**Technical interview questions**

|  |  |
| --- | --- |
| ⚠️ | The answers here are given by the community. Be careful and double check the answers before using them. If you see an error, please create a PR with a fix |

The list is based on [this post](https://medium.com/data-science-insider/technical-data-science-interview-questions-f61cd9cf218?source=friends_link&sk=01f4de0de746d28fe714d92a1e91e190)

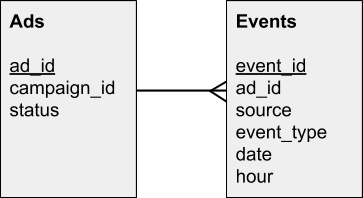
**Table of contents**

* [SQL](https://github.com/alexeygrigorev/data-science-interviews/blob/master/technical.md?fbclid=IwAR1H6QBRkuVaJ3zp0o1Jaf2hN26P8xATJlmSFv-gFmnZiKd0GI359jFQ5Vo#sql)
* [Coding (Python)](https://github.com/alexeygrigorev/data-science-interviews/blob/master/technical.md?fbclid=IwAR1H6QBRkuVaJ3zp0o1Jaf2hN26P8xATJlmSFv-gFmnZiKd0GI359jFQ5Vo#coding-python)
* [Algorithmic Questions](https://github.com/alexeygrigorev/data-science-interviews/blob/master/technical.md?fbclid=IwAR1H6QBRkuVaJ3zp0o1Jaf2hN26P8xATJlmSFv-gFmnZiKd0GI359jFQ5Vo#algorithmic-questions)

**SQL**

Suppose we have the following schema with two tables: Ads and Events

* Ads(ad\_id, campaign\_id, status)
* status could be active or inactive
* Events(event\_id, ad\_id, source, event\_type, date, hour)
* event\_type could be impression, click, conversion

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/schema.png)

Write SQL queries to extract the following information:

**1)** The number of active ads.

SELECT count(\*) FROM Ads WHERE status = 'active';

**2)** All active campaigns. A campaign is active if there’s at least one active ad.

SELECT DISTINCT a.campaign\_id

FROM Ads AS a

WHERE a.status = 'active';

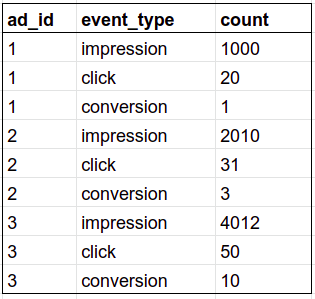
**3)** The number of active campaigns.

SELECT COUNT(DISTINCT a.campaign\_id)

FROM Ads AS a

WHERE a.status = 'active';

**4)** The number of events per each ad — broken down by event type.

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_4_example.png)

SELECT a.ad\_id, e.event\_type, count(\*) as "count"

FROM Ads AS a

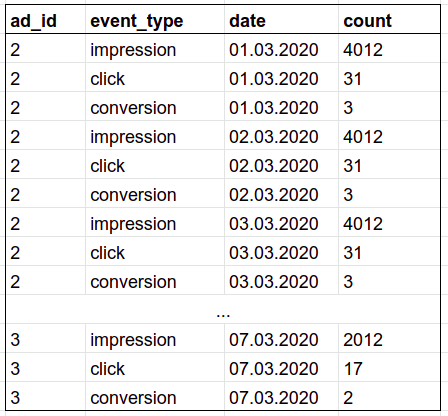
JOIN Events AS e

ON a.ad\_id = e.ad\_id

GROUP BY a.ad\_id, e.event\_type

ORDER BY a.ad\_id, "count" DESC;

**5)** The number of events over the last week per each active ad — broken down by event type and date (most recent first).

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_5_example.png)

SELECT a.ad\_id, e.event\_type, e.date, count(\*) as "count"

FROM Ads AS a

JOIN Events AS e

ON a.ad\_id = e.ad\_id

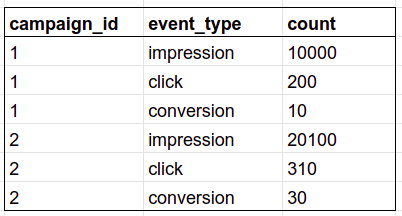
WHERE a.status = 'active'

AND e.date >= DATEADD(week, -1, GETDATE())

GROUP BY a.ad\_id, e.event\_type, e.date

ORDER BY e.date ASC, "count" DESC;

**6)** The number of events per campaign — by event type.

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_6_example.png)

SELECT a.campaign\_id, e.event\_type, count(\*) as count

FROM Ads AS a

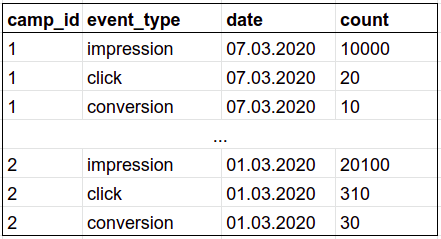
INNER JOIN Events AS e

ON a.ad\_id = e.ad\_id

GROUP BY a.campaign\_id, e.event\_type

ORDER BY a.campaign\_id, "count" DESC

**7)** The number of events over the last week per each campaign and event type — broken down by date (most recent first).

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_7_example.png)

-- for Postgres

SELECT a.campaign\_id, e.event\_type, e.date, count(\*)

FROM Ads AS a

INNER JOIN Events AS e

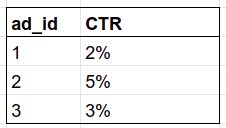
ON a.ad\_id = e.ad\_id

WHERE e.date >= DATEADD(week, -1, GETDATE())

GROUP BY a.campaign\_id, e.event\_type, e.date

ORDER BY a.campaign\_id, e.date DESC, "count" DESC;

**8)** CTR (click-through rate) for each ad. CTR = number of clicks / number of impressions.

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_8_example.png)

-- for Postgres

SELECT impressions\_clicks\_table.ad\_id,

(impressions\_clicks\_table.clicks \* 100 / impressions\_clicks\_table.impressions)::FLOAT || '%' AS CTR

FROM

(

SELECT a.ad\_id,

SUM(CASE e.event\_type WHEN 'impression' THEN 1 ELSE 0 END) impressions,

SUM(CASE e.event\_type WHEN 'click' THEN 1 ELSE 0 END) clicks

FROM Ads AS a

INNER JOIN Events AS e

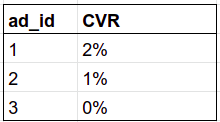
ON a.ad\_id = e.ad\_id

GROUP BY a.ad\_id

) AS impressions\_clicks\_table

ORDER BY impressions\_clicks\_table.ad\_id;

**9)** CVR (conversion rate) for each ad. CVR = number of conversions / number of clicks.

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_9_example.png)

-- for Postgres

SELECT conversions\_clicks\_table.ad\_id,

(conversions\_clicks\_table.conversions \* 100 / conversions\_clicks\_table.clicks)::FLOAT || '%' AS CVR

FROM

(

SELECT a.ad\_id,

SUM(CASE e.event\_type WHEN 'conversion' THEN 1 ELSE 0 END) conversions,

SUM(CASE e.event\_type WHEN 'click' THEN 1 ELSE 0 END) clicks

FROM Ads AS a

INNER JOIN Events AS e

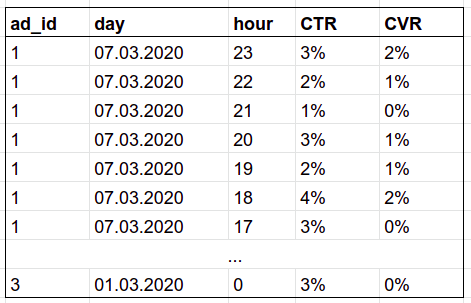
ON a.ad\_id = e.ad\_id

GROUP BY a.ad\_id

) AS conversions\_clicks\_table

ORDER BY conversions\_clicks\_table.ad\_id;

**10)** CTR and CVR for each ad broken down by day and hour (most recent first).

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_10_example.png)

-- for Postgres

SELECT conversions\_clicks\_table.ad\_id,

conversions\_clicks\_table.date,

conversions\_clicks\_table.hour,

(impressions\_clicks\_table.clicks \* 100 / impressions\_clicks\_table.impressions)::FLOAT || '%' AS CTR,

(conversions\_clicks\_table.conversions \* 100 / conversions\_clicks\_table.clicks)::FLOAT || '%' AS CVR

FROM

(

SELECT a.ad\_id, e.date, e.hour,

SUM(CASE e.event\_type WHEN 'conversion' THEN 1 ELSE 0 END) conversions,

SUM(CASE e.event\_type WHEN 'click' THEN 1 ELSE 0 END) clicks,

SUM(CASE e.event\_type WHEN 'impression' THEN 1 ELSE 0 END) impressions

FROM Ads AS a

INNER JOIN Events AS e

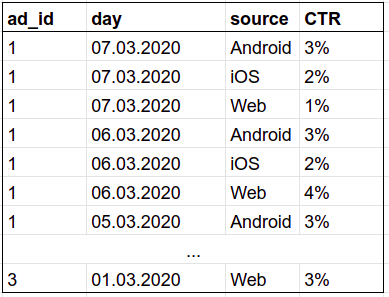
ON a.ad\_id = e.ad\_id

GROUP BY a.ad\_id, e.date, e.hour

) AS conversions\_clicks\_table

ORDER BY conversions\_clicks\_table.ad\_id, conversions\_clicks\_table.date DESC, conversions\_clicks\_table.hour DESC, "CTR" DESC, "CVR" DESC;

**11)** CTR for each ad broken down by source and day

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/sql_11_example.png)

-- for Postgres

SELECT conversions\_clicks\_table.ad\_id,

conversions\_clicks\_table.date,

conversions\_clicks\_table.source,

(impressions\_clicks\_table.clicks \* 100 / impressions\_clicks\_table.impressions)::FLOAT || '%' AS CTR

FROM

(

SELECT a.ad\_id, e.date, e.source,

SUM(CASE e.event\_type WHEN 'click' THEN 1 ELSE 0 END) clicks,

SUM(CASE e.event\_type WHEN 'impression' THEN 1 ELSE 0 END) impressions

FROM Ads AS a

INNER JOIN Events AS e

ON a.ad\_id = e.ad\_id

GROUP BY a.ad\_id, e.date, e.source

) AS conversions\_clicks\_table

ORDER BY conversions\_clicks\_table.ad\_id, conversions\_clicks\_table.date DESC, conversions\_clicks\_table.source, "CTR" DESC;

**Coding (Python)**

**1) FizzBuzz.** Print numbers from 1 to 100

* If it’s a multiplier of 3, print “Fizz”
* If it’s a multiplier of 5, print “Buzz”
* If both 3 and 5 — “Fizz Buzz"
* Otherwise, print the number itself

Example of output: 1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, Fizz Buzz, 31, 32, Fizz, 34, Buzz, Fizz, ...

for i in range(1, 101):

if i % 3 == 0 and i % 5 == 0:

print('Fizz Buzz')

elif i % 3 == 0:

print('Fizz')

elif i % 5 == 0:

print('Buzz')

else:

print(i)

**2) Factorial**. Calculate a factorial of a number

* factorial(5) = 5! = 1 \* 2 \* 3 \* 4 \* 5 = 120
* factorial(10) = 10! = 1 \* 2 \* 3 \* 4 \* 5 \* 6 \* 7 \* 8 \* 9 \* 10 = 3628800

def factorial(n):

result = 1

for i in range(2, n + 1):

result \*= i

return result

We can also write this function using recursion:

def factorial(n: int):

if n == 0 or n == 1:

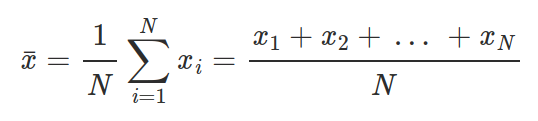
return 1

else:

return n \* factorial(n - 1)

**3) Mean**. Compute the mean of number in a list

* mean([4, 36, 45, 50, 75]) = 42
* mean([]) = NaN (use float('NaN'))

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/formula_mean.png)

def mean(numbers):

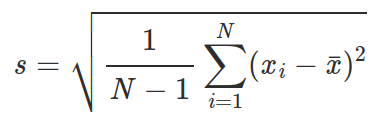
if len(numbers) > 0:

return sum(numbers) / len(numbers)

return float('NaN')

**4) STD**. Calculate the standard deviation of elements in a list.

* std([1, 2, 3, 4]) = 1.29
* std([1]) = NaN (use float('NaN'))
* std([]) = NaN

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/formula_std.png)

from math import sqrt

from statistics import mean

def std\_dev(numbers):

if len(numbers) > 1:

avg = mean(numbers)

var = sum((i - avg) \*\* 2 for i in numbers) / (len(numbers) - 1)

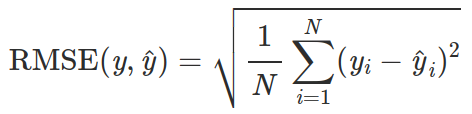
std = sqrt(var)

return std

return float('NaN')

**5) RMSE**. Calculate the RMSE (root mean squared error) of a model. The function takes in two lists: one with actual values, one with predictions.

* rmse([1, 2], [1, 2]) = 0
* rmse([1, 2, 3], [3, 2, 1]) = 1.63

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/formula_rmse.png)

import math

def rmse(y\_true, y\_pred):

assert len(y\_true) == len(y\_pred), 'different sizes of the arguments'

squares = sum((x - y)\*\*2 for x, y in zip(y\_true, y\_pred))

return math.sqrt(squares / len(y\_true))

**6) Remove duplicates**. Remove duplicates in list. The list is not sorted and the order of elements from the original list should be preserved.

* [1, 2, 3, 1] ⇒ [1, 2, 3]
* [1, 3, 2, 1, 5, 3, 5, 1, 4] ⇒ [1, 3, 2, 5, 4]

def remove\_duplicates(lst):

new\_list = []

mentioned\_values = set()

for elem in lst:

if elem not in mentioned\_values:

new\_list.append(elem)

mentioned\_values.add(elem)

return new\_list

# The above solution checks the values into a set and it is O(1) efficient using

# a few of lines.

# A shorter solution follows: it is O(n^2) but can be fine when lst has no "too

# many elements" - the quantity depends by the running box.

def remove\_duplicates2(lst):

new\_list = []

for elem in lst:

if elem not in new\_list:

new\_list.append(elem)

return new\_list

**7) Count**. Count how many times each element in a list occurs.

[1, 3, 2, 1, 5, 3, 5, 1, 4] ⇒

* 1: 3 times
* 2: 1 time
* 3: 2 times
* 4: 1 time
* 5: 2 times

numbers = [1, 3, 2, 1, 5, 3, 5, 1, 4]

counter = dict()

for elem in numbers:

counter[elem] = counter.get(elem, 0) + 1

or

from collections import Counter

numbers = [1, 3, 2, 1, 5, 3, 5, 1, 4]

counter = Counter(numbers)

**8) Palindrome**. Is string a palindrome? A palindrome is a word which reads the same backward as forwards.

* “ololo” ⇒ Yes
* “cafe” ⇒ No

def is\_palindrome(s):

return s == s[::-1]

or

def is\_palindrome(s):

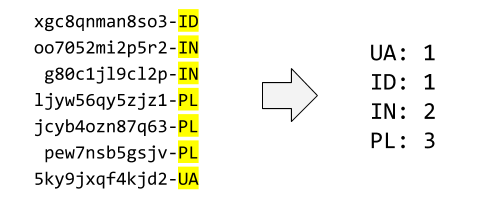
for i in range(len(s) // 2):

if s[i] != s[-i - 1]:

return False

return True

**9) Counter**. We have a list with identifiers of form “id-SITE”. Calculate how many ids we have per site.

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/counter_1.png)

def counter(lst):

ans = {}

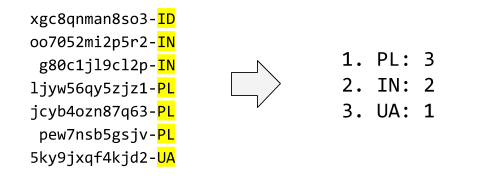
for i in lst:

site = i[-2:]

ans[site] = ans.get(site, 0) + 1

return ans

**10) Top counter**. We have a list with identifiers of form “id-SITE”. Show the top 3 sites. You can break ties in any way you want.

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/counter_2_top.png)

def top\_counter(lst):

site\_dict = counter(lst) # using last problem's solution

top\_keys = sorted(site\_dict, reverse=True, key=site\_dict.get)[:3]

return {key: site\_dict[key] for key in top\_keys}

**11) RLE**. Implement RLE (run-length encoding): encode each character by the number of times it appears consecutively.

* 'aaaabbbcca' ⇒ [('a', 4), ('b', 3), ('c', 2), ('a', 1)]
* (note that there are two groups of 'a')

def rle(s):

ans, cur, num = [], None, 0

for i in range(len(s)):

if i == 0:

cur, num = s[i], 1

elif cur != s[i]:

ans.append((cur, num))

cur, num = s[i], 1

else:

num += 1

if i == len(s) - 1:

ans.append((cur, num))

return ans

Using itertools.groupby

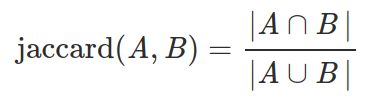
import itertools

def rle(s):

return [(l, len(list(g))) for l, g in itertools.groupby(s)]

**12) Jaccard**. Calculate the Jaccard similarity between two sets: the size of intersection divided by the size of union.

* jaccard({'a', 'b', 'c'}, {'a', 'd'}) = 1 / 4

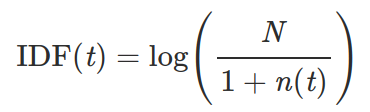
[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/formula_jaccard.png)

def jaccard(a, b):

return len(a & b) / len(a | b)

**13) IDF**. Given a collection of already tokenized texts, calculate the IDF (inverse document frequency) of each token.

* input example: [['interview', 'questions'], ['interview', 'answers']]

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/formula_idf_1.png)

Where:

* t is the token,
* n(t) is the number of documents that t occurs in,
* N is the total number of documents

from math import log10

def idf1(docs):

docs = [set(doc) for doc in docs]

n\_tokens = {}

for doc in docs:

for token in doc:

n\_tokens[token] = n\_tokens.get(token, 0) + 1

ans = {}

for token in n\_tokens:

ans[token] = log10(len(docs) / (1 + n\_tokens[token]))

return ans

import math

def idf2(docs):

n\_docs = len(docs)

docs = [set(doc) for doc in docs]

all\_tokens = set.union(\*docs)

idf\_coefficients = {}

for token in all\_tokens:

n\_docs\_w\_token = sum(token in doc for doc in docs)

idf\_c = math.log10(n\_docs / (1 + n\_docs\_w\_token))

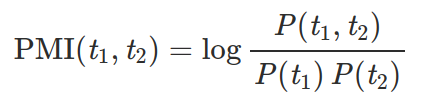
idf\_coefficients[token] = idf\_c

return idf\_coefficients

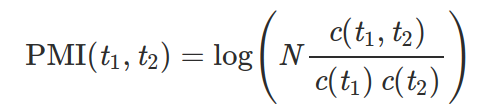
**14) PMI**. Given a collection of already tokenized texts, find the PMI (pointwise mutual information) of each pair of tokens. Return top 10 pairs according to PMI.

* input example: [['interview', 'questions'], ['interview', 'answers']]

PMI is used for finding collocations in text — things like “New York” or “Puerto Rico”. For two consecutive words, the PMI between them is:

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/formula_pmi_1.png)

The higher the PMI, the more likely these two tokens form a collection. We can estimate PMI by counting:

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/formula_pmi_2.png)

Where:

* N is the total number of tokens in the text,
* c(t1, t2) is the number of times t1 and t2 appear together,
* c(t1) and c(t2) — the number of times they appear separately.

Answer here

**Algorithmic Questions**

**1) Two sum**. Given an array and a number N, return True if there are numbers A, B in the array such that A + B = N. Otherwise, return False.

* [1, 2, 3, 4], 5 ⇒ True
* [3, 4, 6], 6 ⇒ False

Brute force, O(n2):

def two\_sum(numbers, target):

n = len(numbers)

for i in range(n):

for j in range(i + 1, n):

if numbers[i] + numbers[j] == target:

return True

return False

Linear, O(n):

def two\_sum(numbers, target):

index = {num: i for (i, num) in enumerate(numbers)}

n = len(numbers)

for i in range(n):

a = numbers[i]

b = target - a

if b in index:

j = index[b]

if i != j:

return True

return False

Using itertools.combinations

from itertools import combinations

def two\_sum(numbers, target):

for elem in combinations(numbers, 2):

if elem[0] + elem[1] == target:

return True

return False

**2) Fibonacci**. Return the n-th Fibonacci number, which is computed using this formula:

* F(0) = 0
* F(1) = 1
* F(n) = F(n-1) + F(n-2)
* The sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

def fibonacci1(n):

'''naive, complexity = O(2 \*\* n)'''

if n == 0 or n == 1:

return n

else:

return fibonacci1(n - 1) + fibonacci1(n - 2)

def fibonacci2(n):

'''dynamic programming, complexity = O(n)'''

base1, base2 = 0, 1

for i in range(n):

base1, base2 = base2, base1 + base2

return base1

def fibonacci3(n):

'''matrix multiplication, complexity = O(log(n))'''

def mx\_mul(m1, m2):

ans = [[0 for i in range(len(m2[0]))] for j in range(len(m1))]

for i in range(len(m1)):

for j in range(len(m2[0])):

for k in range(len(m2)):

ans[i][j] += m1[i][k] \* m2[k][j]

return ans

def pow(a, b):

ans = [[1, 0], [0, 1]]

while b > 0:

if b % 2 == 1:

ans = mx\_mul(ans, a)

a = mx\_mul(a, a)

b //= 2

return ans

ans = mx\_mul(pow([[1, 1], [1, 0]], n), [[1], [0]])[1][0]

return ans

Memoization with a dictionary

memo = {0: 0, 1: 1}

def fibonacci4(n):

'''Top down + memorization (dictionary), complexity = O(n)'''

if n not in memo:

memo[n] = fibonacci4(n-1) + fibonacci4(n-2)

return memo[n]

Memoization with lru\_cache

from functools import lru\_cache

@lru\_cache()

def fibonacci4(n):

if n == 0 or n == 1:

return n

return fibonacci4(n - 1) + fibonacci4(n - 2)

def fibonacci5(n):

'''Top down + memorization (list), complexity = O(n) '''

if n == 1:

return 1

dic = [-1 for i in range(n)]

dic[0], dic[1] = 1, 2

def helper(n, dic):

if dic[n] < 0:

dic[n] = helper(n-1, dic) + helper(n-2, dic)

return dic[n]

return helper(n-1, dic)

**3) Most frequent outcome**. We have two dice of different sizes (D1 and D2). We roll them and sum their face values. What are the most probable outcomes?

* 6, 6 ⇒ [7]
* 2, 4 ⇒ [3, 4, 5]

def most\_frequent\_outcome(d1, d2):

len\_ans = abs(d1 - d2) + 1

mi = min(d1, d2)

ans = [mi + i for i in range(1, len\_ans + 1)]

return ans

**4) Reverse a linked list**. Write a function for reversing a linked list.

* The definition of a list node: Node(value, next)
* Example: a -> b -> c ⇒ c -> b -> a

def reverse\_ll(head):

if head.next is not None:

last = None

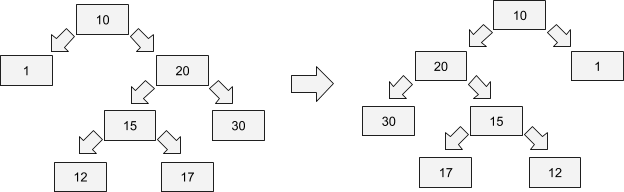
point = head

while point is not None:

point.next, point, last = last, point.next, point

**5) Flip a binary tree**. Write a function for rotating a binary tree.

* The definition of a tree node: Node(value, left, right)

[](https://github.com/alexeygrigorev/data-science-interviews/blob/master/img/flip_binary_tree.png)

def flip\_bt(head):

if head is not None:

head.left, head.right = head.right, head.left

flip\_bt(head.left)

flip\_bt(head.right)

**6) Binary search**. Return the index of a given number in a sorted array or -1 if it’s not there.

* [1, 4, 6, 10], 4 ⇒ 1
* [1, 4, 6, 10], 3 ⇒ -1

def binary\_search(lst, num):

left, right = -1, len(lst)

while right - left > 1:

mid = (left + right) // 2

if lst[mid] >= num:

right = mid

else:

left = mid

if right < 0 or right >= len(lst) or lst[right] != num:

return -1

else:

return right

**7) Deduplication**. Remove duplicates from a sorted array.

* [1, 1, 1, 2, 3, 4, 4, 4, 5, 6, 6] ⇒ [1, 2, 3, 4, 5, 6]

def deduplication1(lst):

'''manual'''

ans = []

last = None

for i in lst:

if last != i:

ans.append(i)

last = i

return ans

def deduplication2(lst):

# order is not guaranteed unless call sorted(list(set(lst))) to sort again

return list(set(lst))

**8) Intersection**. Return the intersection of two sorted arrays.

* [1, 2, 4, 6, 10], [2, 4, 5, 7, 10] ⇒ [2, 4, 10]

def intersection1(lst1, lst2):

'''reserves duplicates'''

ans = []

p1, p2 = 0, 0

while p1 < len(lst1) and p2 < len(lst2):

if lst1[p1] == lst2[p2]:

ans.append(lst1[p1])

p1, p2 = p1 + 1, p2 + 1

elif lst1[p1] < lst2[p2]:

p1 += 1

else:

p2 += 1

return ans

def intersection2(lst1, lst2):

'''removes duplicates'''

# order is not guaranteed unless call sorted(...) to sort again

return list(set(lst1) & set(lst2))

**9) Union**. Return the union of two sorted arrays.

* [1, 2, 4, 6, 10], [2, 4, 5, 7, 10] ⇒ [1, 2, 4, 5, 6, 7, 10]

def union1(lst1, lst2):

'''reserves duplicates'''

ans = []

p1, p2 = 0, 0

while p1 < len(lst1) or p2 < len(lst2):

if lst1[p1] == lst2[p2]:

ans.append(lst1[p1])

p1, p2 = p1 + 1, p2 + 1

elif lst1[p1] < lst2[p2]:

ans.append(lst1[p1])

p1 += 1

else:

ans.append(lst2[p2])

p2 += 1

return ans

def union2(lst1, lst2):

'''removes duplicates'''

# order is not guaranteed unless call sorted(...) to sort again

return list(set(lst1) | set(lst2))

**10) Addition**. Implement the addition algorithm from school. Suppose we represent numbers by a list of integers from 0 to 9:

* 12 is [1, 2]
* 1000 is [1, 0, 0, 0]

Implement the “+” operation for this representation

* [1, 1] + [1] ⇒ [1, 2]
* [9, 9] + [2] ⇒ [1, 0, 1]

def addition(lst1, lst2):

def list\_to\_int(lst):

ans, base = 0, 1

for i in lst[::-1]:

ans += i \* base

base \*= 10

return ans

val = list\_to\_int(lst1) + list\_to\_int(lst2)

ans = [int(i) for i in str(val)]

return ans

# another solution without int() and str() should be helpful

def addition2(lst1, lst2):

if len(lst2) == 0 or lst2 == [0]:

return lst1[:]

elif len(lst1) == 0:

return lst2[:]

# lst1, lst2 not empty

digit1, lst1rest = lst1[-1], lst1[:-1]

digit2, lst2rest = lst2[-1], lst2[:-1]

digit, remainder = divmod(digit1 + digit2, 10)

lst = addition2(

addition2(lst1rest, [digit]), # recursively add digit to lst1

lst2rest) # and then continue to add lst2

ans = lst + [remainder] # add the remainder as last digit

return ans

**11) Sort by custom alphabet**. You’re given a list of words and an alphabet (e.g. a permutation of Latin alphabet). You need to use this alphabet to order words in the list.

Example:

* Words: ['home', 'oval', 'cat', 'egg', 'network', 'green']
* Dictionary: 'bcdfghijklmnpqrstvwxzaeiouy'

Output:

* ['cat', 'green', 'home', 'network', 'egg', 'oval']

def sort\_by\_custom\_alphabet(dictionary, words):

words = sorted(words, key = lambda word: [dictionary.index(c) for c in word])

return words

**12) Check if a tree is a binary search tree**. In BST, the element in the root is:

* Greater than or equal to the numbers on the left
* Less than or equal to the number on the right
* The definition of a tree node: Node(value, left, right)

def check\_is\_bst(head, min\_val=None, max\_val=None):

"""Check whether binary tree is binary search tree

Aside of the obvious node.left.val <= node.val <= node.right.val have to be

fulfilled, we also have to make sure that there is NO SINGLE leaves in the

left part of node have more value than the current node.

"""

check\_val = True

check\_left = True

check\_right = True

if min\_val:

check\_val = check\_val and (head.val >= min\_val)

min\_new = min(min\_val, head.val)

else:

min\_new = head.val

if max\_val:

check\_val = check\_val and (head.val <= max\_val)

max\_new = max(max\_val, head.val)

else:

max\_new = head.val

if head.left:

check\_left = check\_is\_bst(head.left, min\_val, max\_new)

if head.right:

check\_right = check\_is\_bst(head.right, min\_new, max\_val)

return check\_val and check\_left and check\_right

**13) Maximum Sum Contiguous Subarray**. You are given an array A of length N, you have to find the largest possible sum of an Subarray, of array A.

* [-2, 1, -3, 4, -1, 2, 1, -5, 4] gives 6 as largest sum (from the subarray [4, -1, 2, -1]

from sys import maxsize

def max\_sum\_subarr(list1, size):

"""Use Kadane's Algorithm for a optimal solution

Time Complexity: O(n)

Desciption: Use one variable for current sum, and one for Overall sum at an index.

So here, the global\_max will keep on updating the max sum at any index-1,

and curr\_max will check the max value at an index.

And finally after iterating through the list, return the value of global\_max variable which contains Maximum sum.

"""

curr\_max=list1[0]

global\_max=list1[0]

for each in range(1, size):

curr\_max = max(list1[each], curr\_max+list1[each])

global\_max = max(global\_max, curr\_max)

return global\_max

n = int(input())

list1 = []

for i in range(0,n):

num = int(input())

list1.append(num)

print(max\_sum\_subarr(list1, len(list1)))

**14) Three sum**. Given an array, and a target value, find all possible combinations of three distinct numbers such that the sum of these three distinct numbers is equal to the target value.

Example:

Input: [12, 3, 1, 2, -6, 5, -8, 6], 0

Output: [[-8, 2, 6], [-8, 3, 5], [-6, 1, 5]]

def threeSum(array, target):

array.sort()

triplets = []

for i in range(len(array) - 2):

left = i + 1

right = len(array) - 1

while left < right:

currentSum = array[i] + array[left] + array[right]

if currentSum == target:

triplets.append([array[i], array[left], array[right]])

left += 1

right -= 1

elif currentSum > target:

right -= 1

elif currentSum < target:

left += 1

return triplets

**15) Find Duplicate in array** Given and array, find all duplicated value in the array

Example:

input: [1,2,3,4,3,4,5] output: [3,4]

a = [1,2,3,4,3,4,5]

def duplicates(a):

seen = set()

duplicated = set()

for i in a:

if i in seen:

duplicated.add(i)

else:

seen.add(i)

return duplicated