```
1
2
3   Sorting ‘Algorithms’ {
4
5
6
7
8
9       < We don’t have better algorithms, we just
10      have more data>
11
12  }
13
14
```

01 {

[Bubble Sort]

< **A simple sorting algorithm** used to rearrange a set of elements in ascending or descending order. >

}

Introduction; {

‘Bubble sort is a basic algorithm for arranging a string of numbers or other elements in the correct order.’

<p The method works by examining each set of adjacent elements in the string, from left to right, switching their positions if they are out of order.
“The algorithm then repeats this process until it can run through the entire string and find no two elements that need to be swapped.”

Useful for smaller sets, inefficient for larger sets.

</p>

}

Concept < /1 > {

< If Bubble Element is out of order with next element, swap the bubble with next element. >

}

Concept < /2 > {

< If Bubble Element is in order with next element, there'll be no swapping & bubble will be the next element. >

}

Pseudo Code

```
1.  BubbleSort (Array[], size)
2.      for i = 0 to i < size-1
3.          for j = 0 to j < size-i-1
4.              if(Array[j] > Array[j+1])
5.                  swap(Array[j], Array[j+1])
```

Ascending Order Sorting

1st Cycle

4 Comparisons

2nd Cycle

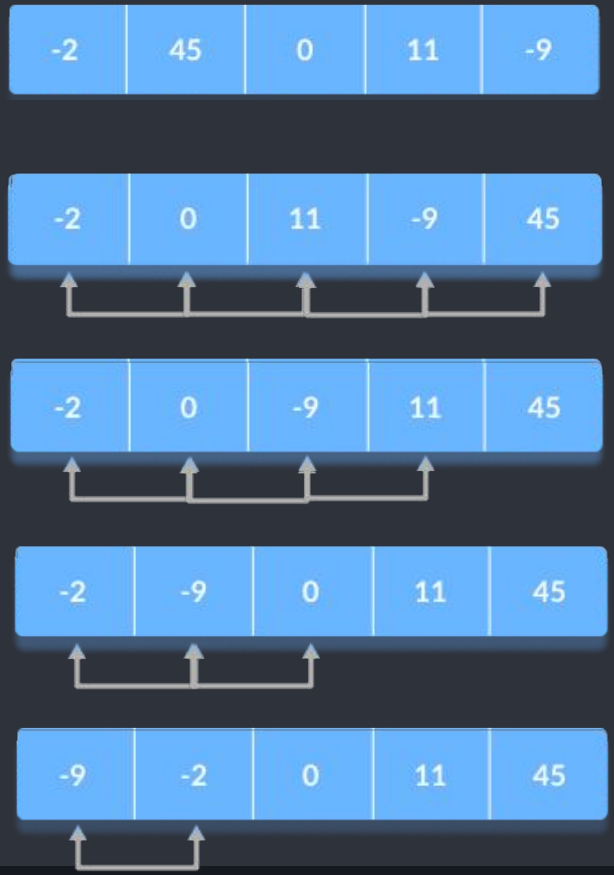
3 Comparisons

3rd Cycle

2 Comparisons

4th Cycle

1 Comparison



Ascending Order Sorting

1st Cyc
4 Compar

2nd Cyc
3 Compar

3rd Cyc
2 Compar

4th Cyc
1 Compar

Cycle	Number of Comparisons
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
.....	.....
last	1

Recursive Implementation

```
void bubblesort(int array[], int size) {
    // inCase there's only 1 or 0 element in array
    if (size == 0 || size == 1)
        return;

    for (int i = 0; i < size - 1; i++)
        // Bubble = array[i]
        if (array[i] > array[i + 1]) {
            swap(array[i], array[i + 1]);
        }

    // recursive call for sub-Array, excluding last
    element because it has reached at right position
    bubblesort(array, size - 1);
}
```

What about this?

Optimized Recursive Implementation

```
void bubblesort(int array[], int size) {
    // inCase there's only 1 or 0 element in array
    if (size == 0 || size == 1)
        return;

    bool swapped = false;
        // Bubble = array[i]
        if (array[i] > array[i + 1]) {
            swap(array[i], array[i + 1]);
        }
            swapped = true;
    // recursive call for sub-Array, excluding last

    if (!swapped) // no swapping occured, array is sorted
        return;
```

Optimized Recursive Implementation

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```c
void bubblesort(int array[], int size) {
    // inCase there's only 1 or 0 element in array
    if (size == 0 || size == 1)
        return;

    bool swapped = false;
    for (int i = 0; i < size - 1; i++)
        // Bubble = array[i]
        if (array[i] > array[i + 1]) {
            swap(array[i], array[i + 1]);
            swapped = true;
        }

    if (!swapped) // no swapping occured, array is sorted
        return;

    // recursive call for sub-Array, excluding last
    element because it has reached at right position
    bubblesort(array, size - 1);
}
```

Iterative Implementation

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

Time Complexity	
Best	O(n)
Worst	$O(n^2)$
Average	$O(n^2)$
Space Complexity	O(1)
Stability	Yes

The ***Worst-Case*** condition for bubble sort occurs when elements of the array are arranged in decreasing order.

1. Time Complexities

- **Worst Case Complexity:** `O(n²)`
  If we want to sort in ascending order and the array is in descending order then the worst case occurs.
- **Best Case Complexity:** `O(n)`
  If the array is already sorted, then there is no need for sorting.
- **Average Case Complexity:** `O(n²)`
  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

2. Space Complexity

- Space complexity is `O(1)` because an extra variable is used for swapping.
- In the **optimized bubble sort algorithm**, two extra variables are used. Hence, the space complexity will be `O(2)`.