599. Minimum Index Sum of Two Lists
This question is quite easy. We only need to iterate through list1 and use a map to store every restaurant's name. Then we will iterate through list2, check whether the restaurant is in map. If it's there, we will compare ind + map[r] with min_ind. In the end, we will get what we want. TC is O(1).

```
def find_restaurant(list1, list2)
    memo = {}
    min_ind = 2 ** 31
    result = []

    list1.each_with_index do |r, ind|
        memo[r] = ind
    end

    list2.each_with_index do |r, ind|
        if memo[r]
            if memo[r] + ind < min_ind
                result = [r]
                min_ind = memo[r] + ind
            elsif memo[r] + ind == min_ind
                result.push(r)
            end
        end
    end

    result
End
```

347. Top K Frequent Elements
This question is quite easy. Nothing to say. O(nlogn) n is number of unique number.

```
def top_k_frequent(nums, k)
    memo = Hash.new(0)
    result = []
    nums.each do |num|
        memo[num] += 1
    end

    memo.sort_by { |_, v| -v }.map {|a| a[0]}[0, k]
end
```

692. Top K Frequent Words
Nothing to say. Quite similar to the previous one.

```
def top_k_frequent(words, k)
    memo = Hash.new(0)

    words.each do |w|
        memo[w] += 1
    end

    memo.keys().sort do |a, b|
        temp = memo[b] <=> memo[a]
        if temp == 0
          a <=> b
        else
          temp
        end
    end[0, k]
end
```

332. Reconstruct Itinerary
Use other ways which is pretty cool.

```
def find_itinerary(tickets)
    memo = Hash.new([])
    visited = Hash.new(false)

    tickets = tickets.sort.reverse.group_by(&:first)
    puts tickets
    route, stack =  [], ['JFK']
    while !stack.empty?
        stack.push tickets[stack[-1]].pop()[1] while (tickets[stack[-1]] || []).any?
        route << stack.pop()
    end
    route.reverse
End
```

104. Maximum Depth of Binary Tree
Quite simple, nothing to say. O(depth).

```
def max_depth(root)
    left = 0
    right = 0
```

```
    if root
        left = max_depth(root.left)
        right = max_depth(root.right)
    else
        return 0
    end
    return [left, right].max + 1
end
```