**Sort with user-defined function**

The third parameter(**comp**) for the std::sort() function can also be a **user-defined** function that defines the order or sorting.

One should note that this function must return a **boolean** value(True/False).

# std::sort(demo , demo + len, My\_func)

bool My\_func( int a, int b)

{

return (a%10)<(b%10);

}

as a result : {18, 45, 34, 92, 21} => {21,92,34,45,18}

**When dealing with negative indices**

How to deal with negative indices when using arrays ?

Well there s a technique that we use a **shift** :

1:28:46

**Power of 10**

**If you want to write X = 10^5**

**int X = 1e5 // 1 \* 10^5**

**Char to Int (but not in ASCII :D )**

**We make :**

**Cout << ( n[2] – ‘0’ ) ;**

**Means that we retrieve from it the value 48:**

**‘5’ – ‘0’ = 5**

**Think MATHEMATICALLY**

**Instead of BForce**

In the example of [the matrix](https://codeforces.com/group/5pUldkahAU/contest/510058/problem/T) you have:

0 0 0 0 1

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

**[0,4]** is the index of

You want **1** to achieve **[2,2]** , so why looking for ~~Swapping elements~~?

Instead of calculating the **Manhattan distance ?**

I mean : **[ 0 , 4 ] – [ 2 , 2 ] = [ 2 , 2 ]**

It just needs a lil bit of insight :

Moves = **abs(** 2 – *row\_pos\_of\_1* **) + abs(** *2 – col\_pos\_of\_1* **)**

*ALWAYS FIND A FORMULA !*

**Working in parallel**

If *i objects* working in parallel, the **result** is always written like:

R = 1/a + 1/b + …. + 1/i

The **result** is **1/R**

Example:

* How much, N taps filling a tank **at the same time** ?
* How much time, for painters that paint **at same time**

**MIN , MAX and The index of a value**

To return the **min** of an array:

min\_of\_arr = **\*min\_element(** arr , arr+n )

To return the **max** of an array:

min\_of\_arr = **\*max\_element(**arr , arr+n)

to return **the index** of a value in the array : **distance( iter0 , iter1)**

index\_of\_min = **distance(** arr , *min\_element(* arr , arr +n *)*  **)**

**IMPORTANT:**

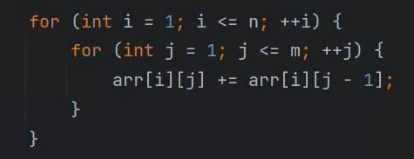
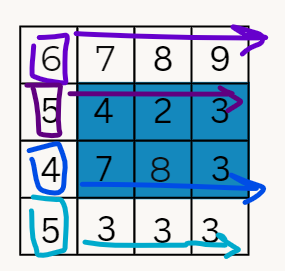
* *distance()* on accepts **ITERATORS** not values !
* *min\_element()* **DOES NOT** return values , so we should **ADD \***

Intro to CP using C++

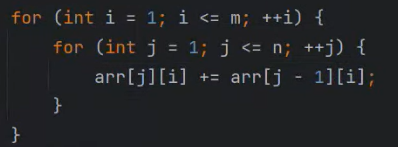
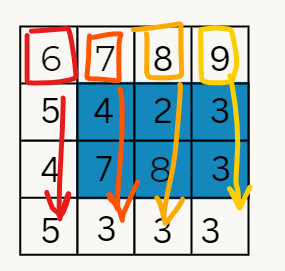
**Column sum vs Row sum**

n : number of rows , m : number of columns

row sum : LOOP **start with row**

Column sum : LOOP **starts with columns**

the result is :

**Partial Sum:**

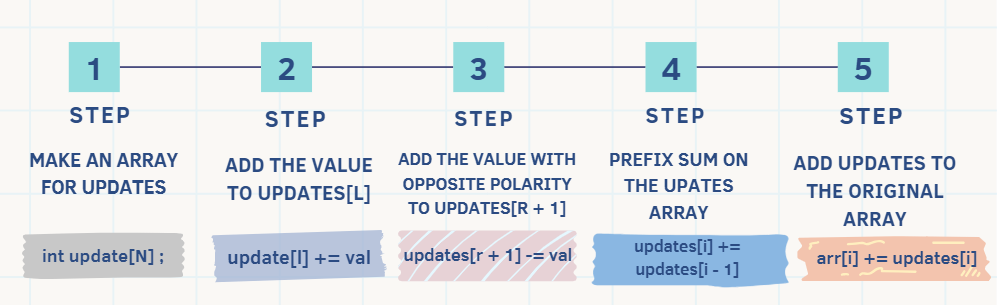
Partial vs Prefix ??

Actually the **Prefix sum** is an important step within the **Partial sum.**

THERE’S NO PARTIAL WITHOUT PREFIX

Use cases

Can be used to determine the max within an interval of time, having a **variable increasing or decreasing.**



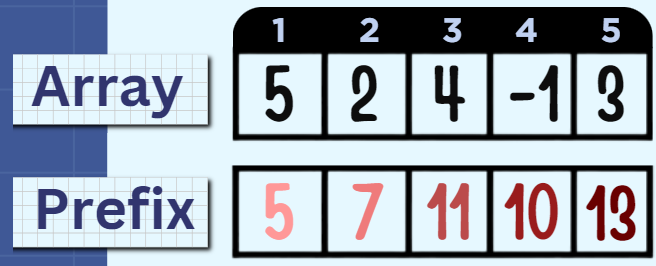
**Prefix Sum:**

What’s the idea behind it?

Each element in the *prefix* is the **sum** from the start to the element

Use cases? (for instance)

When working with **ranges**, like calculating the **sum between two numbers [l , r]**



Adhocs