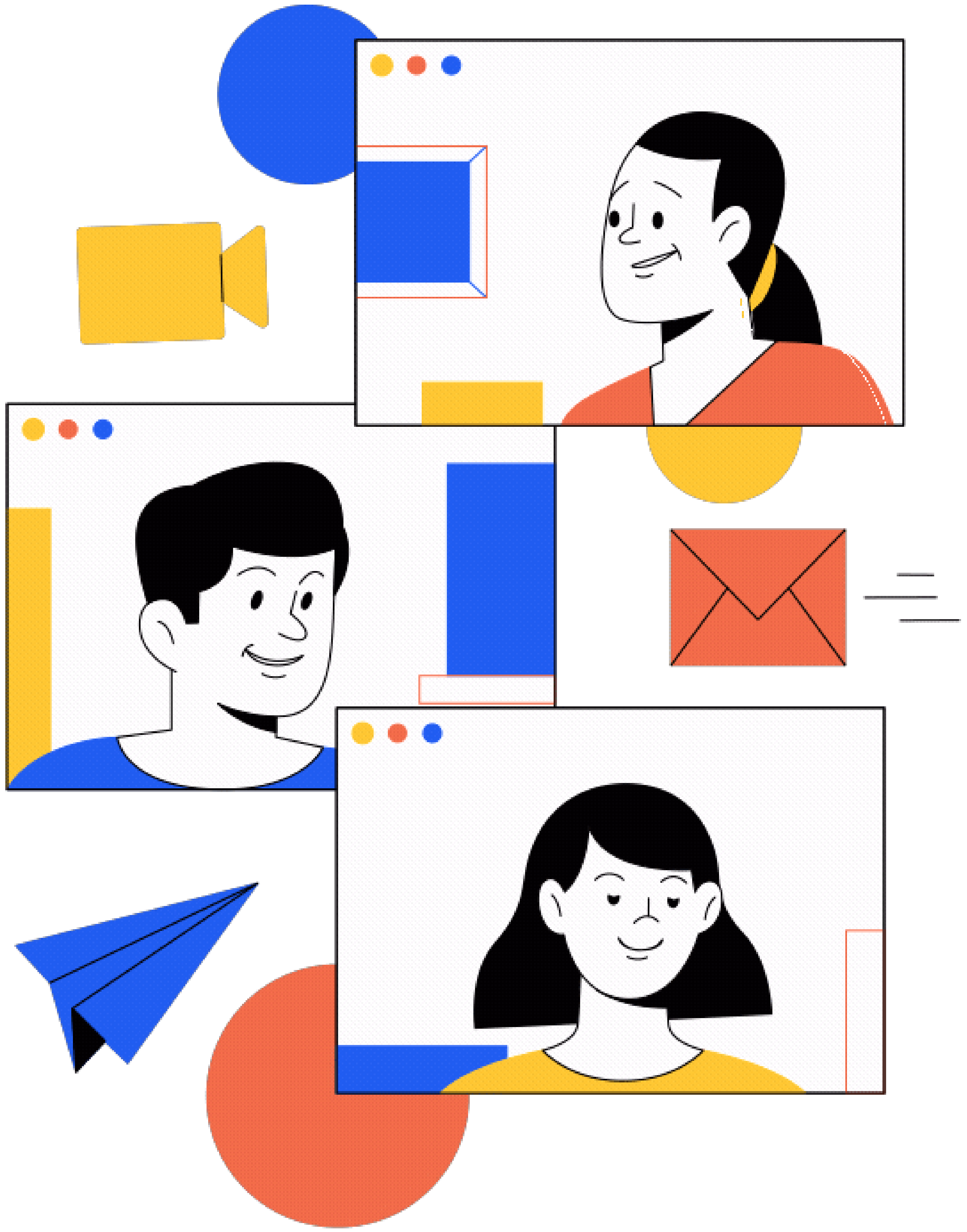
CLOSED HASHING

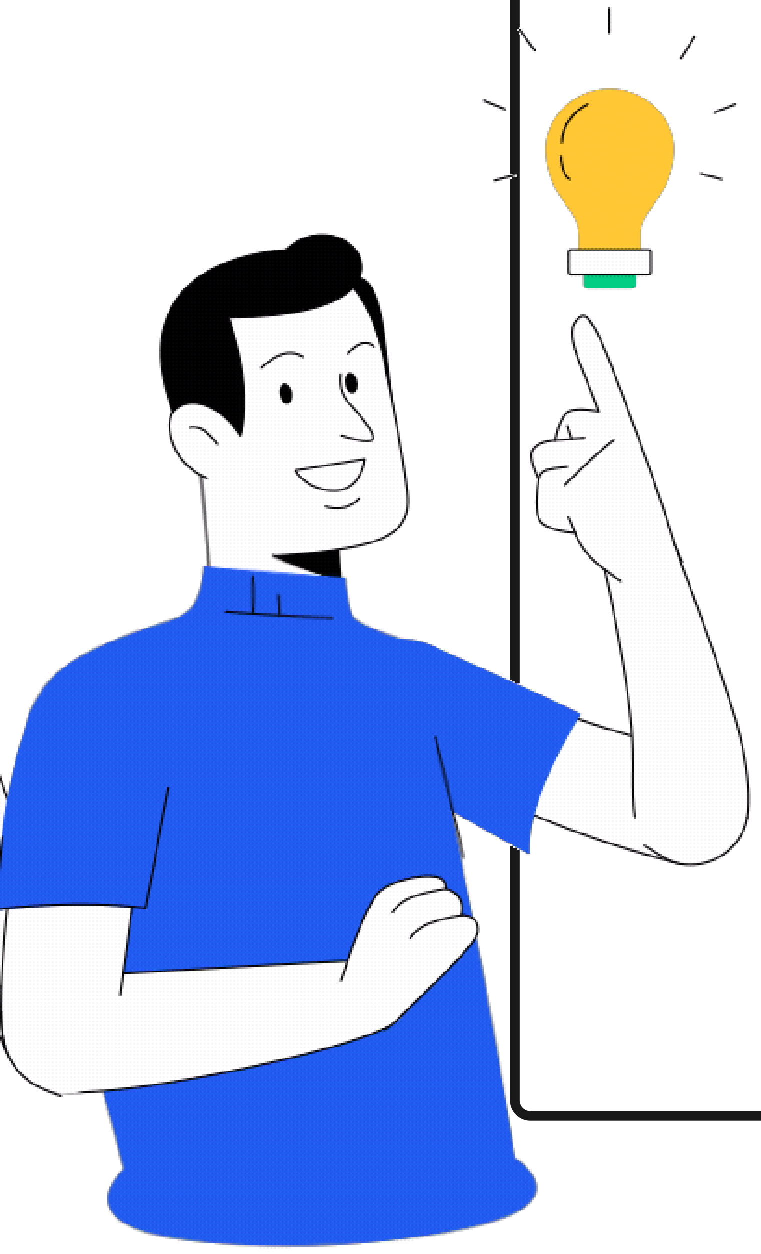
Name:Rokaia Emad ID:220617



HOW TO IMPROVE EXECUTION TIME?

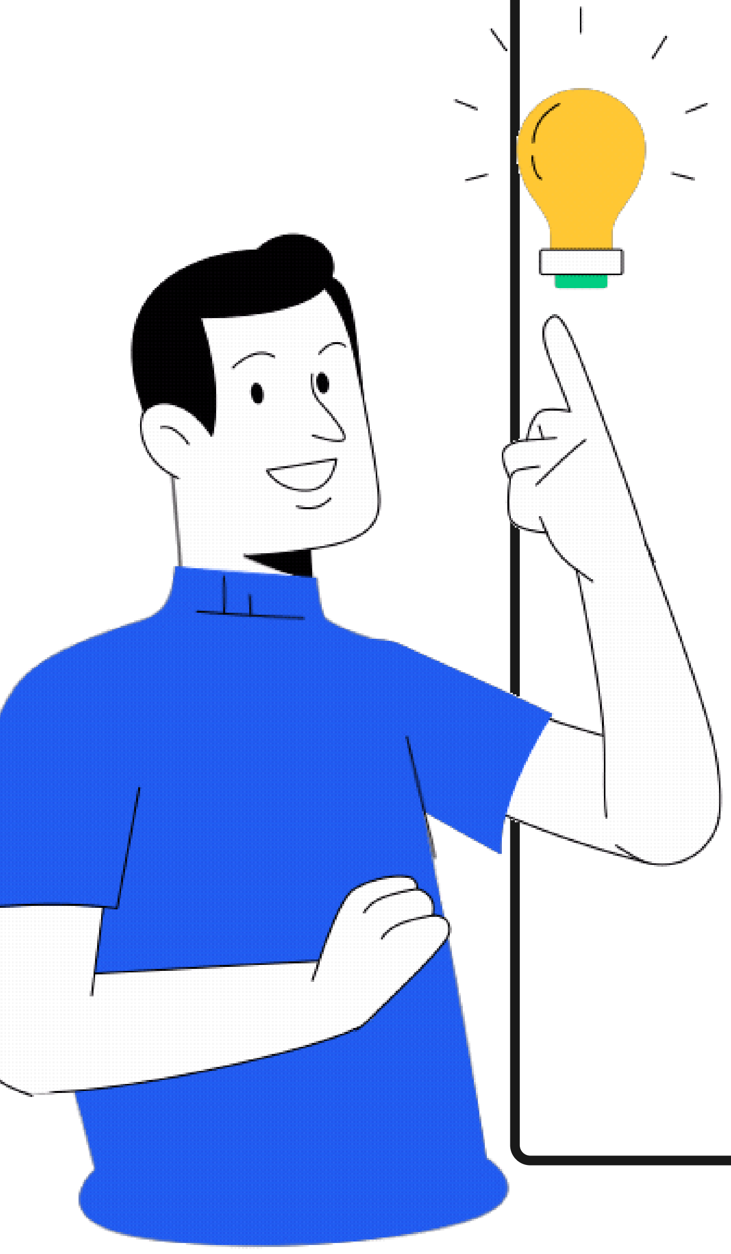
IF WE WANT TO SEARCH ABOUT LEO IN THIS ARRAY USING BBRUTE FORCE

JAN	TIM	MIA	LEO	SAM



HOW TO IMPROVE EXECUTION TIME?

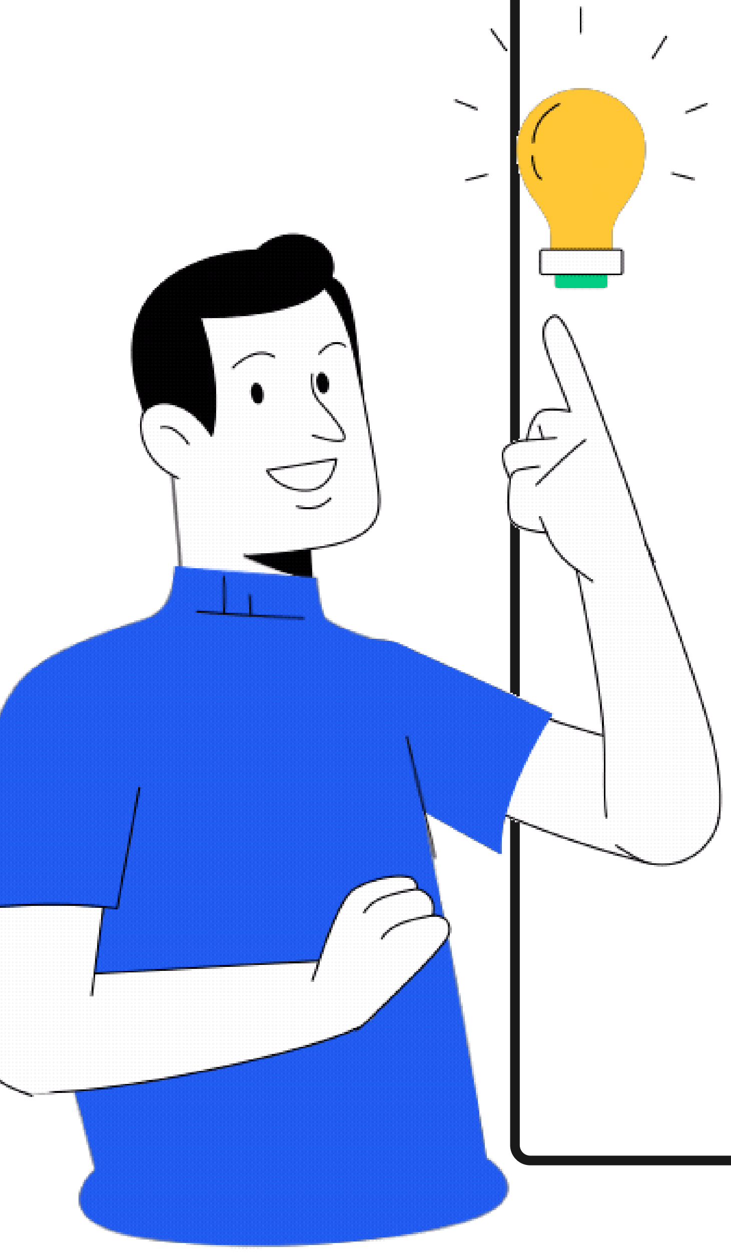
IF WE WANT TO SEARCH ABOUT LEO IN THIS ARRAY USING BBRUTE FORCE



JAN	TIM	MIA	LEO	SAM

HOW TO IMPROVE EXECUTION TIME?

IF WE WANT TO SEARCH ABOUT LEO IN THIS ARRAY USING BBRUTE FORCE

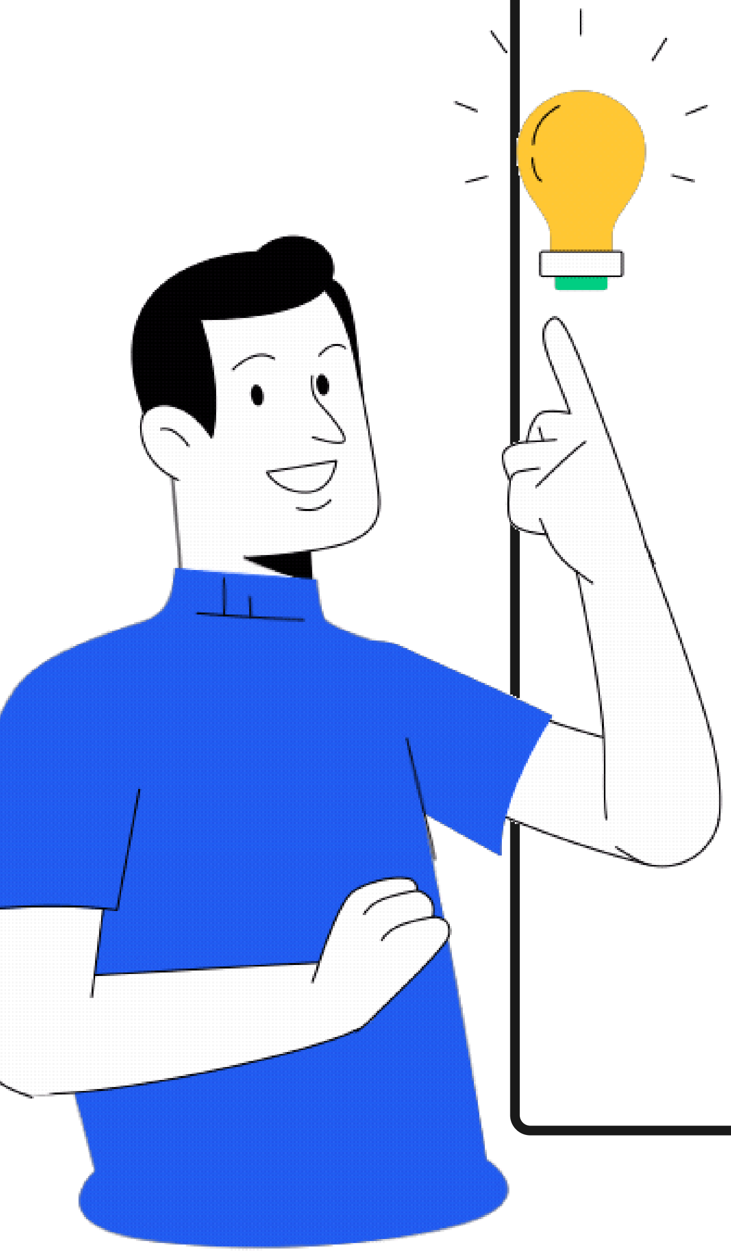


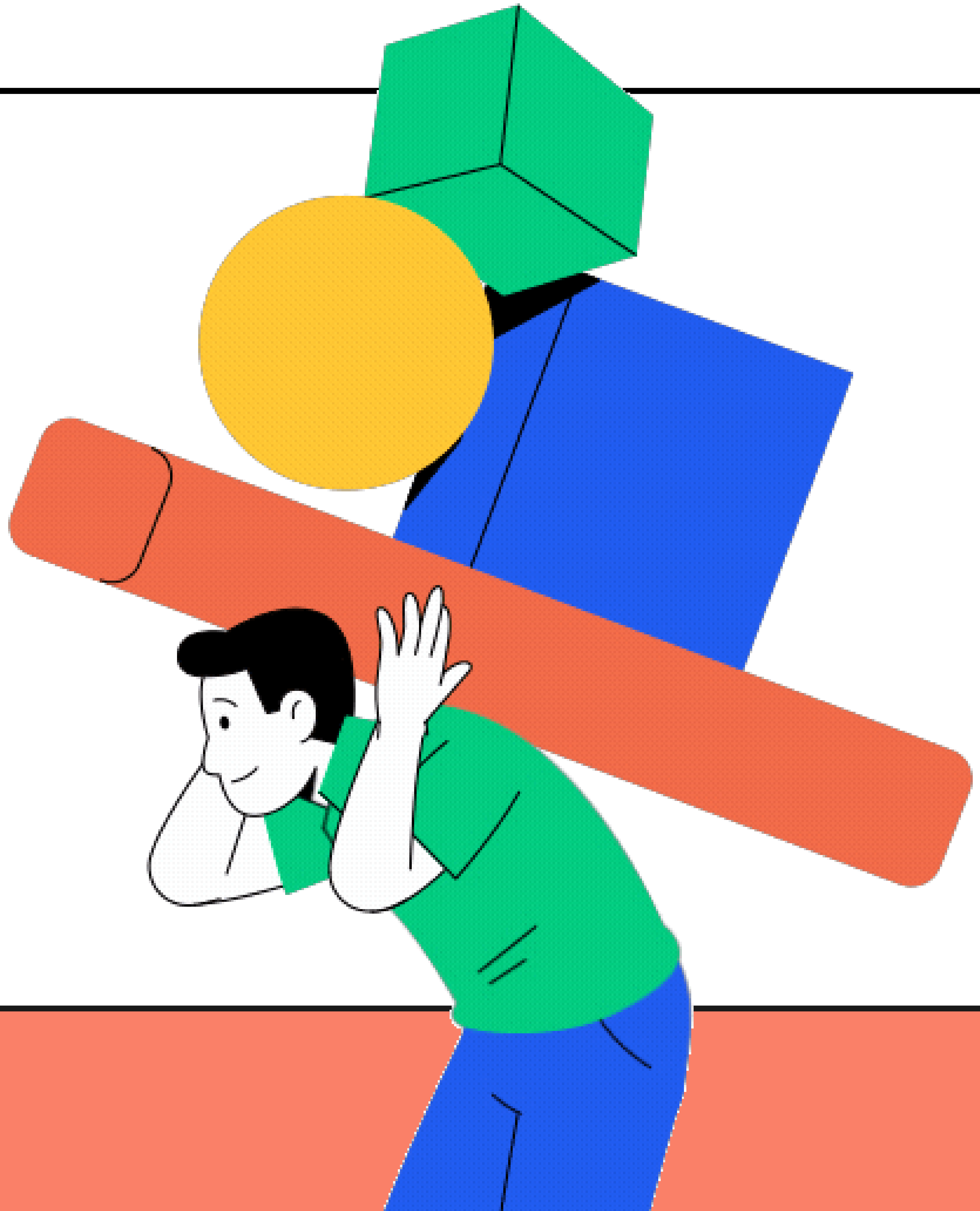
JAN	TIM	MIA	LEO	SAM

HOW TO IMPROVE EXECUTION TIME ?

IF WE WANT TO SEARCH ABOUT LEO IN THIS ARRAY USING BBRUTE FORCE

JAN	TIM	MIA	LEO	SAM





A WAY TO IMPROVE

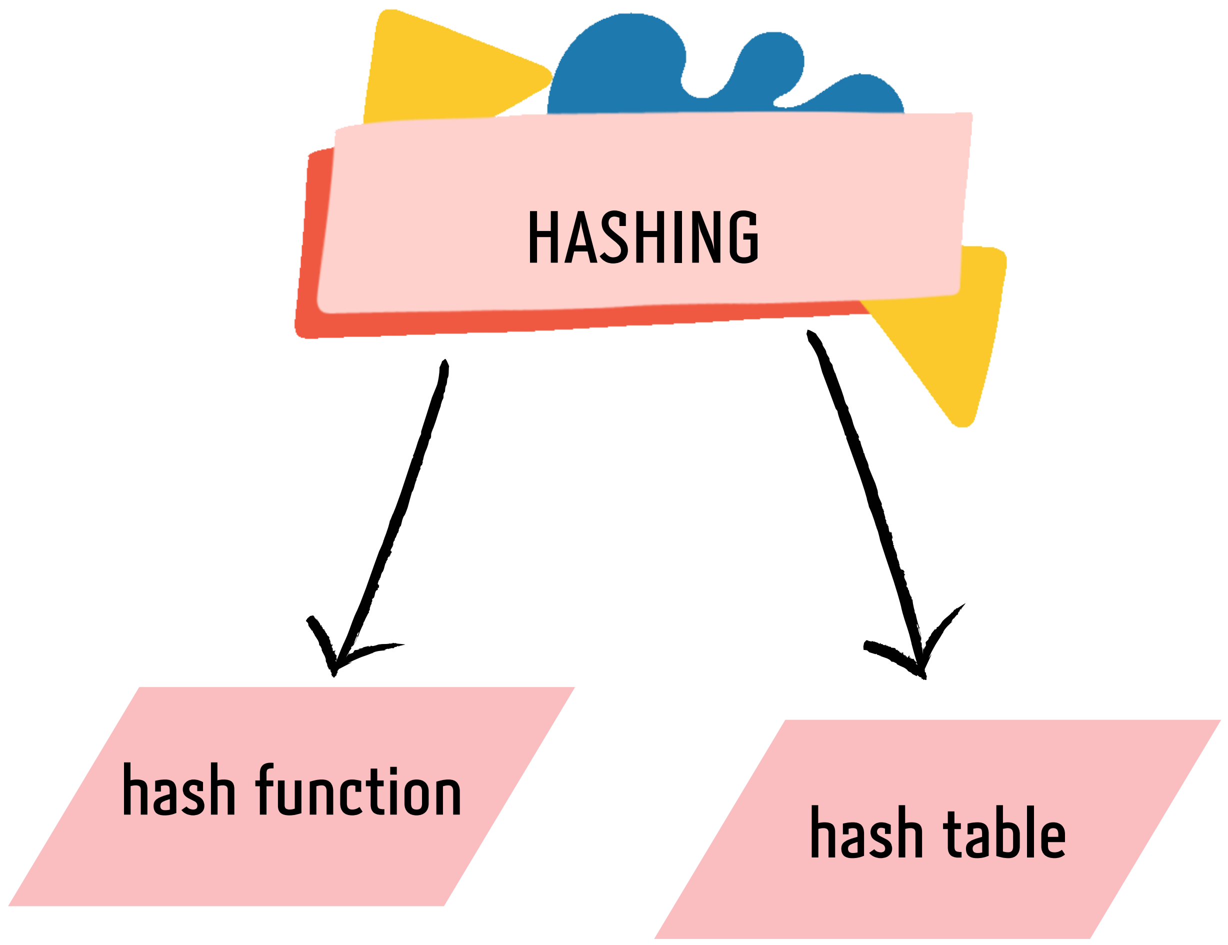
HOW TO IMPROVE EXECUTION TIME?

IF WE KNOW THE INDEX OF LEO CAN THAT REDUCE TIME TO SEARCH ABOUT IT?

LEO=? LEO=3

JAN	TIM	MIA	LEO	SAM

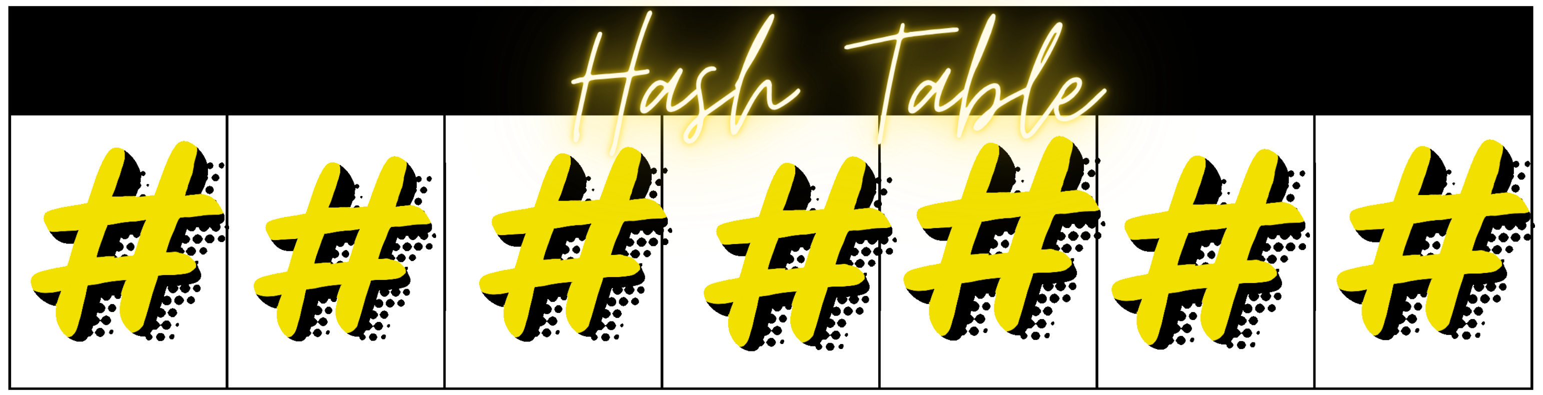
HERE COME THE HASHING IDEA

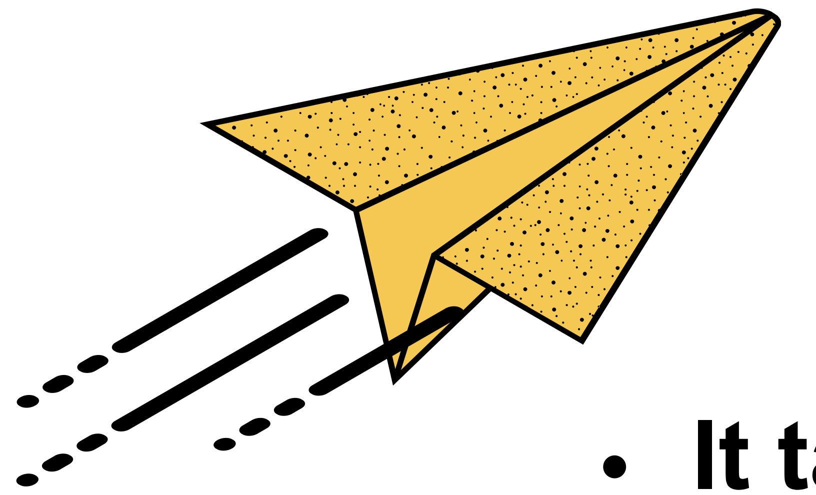

HASHING
hash function
hash table

HASH TABLE

- based on space and time trade off strategy as we take extra space to reduce time

  - Moreover, Prestructuring is a variety of space for-time tradeoff which takes the input and design a data structure(Ex:Hash Table) that has a role in speeding up the execution time





HASH FUNCTION

- It takes object and return index from[0...N-1]

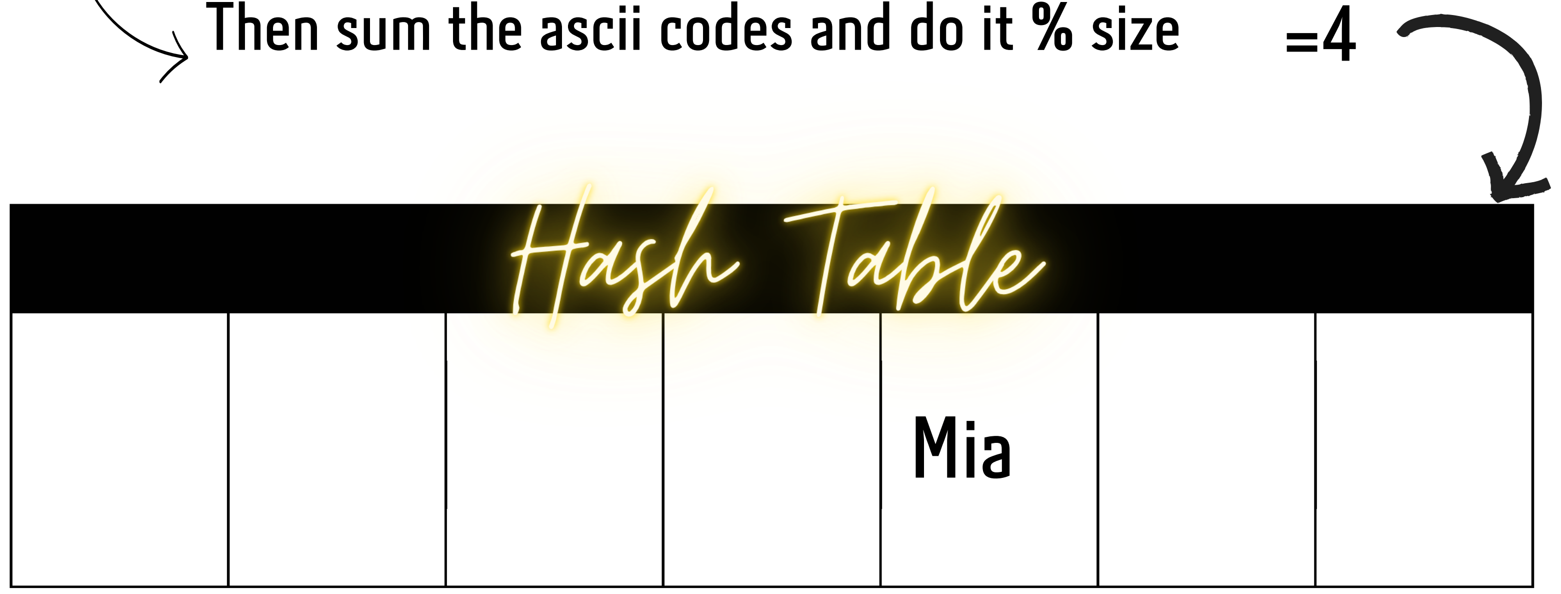
Example:

Mia

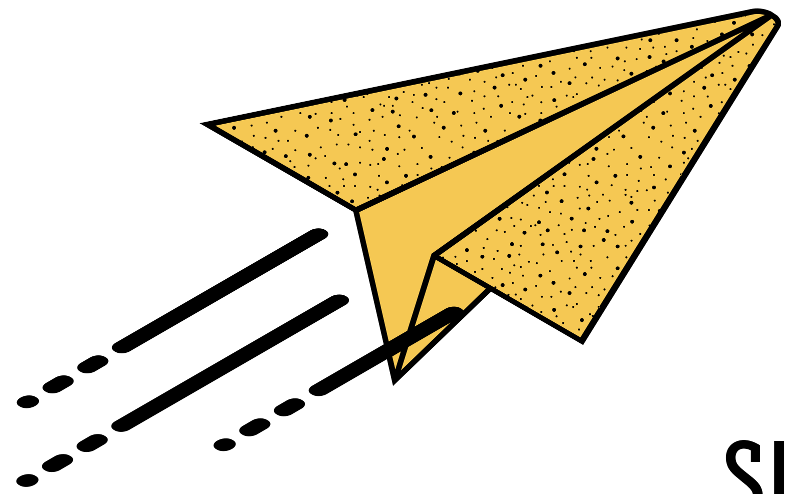
Bring ascii code of each letter

M 77 i 105 a 97

Then sum the ascii codes and do it % size =4



Example:



SUE Bring ascii code of each letter

S 83 U 117 E 101

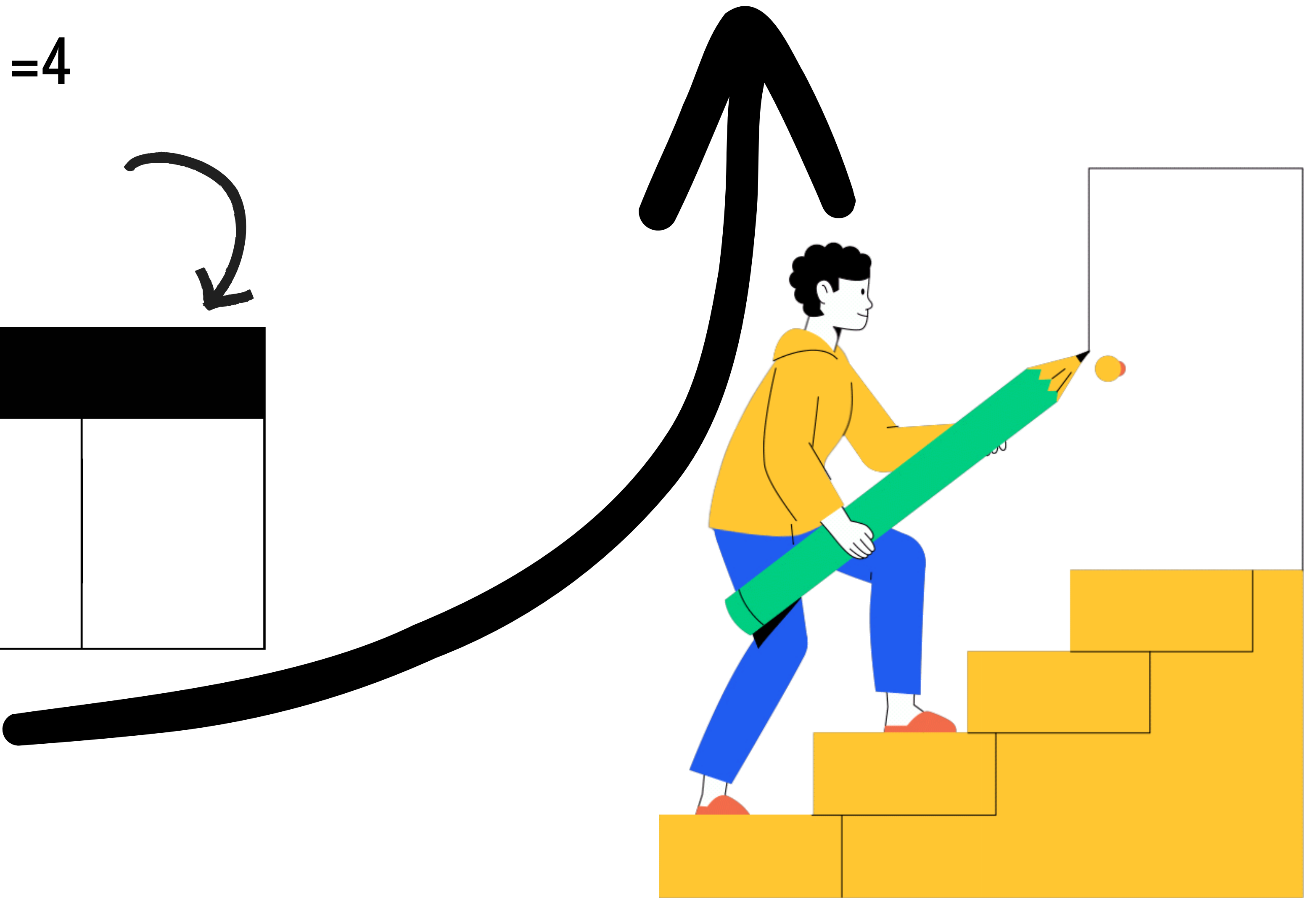
Then sum the ascii codes and do it % size =4

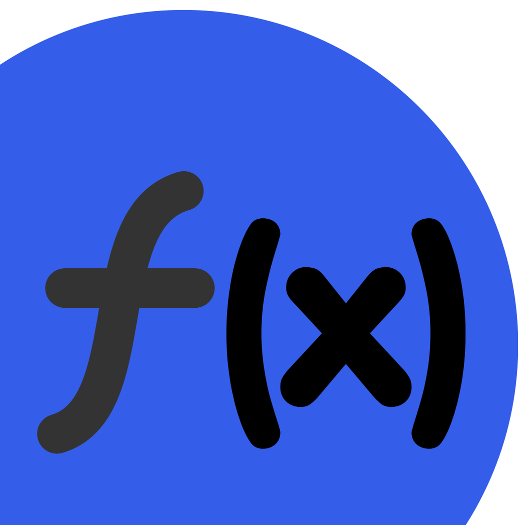
COLLISION :Hash function map diffrent element in the same index

Hash Table

				Mia		

SUE ?



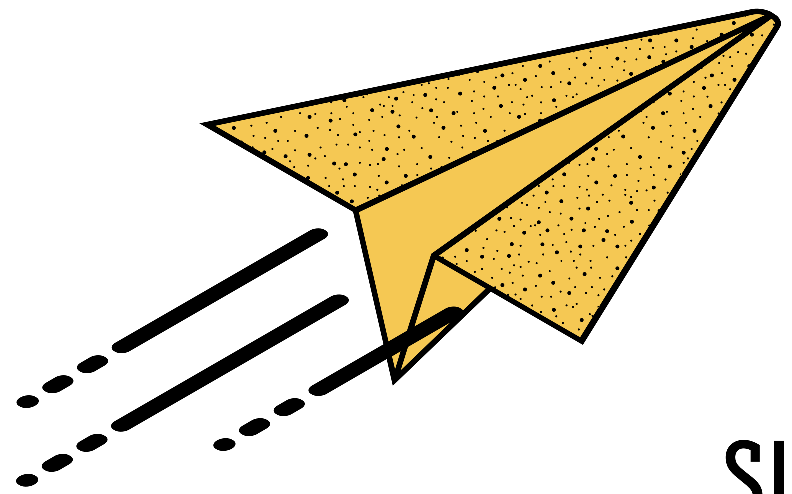


CLOSED HASHING

- USE 1D ARRAY ELEMENTS (KEY,VALUE)

- When adding an entry, check if the hash index is empty. If it is, add the element to this index

- In case of a collision, employ a systematic procedure (such as linear probing) to store elements in the nearest empty cell within the table



Example:

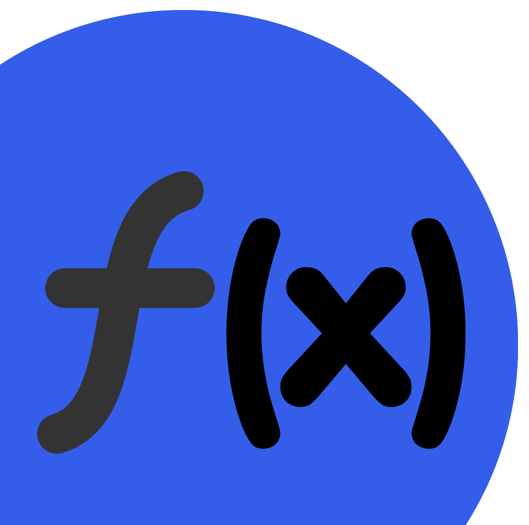
SUE Bring ascii code of each letter

S 83 U 117 E 101

SUE ADDED IN THE NEAREST EMPTY SLOT AS COLLISION OCCUR

Hash Table

				Mia	SUE	



CODE ANALYSIS

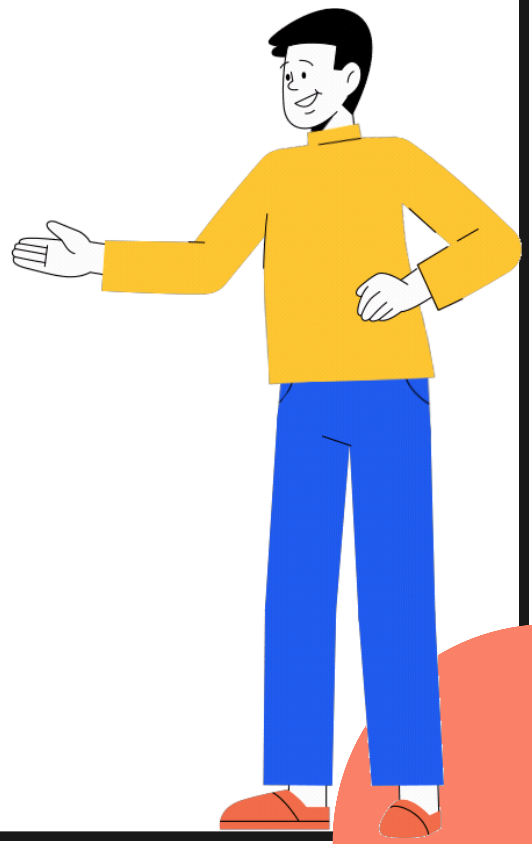
SINCE THE HASH FUNCTION ITERATES AROUND THE LETTERS OF THE KEY TO GET THE SUMMATION OF ASCII CODE %SIZE THE ANALYSIS WILL BE

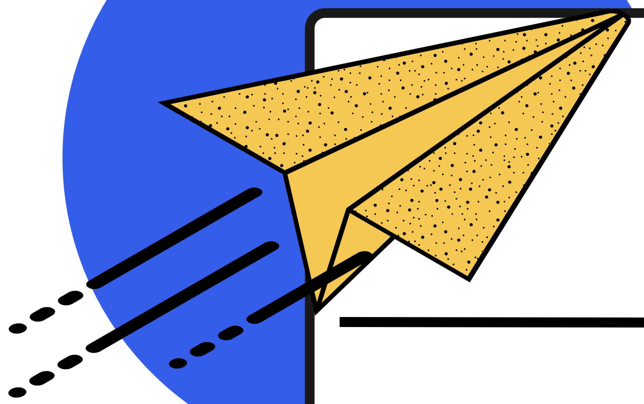
```python
def hashfunction(key,size):
        return sum(ord(c) for c in key) % size
```

$$\sum_{i=0}^{n-1} 1 \qquad u-l+1$$

$$n-1-0+1$$

$$\in o(n)$$





CODE ANALYSIS

CONSTRUCTION OF HASH TABLE

$$\sum_{i=0}^{n-1} 1 \quad \overset{u-l+1}{n-1-0+1}$$

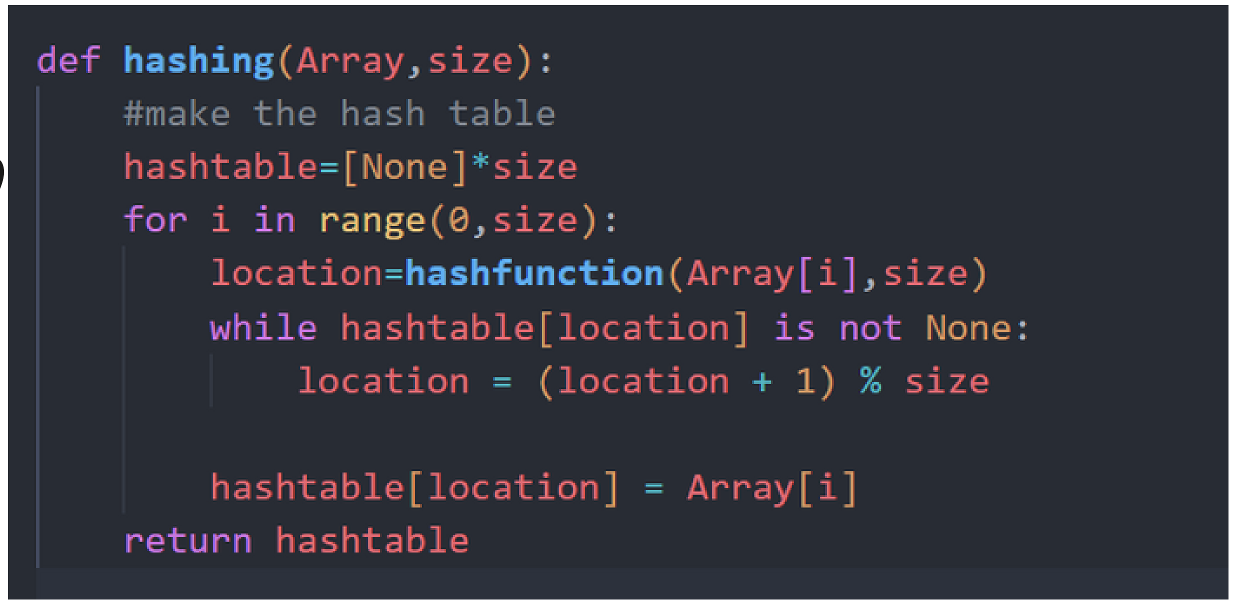
$$\in o[n]$$

TO ADD IN HASH TABLE

IN THE WORST CASE

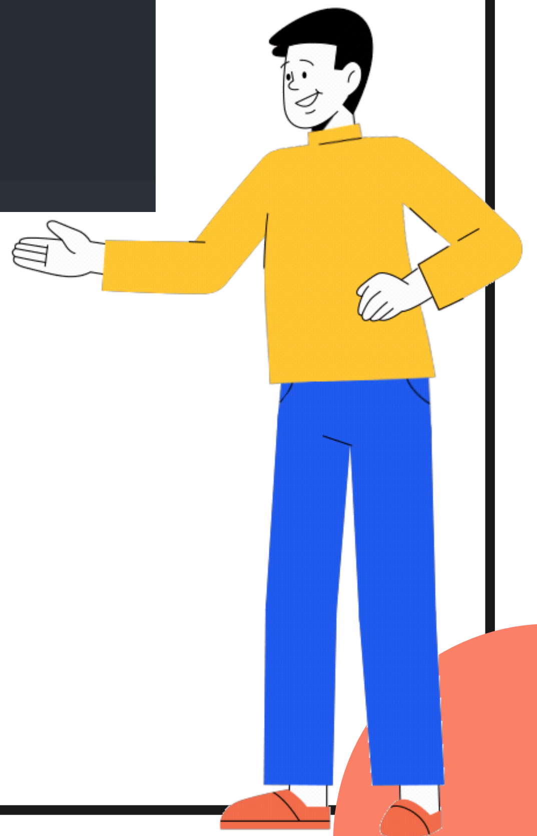
$$\sum_{i=0}^{n-1} 1 \quad \overset{u-l+1}{n-1-0+1}$$

$$\in o[n]$$



```python
def hashing(Array,size):
    #make the hash table
    hashtable=[None]*size
    for i in range(0,size):
        location=hashfunction(Array[i],size)
        while hashtable[location] is not None:
            location = (location + 1) % size

        hashtable[location] = Array[i]
    return hashtable
```



CODE ANALYSIS

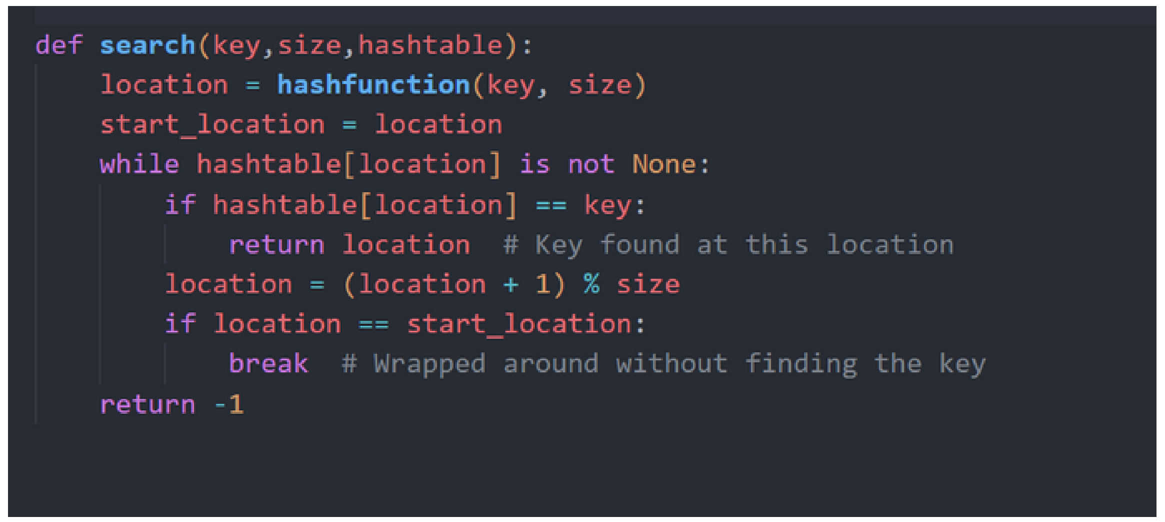
IIn the worst case if collision occur ,
there is a linear probing

$$\sum_{i=0}^{n-1} 1 \quad = n-1-0+1$$

$$\in o(n)$$

but in best case if there is no collision it will be

$$\in o(1)$$



```python
def search(key,size,hashtable):
    location = hashfunction(key, size)
    start_location = location
    while hashtable[location] is not None:
        if hashtable[location] == key:
            return location  # Key found at this location
        location = (location + 1) % size
        if location == start_location:
            break  # Wrapped around without finding the key
    return -1
```