Exercise 1: Sorting Array in Ascending Order

In this exercise, you will be required to use pointers to sort the values of an array in ascending order

We'll cover the following

Problem Statement

Let's start with a *basic* example.

Write a C++ *function* called `sortAsc` to **sort** an *array* of **ten** *integer* values in **ascending** order.

The *function* `sortAsc` will accept **two** *arguments*

- *First*, a **pointer** that points to the **array**.
- *Second*, the **array size**.

Fucntion Return

The function `sortAsc` *returns* a **pointer** that points to the **sorted array**.

Example Input

The array given to you as **input** in the exercise below is:

```
int arr[] = {23,34,2,3,5,12,42,56,89,8};
```

Expected output

```
Values in ascending order are: {2, 3, 5, 8, 12, 23, 34, 42, 56, 89}
```

Hint

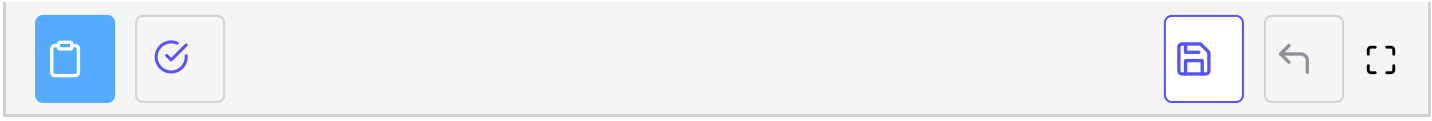
For ease break down the problem into parts:

- **First** figure out how the `sortAsc` will be declared which includes
  - Setting the *return* type of the function
  - Passing the **appropriate** *parameters* to function
- **Next**, come up with the logic on how to *sort* a simple array in **ascending** order.
- **Lastly** try *implementing* the same logic but using *pointers*.

Write your code below. It is recommended that you try solving the exercise yourself before viewing the solution.

Good Luck!

```
int *sortAsc(int *arr, int n) {
  //write code here
    return arr;
}
```



Solution Explanation

As explained in the problem statement above:

- We call our *function* `sortAsc` and pass our *array* `arr` and it's size **10** to it as *parameters*.
- The function above has the *return* type *pointer* that points to the array. Hence the `int*` *return* type is written.
- We use *nested* loops to iterate over the *array*.
- We compare **each** value of the *array* with **all** the values **after** it in the *array*.
- If the **next** value is **less** then the *current* value we **swap** them.
- The *function* then *returns* the **updated** `p`.
- At the end, we *display* the values by *dereferencing* the *array* values pointed to by *pointer* `p`.