1. Sort Colors

   We will iterate through our nums array until our current index is larger than our minimum 2’s index. If current number is 0 and 0’s rightest bound index is not equal to current index, we will replace the number, move 0’s bound to right by one step. The same with 2. If it’s 1, we will move index right by 1. In this way, we could sort our ballens in place in one pass.

```
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        length = len(nums)
        i = 0
        j, k = 0, len(nums) - 1
        while i <= k:
            if nums[i] == 0 and i != j:
                nums[i], nums[j] = nums[j], nums[i]
                j += 1
            elif nums[i] == 2 and i != k:
                nums[i], nums[k] = nums[k], nums[i]
                k -= 1
            else:
                i += 1
```

528. Random Pick with Weight

We will accumulate all number previously to the current position. Then we could Random pick a int from 1 to last num in the array. Then use binary search to get it’s index it belongs. That’s what we need. TC is n, logn

```
from bisect import *
from random import *
class Solution:

    def __init__(self, w: List[int]):
        self.w = w
        self.length = len(w)
        for i in range(1, len(w)):
            self.w[i] += self.w[i - 1]

    def pickIndex(self) -> int:
        num = randint(1, self.w[-1])
        return bisect_left(self.w, num)
```

3. Reconstruct **Itinerary:**

We will build a dictinary and store every path into it, (key, list), Every time we get the next node, we will pop from that dictionary’s list. Until we cannot find next one. That’s what we want. TC O(n)

```
from collections import defaultdict
def findDest(nodes, start):
    path = defaultdict(list)
    result = [start]
    for node in nodes:
        path[node[0]].append(node[1])
    while start in path and path[start]:
        start = path[start].pop()
        result.append(start)
    return result

nodes = [['A', 'B'], ['B', 'C'], ['C', 'D'], ['B', 'A'],['A', 'B']]
print(findDest(nodes, 'A'))
```

4, Find Multiple Dest

We will use dfs to find dest, for every node there is binary way, we will dfs for each way until the end of the path. We will add end node to our result. We will use visited to prevent iterate through one node multiple times.

```
from collections import defaultdict
def findDest(nodes, start):
    path = defaultdict(list)
    result = set()
    visited = set([start])
    for node in nodes:
        path[node[0]].append(node[1])
    dfs(visited, path, [start], result)
    return result
def dfs(visited, path, cur, result):
    if cur[-1] not in path:
        result.add(cur[-1])
        return
    for next_node in path[cur[-1]]:
        if next_node not in visited:
            visited.add(next_node)
            dfs(visited, path, cur + [next_node], result)
            visited.remove(next_node)


nodes = [['A', 'B'], ['B', 'C'], ['C', 'D'], ['B', 'A'],['A', 'B'], ['B', 
 'F']]
 print(findDest(nodes, 'A'))
```

5. WaitingList

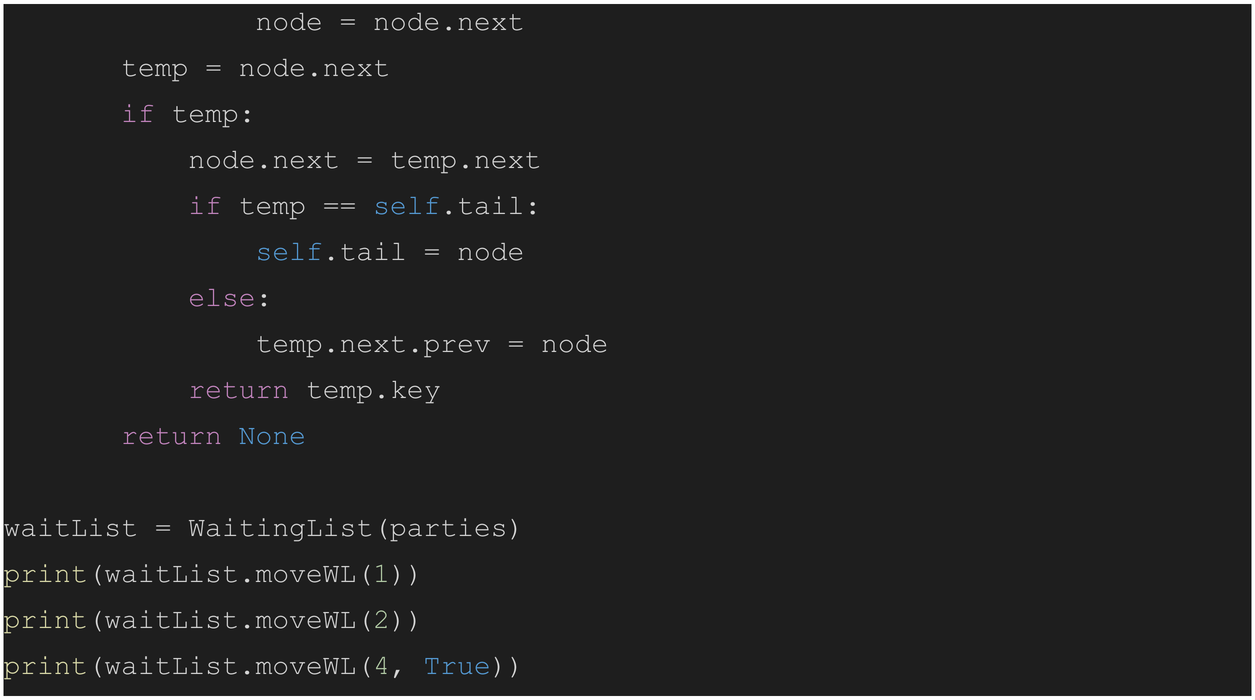
We will use a double linked list and store each party as a node. When move party to linked linked list, we will append it directly to self.tail. And let tail equal to next node. When find move line. We will find one node with value less or equal to current capacity, and delete it from our linked list. TC is O(1) and O(n).

```
class Node:
    def __init__(self, key, val):
        self.val = val
        self.key = key
        self.prev = None
        self.next = None

class WaitingList:
    def __init__(self, parties):
        self.dummy = Node(0, 0)
        self.tail = self.dummy
        for party in parties:
            self.addPartyToWL(party)

    def addPartyToWL(self, party):
        self.tail.next = Node(party["label"], party["number"])
        self.tail.next.prev = self.tail
        self.tail = self.tail.next

    def moveWL(self, num, strict=False):
        node = self.dummy
        if strict:
            while node.next and node.next.val != num:
                node = node.next
        else:
            while node.next and node.next.val > num:
```



```
                node = node.next
        temp = node.next
        if temp:
            node.next = temp.next
            if temp == self.tail:
                self.tail = node
            else:
                temp.next.prev = node
            return temp.key
        return None

waitList = WaitingList(parties)
print(waitList.moveWL(1))
print(waitList.moveWL(2))
print(waitList.moveWL(4, True))
```