BUBBLE SORT

ALGORITHM

What is Bubble Sort?

- Bubble sort is one of the fundamental forms of sorting in programming.

- Bubble sort algorithms move through a sequence of data and rearrange them into ascending or descending order one number at a time.

Why is it called Bubble Sort?

- The name bubble sort comes from the fact that smaller or larger elements "bubble" to the top of a dataset.

- This algorithm is alternatively called the sinking sort for the opposite reason; some of the elements are sinking to the bottom of the dataset.

ADVANTAGES

Simplicity

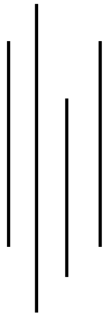
DISADVANTAGES

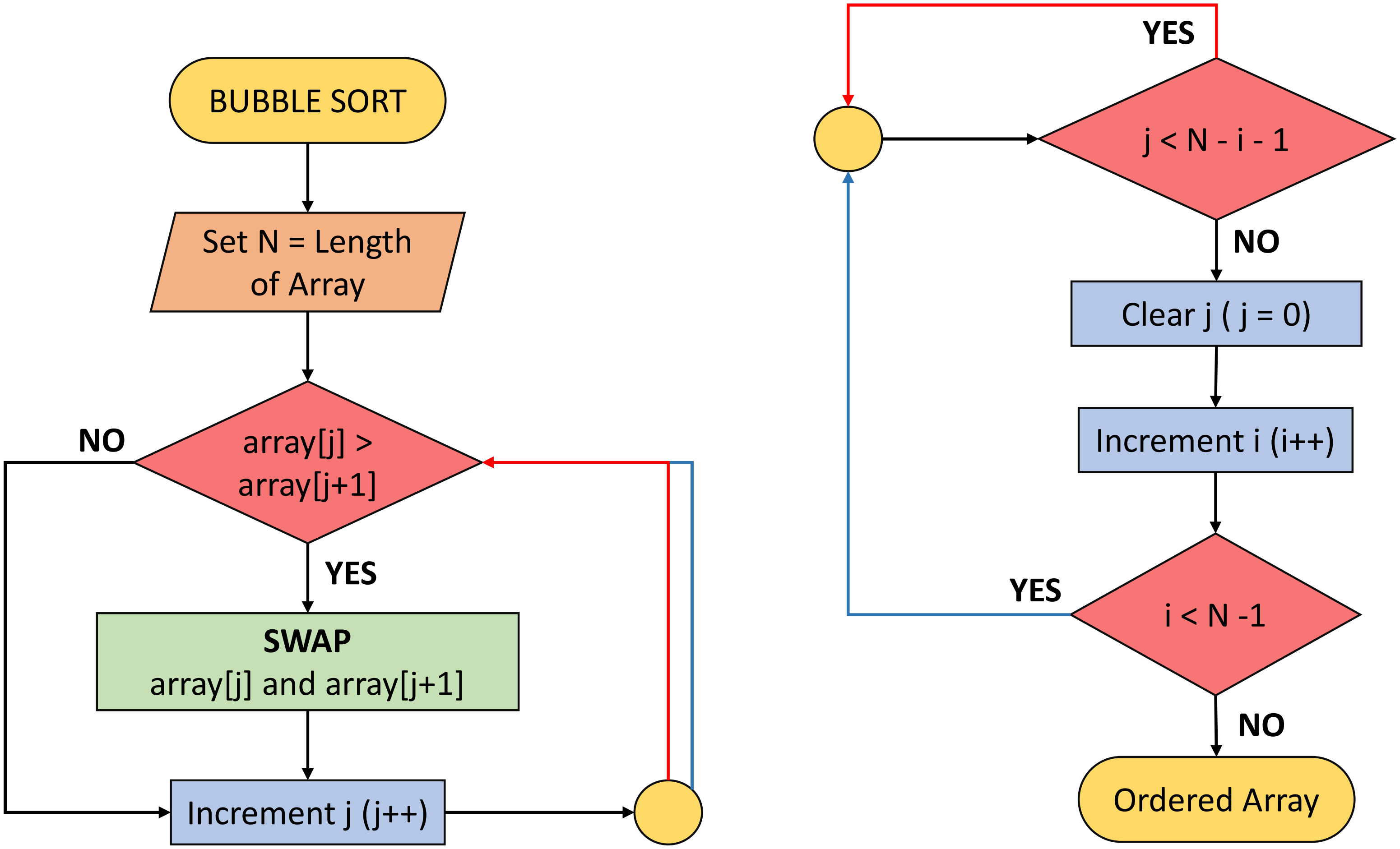
Inefficient for larger set of numbers

ALGORITHM

$O(n^2)$

```
for i from 1 to N
    for j from 0 to N – 1
        if array[j] > array[j + 1]
            swap(array[j], array[j + 1])
```

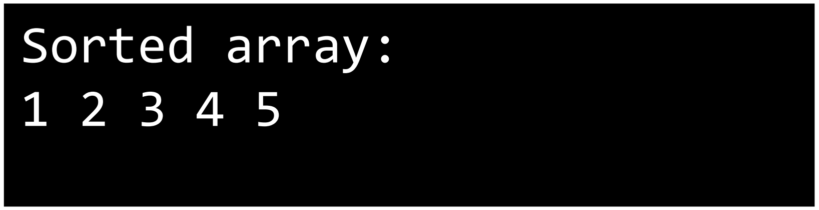



BUBBLE SORT
Set N = Length of Array
array[j] > array[j+1]
NO
YES
SWAP
array[j] and array[j+1]
Increment j (j++)
YES
j < N - i - 1
NO
Clear j (j = 0)
Increment i (i++)
YES
i < N -1
NO
Ordered Array

```
void bubbleSort(int arr[], int n)
{
    for(int i = 0; i < n - 1; i++)
        for(int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
```

```
void printArray(int arr[], int size)
{
    for(int i = 0; i < size; i++)
        cout << arr[i] << " “;
}
```

```
int main()
{
    int arr[] = {4, 2, 5, 3, 1};
    int N = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, N);
    cout <<"Sorted array: \n";
    printArray(arr, N);
    return 0;
}
```



```
Sorted array:
1 2 3 4 5
```

	4	2	5	3	1
array[j] > array[j+1]: TRUE;	4	2	5	3	1
SWAP: array[j] and array[j+1];	2	4	5	3	1
Increment j (j++); j < N - i - 1: TRUE; array[j] > array[j+1]: FALSE;	2	4	5	3	1
Increment j (j++); j < N - i - 1: TRUE; array[j] > array[j+1]: TRUE;	2	4	5	3	1
SWAP: array[j] and array[j+1];	2	4	3	5	1
Increment j (j++); j < N - i - 1: TRUE; array[j] > array[j+1]: TRUE;	2	4	3	5	1
SWAP: array[j] and array[j+1];	2	4	3	1	5
Increment j (j++); j < N - i - 1: FALSE;	2	4	3	1	5
Clear j (j = 0); Increment i (i++); i < N - 1: TRUE;	2	4	3	1	5

	2	4	3	1	5
array[j] > array[j+1]: FALSE;	2	4	3	1	5
Increment j (j++); j < N - i - 1: TRUE; array[j] > array[j+1]: TRUE;	2	4	3	1	5
SWAP: array[j] and array[j+1];	2	3	4	1	5
Increment j (j++); j < N - i - 1: TRUE; array[j] > array[j+1]: TRUE;	2	3	4	1	5
SWAP: array[j] and array[j+1];	2	3	1	4	5
Increment j (j++); j < N - i - 1: FALSE;	2	3	1	4	5
Clear j (j = 0); Increment i (i++); i < N - 1: TRUE;	2	3	1	4	5

	2	3	1	4	5
array[j] > array[j+1]: FALSE;	2	3	1	4	5
Increment j (j++); j < N - i - 1: TRUE; array[j] > array[j+1]: TRUE;	2	3	1	4	5
SWAP: array[j] and array[j+1];	2	1	3	4	5
Increment j (j++); j < N - i - 1: FALSE;	2	1	3	4	5
Clear j (j = 0); Increment i (i++); i < N - 1: TRUE;	2	1	3	4	5

	2	1	3	4	5
array[j] > array[j+1]: TRUE;	2	1	3	4	5
SWAP: array[j] and array[j+1];	1	2	3	4	5
Increment j (j++); j < N - i - 1: FALSE;	1	2	3	4	5
Clear j (j = 0); Increment i (i++); i < N - 1: FALSE;	1	2	3	4	5

CONCLUSION

- The simplest sorting algorithm
- Works by repeatedly swapping the adjacent elements if they are in the wrong order
- Not suitable for large data sets
- Worst-case time complexity is quite high.