Data Structures and Algorithms

Lab 6
Sorting Algorithms

Agenda

- Sorting Algorithms:
  - Bubble sort
  - MergeSort
  - QuickSort
- Codeforces!

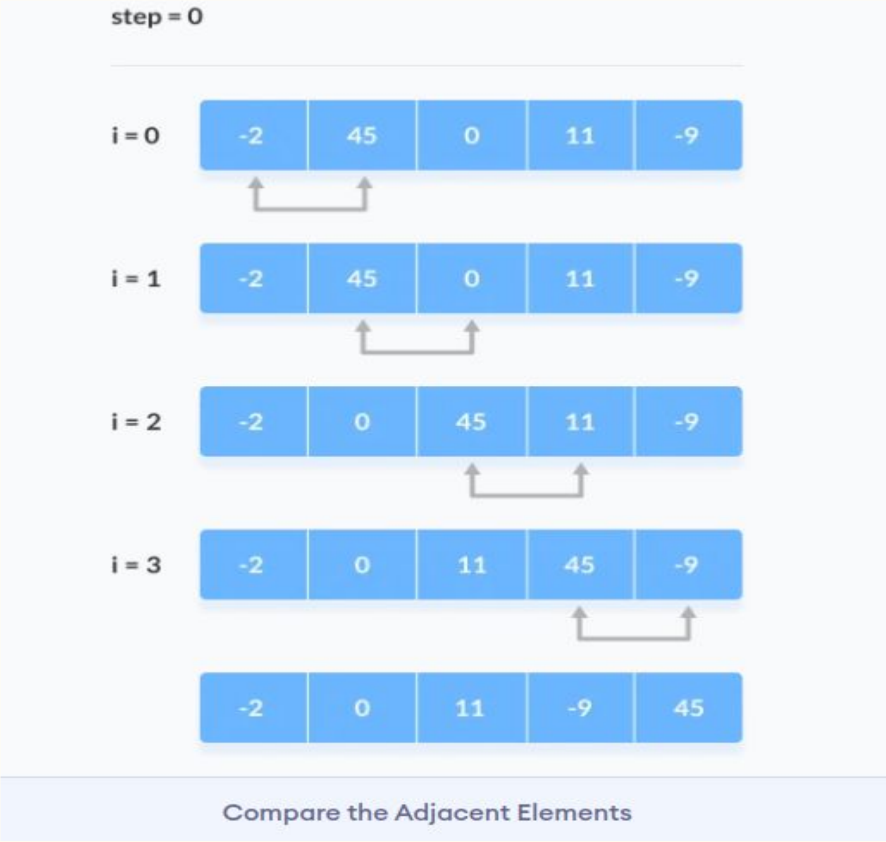
Bubble Sort

- **6.1.** Give the principle of the Bubble Sort algorithm.
- **6.2.** Determine the time complexity of the Bubble Sort algorithm.

Bubble Sort

1. Starting from the first index, compare the first and the second elements.
2. If the first element is greater than the second element, they are swapped.
3. Now, compare the second and the third elements. Swap them if they are not in order.
4. The above process goes on until the last element.
5. The same process goes on for the remaining iterations.
6. The array is sorted when all the unsorted elements are placed at their correct positions.

Bubble Sort



step = 0
i = 0
-2
45
0
11
-9
i = 1
-2
45
0
11
-9
i = 2
-2
0
45
11
-9
i = 3
-2
0
11
45
-9
-2
0
11
-9
45
Compare the Adjacent Elements

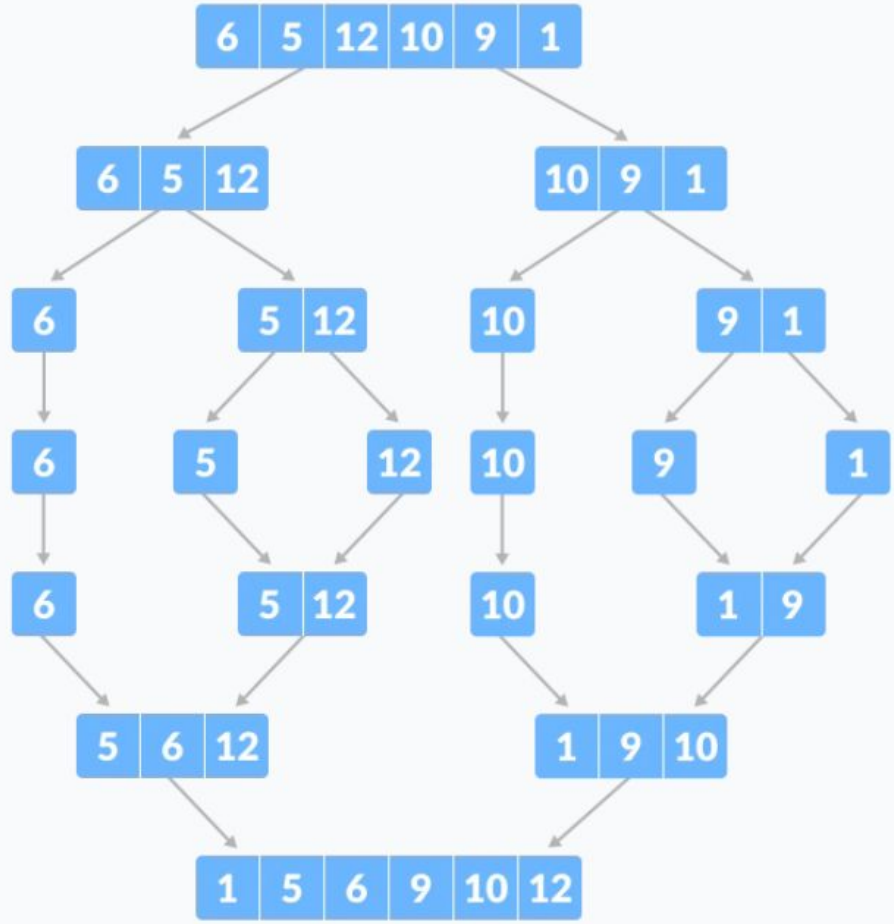
Bubble Sort

```
BubbleSort(Array, n)
{
    for i = 0 to n-2
    {
        for j = 0 to n-2
        {
            if Array[j] > Array[j+1]
            {
                swap(Array[j], Array[j+1])
            }
        }
    }
}
```

Merge sort:

1. The MergeSort function repeatedly divides the array into two halves until we reach a stage where we try to perform MergeSort on a subarray of size 1 i.e. p == r.
2. After that, the merge function comes into play and combines the sorted arrays into larger arrays until the whole array is merged.

```
MergeSort(A, p, r):
    if p > r
        return
    q = (p+r)/2
    mergeSort(A, p, q)
    mergeSort(A, q+1, r)
    merge(A, p, q, r)
```



Merge Sort example

The merge Step of Merge Sort:

The merge step is the solution to the simple problem of merging two sorted lists(arrays) to build one large sorted list(array).

```
Have we reached the end of any of the arrays?
    No:
        Compare current elements of both arrays
        Copy smaller element into sorted array
        Move pointer of element containing smaller element
    Yes:
        Copy all remaining elements of non-empty array
```

Since there are no more elements remaining in the second array, and we know that both the arrays were sorted when we started, we can copy the remaining elements from the first array directly.

Merge step

QuickSort

- Divide Step
- Conquer Step
- Combine Step

QuickSort

- Divide Step
  - Pick an element (pivot)
  - Put everything **less** than pivot to left
  - Put everything **greater** than pivot to right
  - You got two subarrays!
- Conquer Step
- Combine Step

QuickSort

- Divide Step
  - Pick an element (pivot)
  - Put everything **less** than pivot to left
  - Put everything **greater** than pivot to right
- Conquer Step
  - Apply the recursive algorithm and sort subarrays
- Combine Step
  - Concatenate results

QuickSort: Example

2	1	0	5	10	3	Input

QuickSort: Example

2	1	0	5	10	3

Input

2	1	0	3	10	5

QuickSort: Example

2	1	0	5	10	3

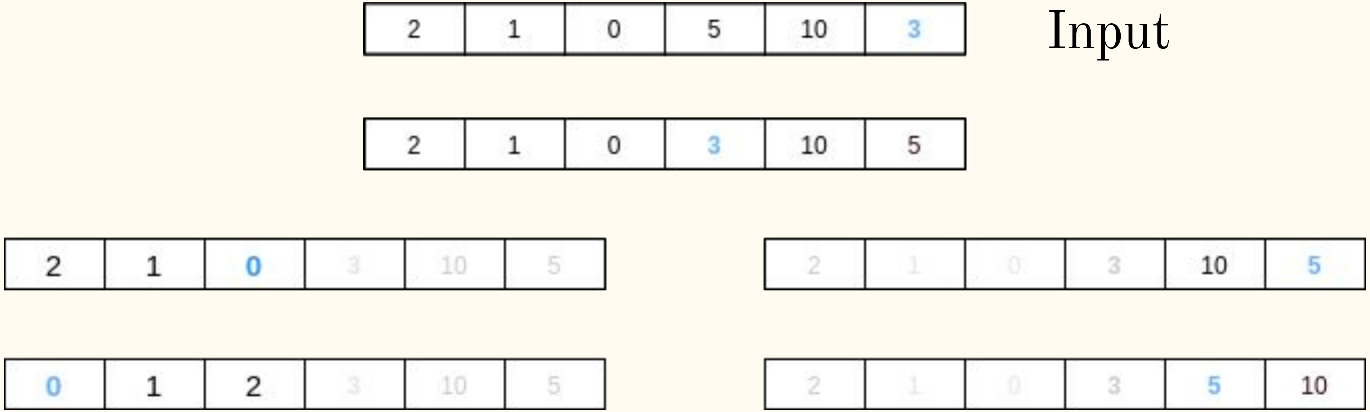
Input

2	1	0	3	10	5

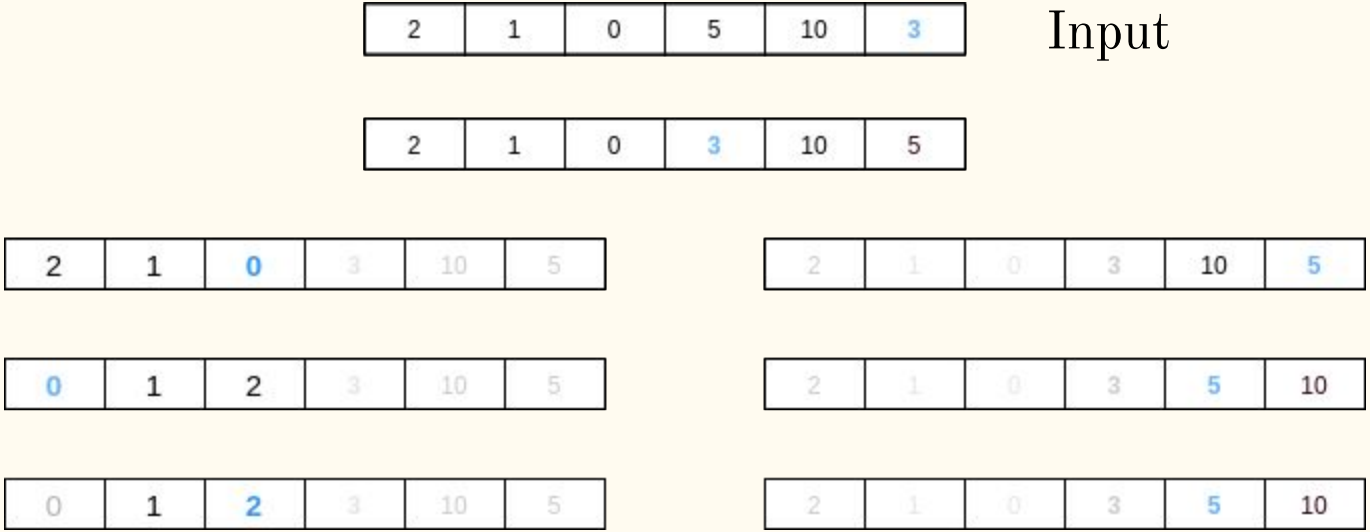
2	1	0	3	10	5

2	1	0	3	10	5

QuickSort: Example

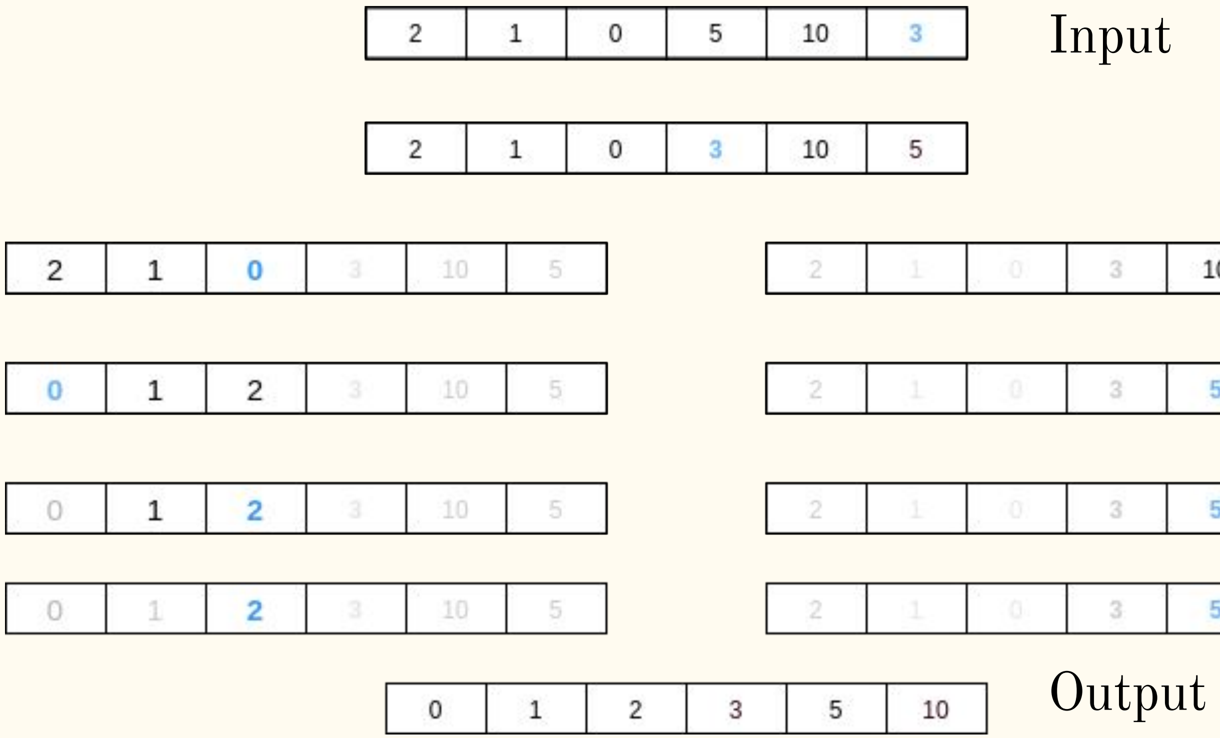


QuickSort: Example



Input

QuickSort: Example

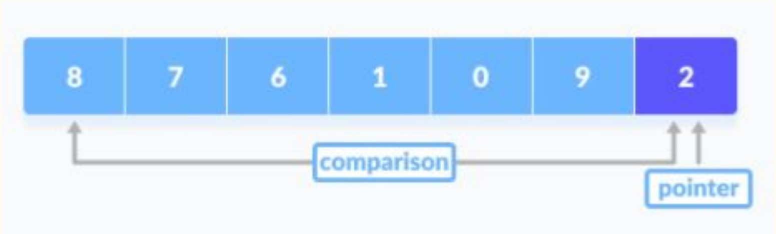


QuickSort: In-place vs not in-place

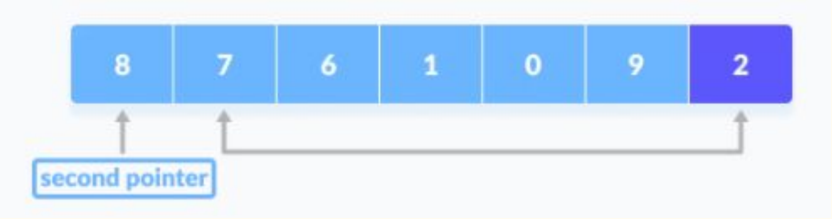
- In-place quick sort
- Out of place quick sort

QuickSort: In-place

- A pointer is fixed at the pivot element. The pivot element is compared with the elements beginning from the first index.



- If the element is greater than the pivot element, a second pointer is set for that element.

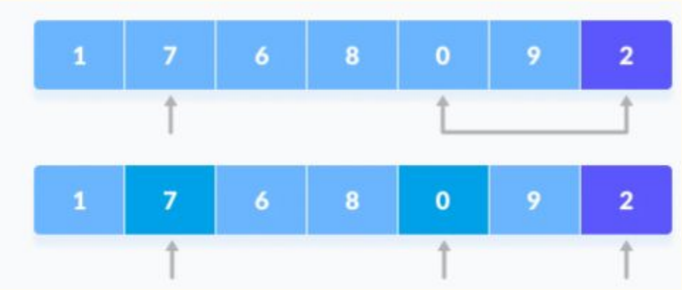


QuickSort: In-place

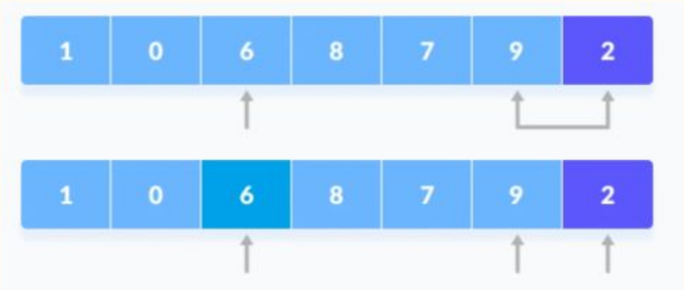
- Now, the pivot is compared with other elements. If an element smaller than the pivot element is reached, the smaller element is swapped with the greater element found earlier.

QuickSort: In-place

- Again, the process is repeated to set the next greater element as the second pointer. And, swap it with another smaller element.



- The process goes on until the second last element is reached.



QuickSort: In-place

- Finally, the pivot element is swapped with the second pointer.

QuickSort: Recursion

```java
private static void quick(int[] arr, int low, int high) {
    if (low < high) {
        int pivot = partition(arr, low, high);
        quick(arr, low, pivot - 1);
        quick(arr, pivot+1, high);
    }
}
```

QuickSort: Partition

```
static int partition(int[] arr, int low, int high) {
    int x = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < x) {
            i++;
            swap(arr, i, j);
        }
    }
    swap(arr, i + 1, high);
    return i + 1;
}
```

QuickSort: pivot selection and time complexity

- pivot is (at each iteration):
  - Last (max)
  - First (min)
  - middle (random)

6.3. How does it affect the time complexity?

QuickSort: pivot selection and time complexity

- Worst-case Partitioning

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) . \end{aligned}$$

$O(n^2)$

- Best-case Partitioning

$$T(n) = 2T(n/2) + \Theta(n) ,$$

$O(nlogn)$

Sorting: Time complexity

	Algorithm	Worst-case running time	Average-case/expected running time
	Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Comparison sorting	Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
	Heapsort	$O(n \lg n)$	—
	Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)
	Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Integer sorting	Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
	Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)

CodeForces

Problems

#	Name		
A	Online item bids	standard input/output 2 s, 256 MB	

https://codeforces.com/group/C71Rz4W66e/contest/317694

See You next week!