Sorting

Comparison Based

Comparison based sorting: sorting based on the comparison of two items
In place sorting

- Sorting of data structure does not require any external data structure for sorting the intermediate steps
  External sorting
- Sorting of records not present in memory
  Stable sorting
- If the same element is present multiple times, then they retain the original positions

Stable
input- 2, 3, *1*, 15, 11, 23, **1**
output- *1*, **1**, 2, 3, 11, 15, 23

Not Stable
input- 2, 3, *1*, 15, 11, 23, **1**
output- **1**, *1*, 2, 3, 11, 15, 23

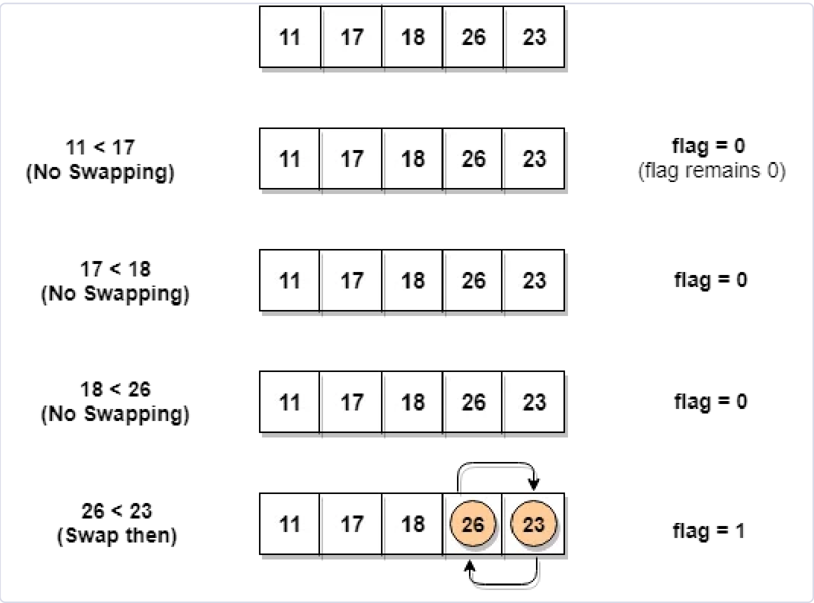
Some Sorting Algorithms

Simple sorting algorithms, performing only adjacent exchanges. Bubble sort and insertion sort are examples of this.

Bubble Sort

- Simple and uncomplicated
- compare to neighbor, swap if x is greater than y

Step	View
Step 1	2 3 1 15
Step 2	2 1 3 15
Step 3	1 2 3 15
Step 4	1 2 3 5



enhanced-bubble-sort.webp

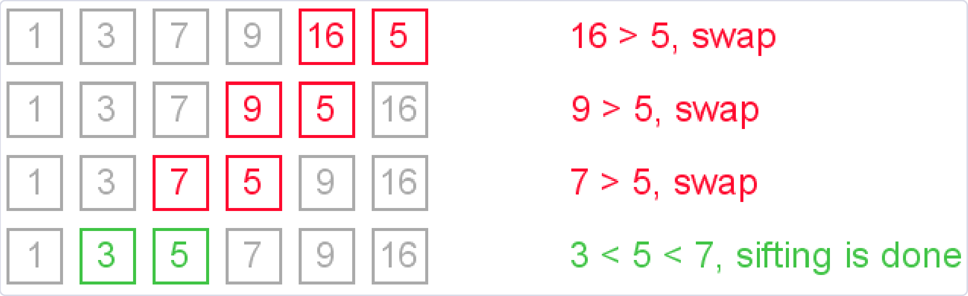
```
// bubble sort
int i, j;
for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
                if (arr[j] > arr[j + 1])
                        swap(arr[j], arr[j + 1]);
```

Insertion Sort

- insert one element at a time
- If reversely sorted you will need to swap every item
- $O(N^2)$ Time Complexity
- $O(N)$ Best Time Complexity
- Good for if data is almost sorted

Step	View
Step 1	8 34 64 51 32 21
Step 2	8 34 64 51 32 21
Step 3	8 32 34 51 64 21
Step 4	8 21 32 34 51 64



```
// insertion sort
int i, key, j;
```

```
for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key,
        // to one position ahead of their
        // current position
        while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
        }
        arr[j + 1] = key;
}
```

Shell Sort

- A sorting algorithm that allows for comparison of not adjacent items
- `h-sort` all elements spaced `h` apart are sorted
- Performing h-sort using insertion sort, the items compared are not longer adjacent - potential for improvement

```C++
for (int gap = n/2; gap > 0; gap /= 2)
{
        // Do a gapped insertion sort for this gap size.
        // The first gap elements a[0..gap-1] are already in gapped order
        // keep adding one more element until the entire array is
        // gap sorted
        for (int i = gap; i < n; i += 1)
        {
                // add a[i] to the elements that have been gap sorted
                // save a[i] in temp and make a hole at position i
                int temp = arr[i];

                // shift earlier gap-sorted elements up until the correct
                // location for a[i] is found
                int j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                        arr[j] = arr[j - gap];

                //  put temp (the original a[i]) in its correct location
                arr[j] = temp;
        }
}
return 0;
```