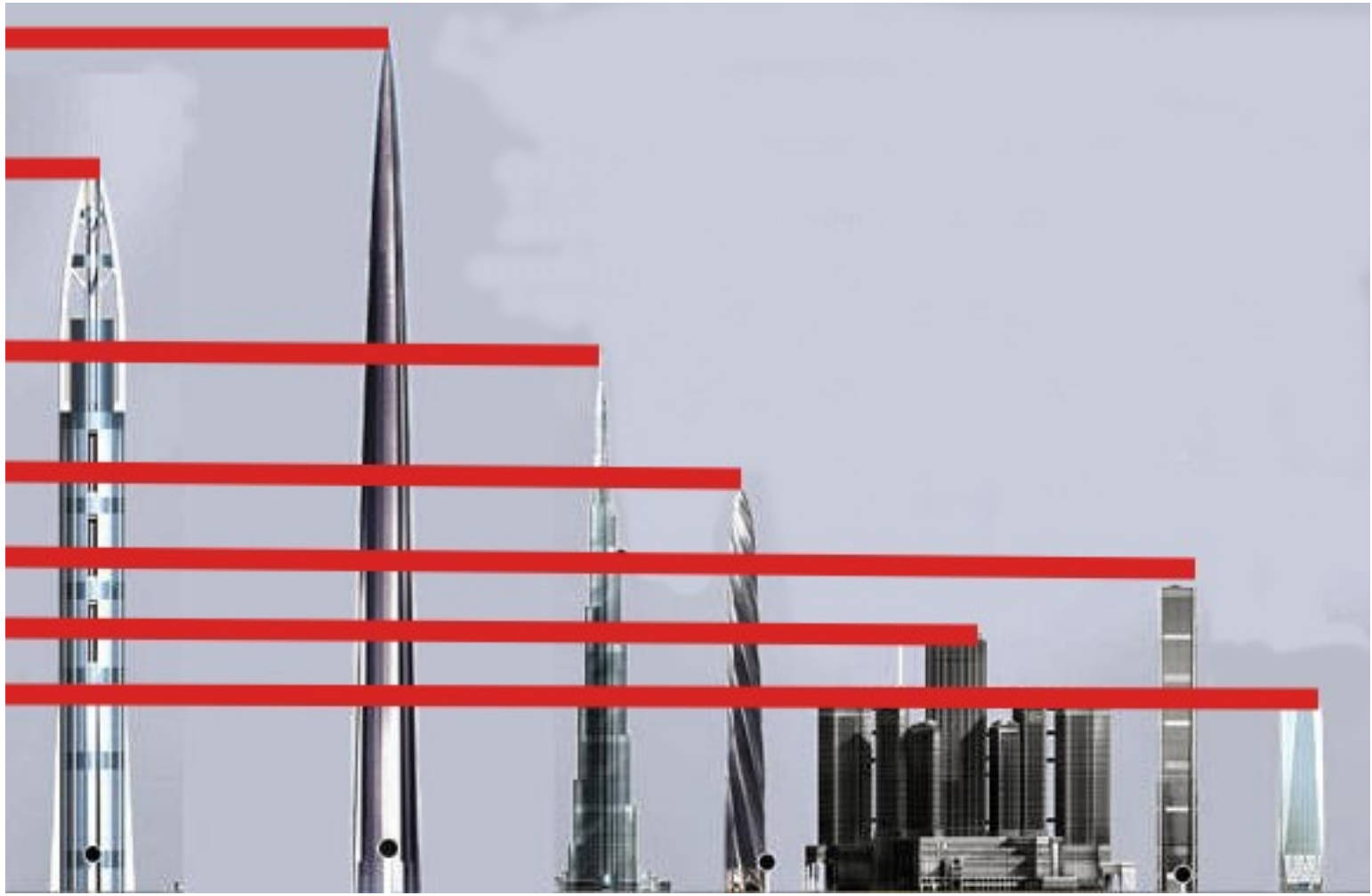
```
int Find (int a) {
    if (a == parent[a])
        return a;
    return parent[a] = Find (parent[a]);
}

void Union (int a, int b) {
    int pa = Find (a);
    int pb = Find (b);
    if (pa != pb)
        parent[pa] = pb;
}
```



Homework & Exercise



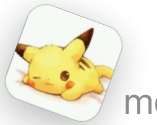
Galileo's Free Fall Experiment



Galileo's Free Fall Experiment

Target

- Find out all the buildings' vision
- A building's vision is the number of buildings that can be seen from the top of the building.
- A building can be seen which means that between the building where Galileo is located and the building to be seen, the height of other buildings is smaller than the two buildings.
- $min(building[i], building[j]) > max(building[i+1], building[i+2], ..., building[j-1])$



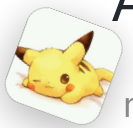
Galileo's Free Fall Experiment

Example

Given the height of the buildings: [9, 12, 7, 5, 3, 4, 2]

standing on the building 0: he can see the building 1
standing on the building 1: he can see the building 2
standing on the building 2: he can see the building 3
standing on the building 3: he can see the building 4 and 5
standing on the building 4: he can see the building 5
standing on the building 5: he can see the building 6
standing on the building 6: he can see nothing...Orz

As a result, all the buildings' vision are: [1, 1, 1, 2, 1, 1, 0]



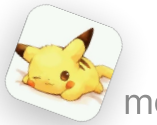
Galileo's Free Fall Experiment

Input

First line has an integer n, which is the number of buildings $(1 \leq n \leq 10^5)$
The second line has n integers, which are the height of buildings $(1 \leq height \leq 10^5)$

Output

the buildings' vision from left to right.
The output should be separated by a space, and after the last number, there should be a newline.



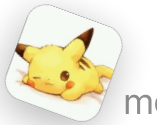
Galileo's Free Fall Experiment

Sample Input

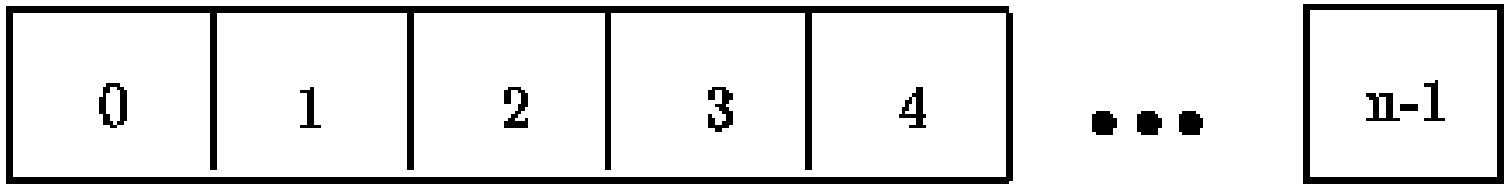
```
7
9 12 7 5 3 4 2
```

Sample Output

```
1 1 1 2 1 1 0
```



搬積木



Any Questions?