Efficiency → How well you are using your resources to get the job done.

Time

↳ How long our code takes to run

Space

↳ How much storage space we need.

Notation Intro :

We express efficiency with O(n) → Big oh Notation

↳ For different algorithms we will replace n with algebric expressions.

e.g O(log(n))

O($n^3$)

O(1)

O(n) → n represents length on an input to your function.

e.g we have a coded message with cipher key.

A → Y , B → E , C → S

we will write a Program:

```
function decode(input):
    create output string
    for each letter in input:
        → get new_letter for letter's locatic
          in cipher
        → add new_letter to output
    return output.
```

↳ Now for Time efficiency just count no of lines.

1 → Creating output string
2 → return output

} Both will happen once when function is executed

→ so let's add 2 in our efficiency

↳ $O( \ 2)$

→ Now we have 2 lines in loop. (if we have n cletters in input). They will be executed 2n times.

↳ $O(2n+2)$

e.g if $n=10$

→ $2(10)+2 = 22$

↳ For actual efficieny you can multiply it with amount of time your computer takes to run one line of code.

↳ This was basic understanding.

Now let's address some complications:-

↳ we had a line for 'for loop' does it counts too in efficiency?

↳ Yes, with the help of loop we are getting each character one by one. As it is also happening one time for every letter.

↳ $O(3n+2)$

↳ Now if we look carefully characters are stored in cipher (if it is a list) → means [A→Y, B→E, C→S]

↳ Then we need to look each letter against our current letter.

↳ so as we have 26 letters we have to check for every letter that what is written in front of it.

↳ $O((3+26)n+2)$

↳ $O(29n+2)$ →

But we can write it as $O(n)$ because as n grows constant become less significant.

now $n=10 \Rightarrow 29(10)+2$

$\Rightarrow$ 292 computations

↳ Now you can see that it went up when we choose a specific data structure.

Worst Case and Approximation :

↳ As above we said we have to check all 26 letters but in reality we can get away without checking all. But we always talk about worst case scenario where we check all letters each time.

↳ We can also talk about efficiency with average case and Best Case.

A, B, C . . . . . X, Y, Z

1 2 3 ·· 13 ·· 24 25 26

↑ Best Case

↑ Average Case

↳ Worst Case

↳ The best case is if we find what we looking for at very first.

↳ The average case will~~t~~ be if we go till the middle i.e get all of our values.

For average case efficiency :

↳ $O((13+3)n + 2)$

↳ $O(16n+2)$

Space Efficiency :

Let's say program ask to copy input string 3 times → mean it will take 3x space so space efficiency will look like :

↳ $O(3n)$

Examples :

input → list_info

↳ It's a list of infos

↳ where each info has 'name' & 'age'

$n$ = no of elements in list_info

$m$ = no of info in each element of list_info

①

```
def example1 (list_info):
    for info in list_info:
        print info['name']
```

↳ print statement is within for loop. so if we have n names it will be executed n times.
or info

↳ ~~so~~ ↳ $\boxed{O(n)}$ ~~for~~ → for is looking for every info so '1' for 'for' loop is multiplied.

②

```
def example2 (list_info):
    print list_info[0]['name']
    print list_info[0]['age']
```

↳ It will directly go to first element & will print its name. & age.

↳ $\boxed{O(1)}$

③

```
def example3(list_info):
    for info in list_info:
        for element in info:
            → print element, ":",
                    info[element]
```

↳ Print will be executed according to how many elements (name, age) are present. } → m

↳ for loop 1 is multiplied, as for loop is looking

for every element once every time.
so for m elements → 1 x m
↳ m

↳ Now Two statements in the outer for loop will be n times.
↳ n → no of info in list_info
↳ O(n) → we can have O(2n) but we ignore 2.
↳

↳ $\boxed{O(n * m) \text{ or } O(nm)}$ → (n * m) we ignore them because they do not have significant effect.

(4) def example4(list_info):

```
(1) →temp = 'NO info'
    for info1 in list_info:
        for info 2 in list_info:
            if info1['age'] < info2['age']:
                temp = info 2['age']
            else:
                temp = info1['age']
(1) → print temp
```

↳ so first statement 'temp = ..' & last statement 'print ..' will be executed once each
so
↳ O( + 2)

↳ Now we have everything left over in loop.

↳ As we have if-else scenario means one of the statement will be executed. & this one statement will be executed according to how many infos it have

↳ list_info has.

↳ Which is n.

↳ $O(n+2)$

↳ Now another for loop will be executed n times.

↳ $O(n^2+2)$

we can ignore this 2.

↳ $\boxed{O(n^2).}$