

15

18/04/24

April

16th Wk • 105-260

Friday

Sorting in Java

			1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21	22	23
25	26	27	28	29	30							

APRIL

M	T	W	T	F	S	S	M	T	W	T	F	S
---	---	---	---	---	---	---	---	---	---	---	---	---

Bubble Sort

8

7	8	3	1	2
---	---	---	---	---

9 It is a sorting algorithm where there shall be $(n-1)$ comparisons, provided there are ' n ' elements present.

10 In each iteration, every elements (starting from the first) is compared with its next element and the largest element is pushed to the right end.

11 With every operation, the comparable positions are reduced because the largest element in the previous iteration is already pushed to the right end.

1 Step 1 -

7, 8, 3, 8, 1, 8, 2, 8	→	7 3, 1, 2, 8
------------------------	---	----------------

2 Step 2 →

3, 7, 1, 7, 2, 7, 8	→	3 1, 2, 7, 8
---------------------	---	----------------

3 Step 3 →

1, 3, 2, 3, 7, 8	→	1 2, 3, 7, 8
------------------	---	----------------

4 Step 4 →

1, 2, 3, 7, 8	→	1 2 3 7, 8
---------------	---	------------------

✓ So, there are 5 elements, and we made a total no.

6 of $(5-1) \cdot 2 = 4$ Comparisons

every

✓ Also, after iteration, the last position is sorted and hence not touched in the next operation. So, after step 1, the last position is sorted. After step 2, the 2nd last position is also sorted and so on.

✓ Time complexity is $O(n^2)$, which is not a good compared to other sorting techniques algorithm.

2022

MAY	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22				
23	24	25	26	27	28	29	30	31
MON	TUE	WED	THU	FRI	SAT	SUN	MON	TUE

* time complexity = $O(n^2)$.
 In asymptotic notations,
 we ignore the constants,
 so, if it is $O(n^2)$.

16th Wk • 106-259

110 14

16
Saturday

Code 5 -
class bubbleSort

8 public static void printArray (int arr[]) {

9 for (int i=0; i<arr.length; i++)

10 { System.out.println(arr[i] + " "); }

11 } System.out.println();

12 public static void main (String[] args)

{

1 int arr[] = { 7, 8, 3, 1, 2 };

2 for (int i=0; i<arr.length-1; i++) // n-1

3 { for (int j=0; j<arr.length-i-1; j++) // inner loop

{

5 if (arr[j] > arr[j+1])

{

6 int temp = arr[j]; // swap

7 arr[j] = arr[j+1];

8 arr[j+1] = temp;

9 } printArray (arr);

* Outer loop runs for $(n-1)$ times.
 * Inner loop runs for $n-1, n-2, n-3$, and so on.
 ignoring constants, we get $O(n^2)$.

17

April

16th Wk • 107-258

Sunday

2022

7M

				1	2	3	4	5	6	7	8	9
11	12	13	14	15	16	17	18	19	20	21	22	23
25	26	27	28	29	30							
M	T	W	T	F	S	S	M	T	W	T	F	S

Selection Sort

In this sorting technique, with every iteration, swapping happens only once per iteration. (The smallest element is found out and pushed to the left. Then, the 2nd smallest element is found, and pushed to the 2nd last and so on.)

Given:-

7	8	3	1	2
---	---	---	---	---

Step 1 :- $i=0$ (we assume) that smallest element is at $i=0$.

7	8	3	1	2
---	---	---	---	---

=

7	8	3	1	2
---	---	---	---	---

$i=0$ element is compared with all other elements & smallest pushed to the left only 1 swap per iteration

Step 2 :-

$i=1$ (we assume that smallest is at $i=1$)

7	8	7	1	7	1	7	1	7
---	---	---	---	---	---	---	---	---

7	8	3	7	2
---	---	---	---	---

$i=1 \rightarrow '8'$ compared to all other elements, and smallest = '2' pushed to the 2nd left position at $i=1$.

1	2	3	7	8
---	---	---	---	---

Step 3 :-

7	8	7	1	7	1	7	1	7
---	---	---	---	---	---	---	---	---

1	2	3	7	8
---	---	---	---	---

Now $i=2$ element compared with right side elements and no swapping happened as $i=2$ itself is the smallest, so,

1	2	3	7	8
---	---	---	---	---

Similarly in step 4 & 5, $i=4$ th and $i=5$ th element is compared and swapped if found a smaller element.

2022

	1	2	3	4	5	6	7	8
MAY	9	10	11	12	13	14	15	16
	17	18	19	20	21	22		
	23	24	25	26	27	28	29	30
	31							
WTWTFSS	SMTWTFSS							

April

17th Wk • 108-257

18
Monday

122/14

Code 2

```

public static void main (String args[])
{
    int arr [] = { 7, 8, 3, 1, 2 };
    for (int i = 0; i < arr.length - 1; i++)
    {
        int smallest = i;
        for (int j = i + 1; j < arr.length; j++)
        {
            if (arr [smallest] > arr [j])
            {
                smallest = j;
            }
        }
        int temp = arr [smallest];
        arr [smallest] = arr [i];
        arr [i] = temp;
    }
}

```

Program

* Time complexity = $O(n^2)$.
The inner loop runs $O(n, n-1, n-2 \dots n-m+1)$ times.
and the outer loop runs approx. n times.
we can say that it's like an A.P series.

19

April

17th Wk • 109-256

Tuesday

2022

113

S	T	W	F	S	S	M	T	W	F	S	S
11	12	13	14	15	16	17	18	19	20	21	22
25	26	27	28	29	30						

Insertion Sort

8

total array

9

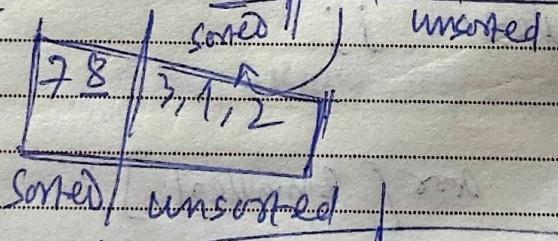
sorted unsorted

- 10 In Insertion Sort, we divide the whole array in 2 parts → sorted and unsorted. We keep on adding 1 by 1 element from the unsorted array to the sorted part. In this manner, the sorted array keeps on getting larger and the unsorted array keeps on getting smaller. So, at the end, the entire array becomes sorted.

1

7 | 8 | 3 | 1, 2

2

Step 1 :-

3

4

5

6

Step 2 :-

3 | 7 | 8 | 1, 2

sorted | unsorted

Step 3 :-

1 | 3 | 7 | 8 |

sorted | unsorted

Step 4 :-

1 | 2 | 3 | 7 | 8 |

sorted array

	2022
MAY	
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	17
17	18
18	19
19	20
20	21
21	22
22	
23	24
24	25
25	26
26	27
27	28
28	29
29	30
30	31
M	T
T	F
F	S
S	M
M	T
T	F
F	S

April
17th Wk • 110-255

20
Wednesday

Code:-

Public static void main (String args[])

{

int arr = { 7, 8, 3, 1, 2 };

// insertion sort

for (int i = 1; i < arr.length; i++) {

 int current = arr[i];

 int j = i - 1;

 while (j >= 0 && current < arr[j])

 {

 arr[j + 1] = arr[j];

 j--;

}

// Placement

 arr[j + 1] = current;

}

Note:- Time complexity of insertion sort is also $O(n^2)$.

Apart from Bubble Sort, Selection Sort & Insertion Sort there are quick sort & merge sort, where the time complexity is $O(n \log n)$.