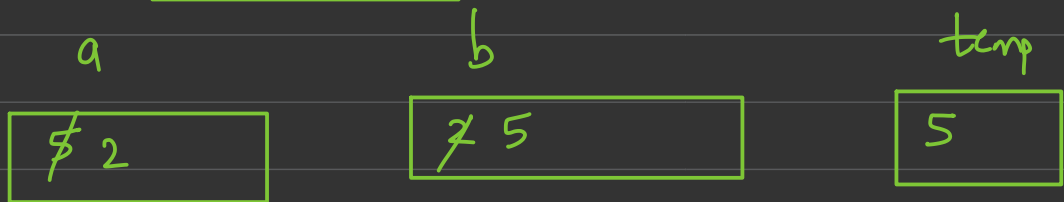
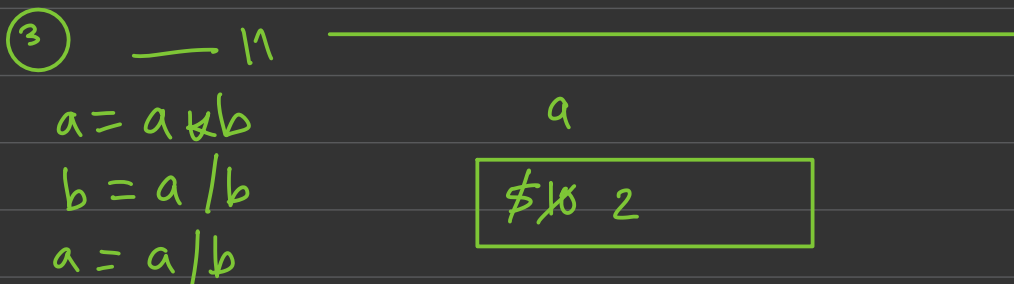
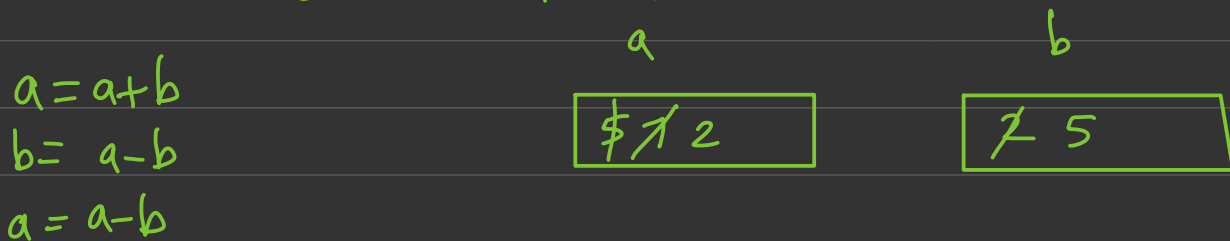


8 Apr 2023



① $\begin{matrix} \text{temp} = a \\ a = b \\ b = \text{temp} \end{matrix}$ } Using temp variable

② without using a temporary variable



NS 8 Apr 2023

J Main.java 1 x

Main.java > Main > bubblesort(int[])

```

1 // WAP to implement Bubble Sort to sort in Ascending order
2 import java.util.*;
3 public class Main
4 {
5     static void bubblesort(int arr[])
6     {
7         int n = arr.length; // 5
8         int temp;
9         for(int i=0; i<n-1; i++)
10             for(int j=0; j<n-i-1; j++)
11                 if(arr[j] > arr[j+1])
12                 {
13                     temp = arr[j];
14                     arr[j] = arr[j+1];
15                     arr[j+1] = temp;
16                 }
17     }
18
19     public static void main(String[] args)
20     {
21         int i, n;
22         Scanner sc = new Scanner(System.in);
23         System.out.print("Enter the number of elements: ");
24         n = sc.nextInt();
25         int arr[] = new int[n];
26         System.out.print("Enter the elements one by one: ");

```

Handwritten diagrams illustrating the first two passes of the Bubble Sort algorithm on the array [7, 6, 5, 4, 3]:

Pass I (PI):

7	6	5	4	3
6	7	5	4	3
6	5	7	4	3
6	5	4	7	3

Pass II (PII):

5	6	4	3	7
5	4	6	3	7
5	4	3	6	7

Run | Debug

Ln 8, Col 18 Tab Size: 4 UTF-8 LF () Java

NS 8 Apr 2023

J Main.java 1 x

Main.java > Main > bubblesort(int[])

```

78 // }
79 // }
80
81 // WAP to implement Bubble Sort to sort in Ascending order - Optimized
82 import java.util.*;
83 public class Main
84 {
85     static void bubblesort(int arr[])
86     {
87         int n = arr.length; // 5
88         int temp;
89         for(int i=0; i<n-1; i++)
90             for(int j=0; j<n-i-1; j++)
91                 if(arr[j] > arr[j+1])
92                 {
93                     temp = arr[j];
94                     arr[j] = arr[j+1];
95                     arr[j+1] = temp;
96                 }
97     }
98
99     public static void main(String[] args)
100     {
101         int i, n;
102         Scanner sc = new Scanner(System.in);
103         System.out.print("Enter the number of elements: ");

```

Handwritten diagrams illustrating the first four passes of the optimized Bubble Sort algorithm on the array [5, 4, 1, 3, 2]:

Pass I (PI):

5	4	1	3	2
4	5	1	3	2
4	1	5	3	2
4	1	3	5	2

Pass II (PII):

1	4	3	2	5
1	3	4	2	5
1	3	2	4	5

Pass III (PIII):

1	3	2	4	5
1	3	2	4	5
1	2	3	4	5

Pass IV (PIV):

1	2	3	4	5
---	---	---	---	---

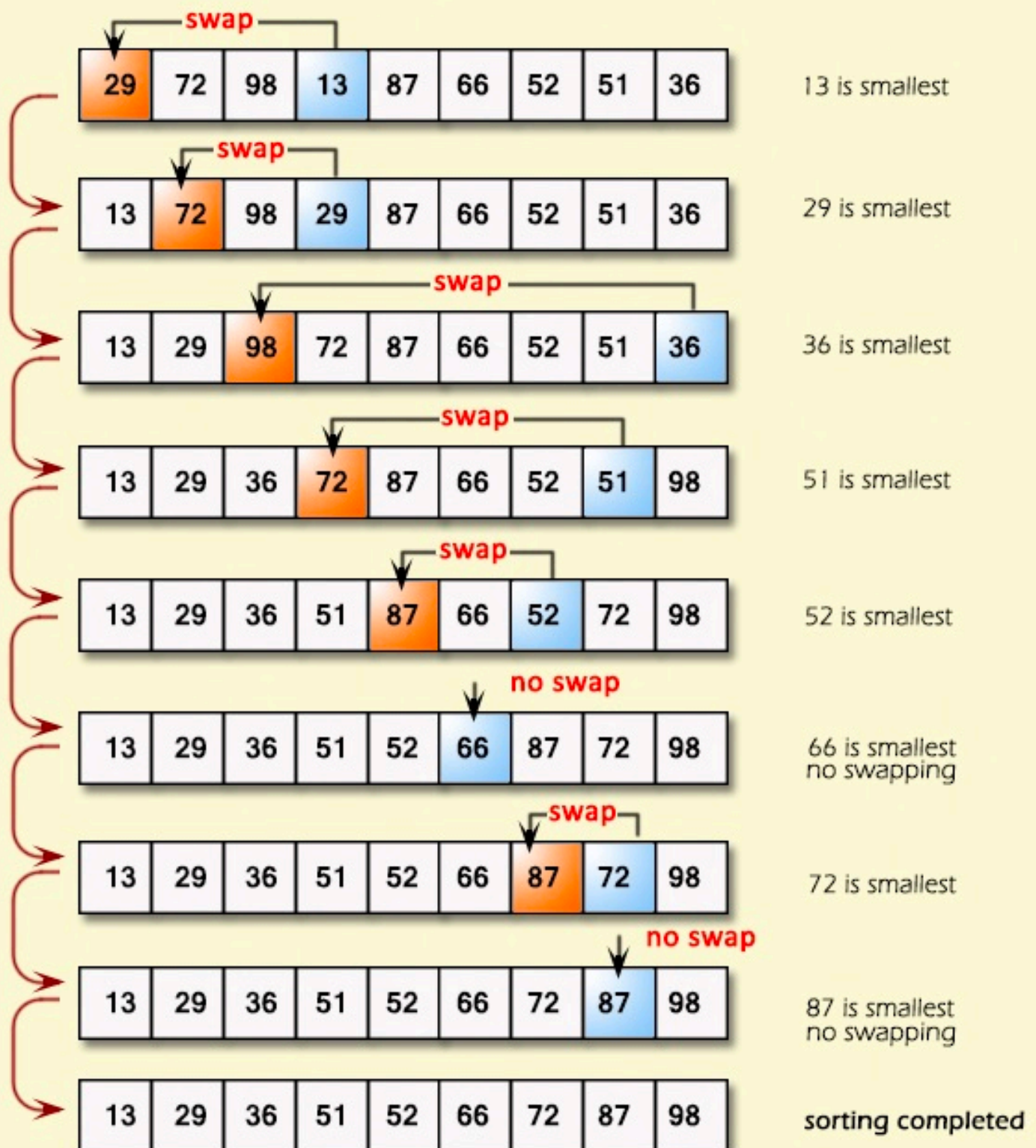
Run | Debug

Ln 90, Col 32 Tab Size: 4 UTF-8 LF () Java

The diagram illustrates the merge sort process on the array [22, 35, 17, 8, 13, 44, 5, 28]. The array is first divided into two halves: [22, 35, 17, 8] and [13, 44, 5, 28]. The process then shows the recursive sorting of these halves and their subsequent merging into a fully sorted array. The steps are as follows:

- Initial Array:** 22 | 35 | 17 | 8 | 13 | 44 | 5 | 28
- First Split:**
 - Left Half: 5, 35, 17, 8
 - Right Half: 13, 44, 22, 28
- Second Split (Left Half):**
 - Left: 5, 8
 - Right: 17, 35
- Second Split (Right Half):**
 - Left: 13, 17
 - Right: 35, 44
- Third Split (Left Half):**
 - Left: 5, 8
 - Right: 13, 17
- Third Split (Right Half):**
 - Left: 13, 17
 - Right: 22, 28
- Fourth Split (Left Half):**
 - Left: 5, 8
 - Right: 13, 17
- Fourth Split (Right Half):**
 - Left: 13, 17
 - Right: 22, 28
- Fifth Split (Left Half):**
 - Left: 5, 8
 - Right: 13, 17
- Fifth Split (Right Half):**
 - Left: 13, 17
 - Right: 22, 28
- Final Merging Steps:**
 - 5, 8 → 5, 8, 13, 17
 - 13, 17 → 13, 17, 22, 28
 - 5, 8, 13, 17 → 5, 8, 13, 17, 22, 28
 - 13, 17, 22, 28 → 13, 17, 22, 28, 35, 44
 - 5, 8, 13, 17, 22, 28 → 5, 8, 13, 17, 22, 28, 35, 44

Selection Sort



Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$, where n is the number of items.

