1) Given an integer **n** and an array **a** of length **n**, your task is to apply the following mutation to **a**:

* Array **a** mutates into a new array **b** of length **n**.
* For each **i** from **0** to **n - 1**, **b[i] = a[i - 1] + a[i] + a[i + 1]**.
* If some element in the sum **a[i - 1] + a[i] + a[i + 1]** does not exist, it should be set to **0**. For example, **b[0]** should be equal to **0 + a[0] + a[1]**.

Example

For **n = 5** and **a = [4, 0, 1, -2, 3]**, the output should be **solution(n, a) = [4, 5, -1, 2, 1]**.

* **b[0] = 0 + a[0] + a[1] = 0 + 4 + 0 = 4**
* **b[1] = a[0] + a[1] + a[2] = 4 + 0 + 1 = 5**
* **b[2] = a[1] + a[2] + a[3] = 0 + 1 + (-2) = -1**
* **b[3] = a[2] + a[3] + a[4] = 1 + (-2) + 3 = 2**
* **b[4] = a[3] + a[4] + 0 = (-2) + 3 + 0 = 1**

So, the resulting array after the mutation will be **[4, 5, -1, 2, 1]**.

function solution(n, a) {

    let b = [];

    for (let i = 0; i <= n-1; i++){

        if( i == 0){

            b[0] = a[0] + a[1];

        } else if ( i == n-1) {

            b[i] = a[i-1] + a[i];

        }else{

            b[i] = a[i-1] + a[i] + a[i + 1];

        }

    }

    return b;

}

let n = 5;

let a = [4, 0, 1, -2, 3]

console.log( solution (n, a));

2) Given an array of integers **a**, your task is to calculate the digits that occur the most number of times in the array. Return the array of these digits in ascending order.

Example

For **a = [25, 2, 3, 57, 38, 41]**, the output should be **solution(a) = [2, 3, 5]**.

Here are the number of times each digit appears in the array:

**0 -> 0**

**1 -> 1**

**2 -> 2**

**3 -> 2**

**4 -> 1**

**5 -> 2**

**6 -> 0**

**7 -> 1**

**8 -> 1**

The most number of times any number occurs in the array is **2**, and the digits which appear **2** times are **2**, **3** and **5**. So the answer is **[2, 3, 5]**.

3) You are given an array of arrays **a**. Your task is to group the arrays **a[i]** by their [mean](keyword://arithmetic-mean) values, so that arrays with equal mean values are in the same group, and arrays with different mean values are in different groups.

Each group should contain a set of indices (**i**, **j**, etc), such that the corresponding arrays (**a[i]**, **a[j]**, etc) all have the same mean. Return the set of groups as an array of arrays, where the indices within each group are sorted in ascending order, and the groups are sorted in ascending order of their minimum element.

Example

* For
* **a = [[3, 3, 4, 2],**
* **[4, 4],**
* **[4, 0, 3, 3],**
* **[2, 3],**
* **[3, 3, 3]]**

the output should be

**solution(a) = [[0, 4],**

**[1],**

**[2, 3]]**

* + **mean(a[0]) = (3 + 3 + 4 + 2) / 4 = 3**;
  + **mean(a[1]) = (4 + 4) / 2 = 4**;
  + **mean(a[2]) = (4 + 0 + 3 + 3) / 4 = 2.5**;
  + **mean(a[3]) = (2 + 3) / 2 = 2.5**;
  + **mean(a[4]) = (3 + 3 + 3) / 3 = 3**.

There are three groups of means: those with mean **2.5**, **3**, and **4**. And they form the following groups:

* + Arrays with indices **0** and **4** form a group with mean **3**;
  + Array with index **1** forms a group with mean **4**;
  + Arrays with indices **2** and **3** form a group with mean **2.5**.

Note that neither

**solution(a) = [[0, 4],**

**[2, 3],**

**[1]]**

nor

**solution(a) = [[0, 4],**

**[1],**

**[3, 2]]**

will be considered as a correct answer:

* + In the first case, the minimal element in the array at index **2** is **1**, and it is less then the minimal element in the array at index **1**, which is **2**.
  + In the second case, the array at index **2** is not sorted in ascending order.
* For
* **a = [[-5, 2, 3],**
* **[0, 0],**
* **[0],**
* **[-100, 100]]**

the output should be

**solution(a) = [[0, 1, 2, 3]]**

The mean values of all of the arrays are **0**, so all of them are in the same group.

4) You've created a new programming language, and now you've decided to add hashmap support to it. Actually you are quite disappointed that in common programming languages it's impossible to add a number to all hashmap keys, or all its values. So you've decided to take matters into your own hands and implement your own hashmap in your new language that has the following operations:

* **insert x y** - insert an object with key **x** and value **y**.
* **get x** - return the value of an object with key **x**.
* **addToKey x** - add **x** to all keys in map.
* **addToValue y** - add **y** to all values in map.

To test out your new hashmap, you have a list of queries in the form of two arrays: **queryTypes** contains the names of the methods to be called (eg: **insert**, **get**, etc), and **queries** contains the arguments for those methods (the **x** and **y** values).

Your task is to implement this hashmap, apply the given queries, and to find the **sum** of all the results for **get** operations.

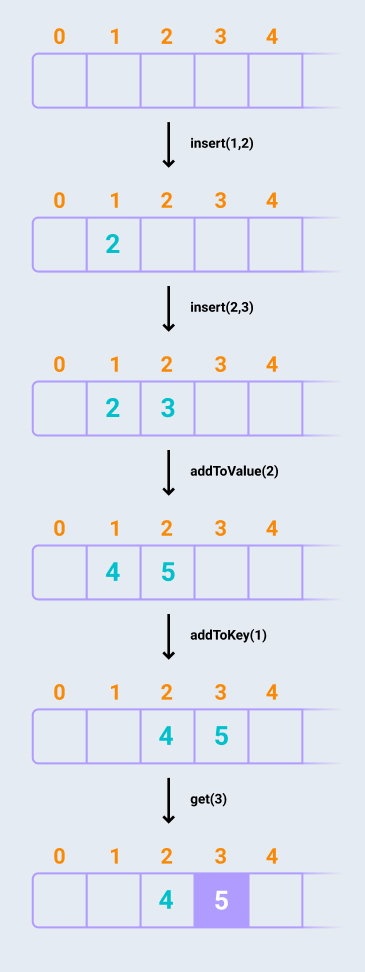
Example

* For **queryType = ["insert", "insert", "addToValue", "addToKey", "get"]** and **query = [[1, 2], [2, 3], [2], [1], [3]]**, the output should be **solution(queryType, query) = 5**.

The hashmap looks like this after each query:

* + 1 query: {1: 2}
  + 2 query: {1: 2, 2: 3}
  + 3 query: {1: 4, 2: 5}
  + 4 query: {2: 4, 3: 5}
  + 5 query: answer is **5**

The result of the last **get** query for **3** is **5** in the resulting hashmap.



* For **queryType = ["insert", "addToValue", "get", "insert", "addToKey", "addToValue", "get"]** and **query = [[1, 2], [2], [1], [2, 3], [1], [-1], [3]]**, the output should be **solution(queryType, query) = 6**.

The hashmap looks like this after each query:

* + 1 query: {1: 2}
  + 2 query: {1: 4}
  + 3 query: answer is **4**
  + 4 query: {1: 4, 2: 3}
  + 5 query: {2: 4, 3: 3}
  + 6 query: {2: 3, 3: 2}
  + 7 query: answer is **2**

The sum of the results for all the **get** queries is equal to **4 + 2 = 6**.