Java code that implements the Selection Sort algorithm. Here's a breakdown of the code line by line:

```java
import java.util.*;

public class tUf {
    static void selection_sort(int arr[], int n) {
        // Outer loop to iterate over each element of the array
        for (int i = 0; i < n - 1; i++) {
            // Assume the current index is the minimum
            int mini = i;

            // Inner loop to find the minimum element in the unsorted portion of
the array
            for (int j = i + 1; j < n; j++) {
                // Compare the current element with the assumed minimum
                if (arr[j] < arr[mini]) {
                    // If a smaller element is found, update the index of the
minimum
                    mini = j;
                }
            }

            // Swap the found minimum element with the first element of the
unsorted portion
            int temp = arr[mini];
            arr[mini] = arr[i];
            arr[i] = temp;
        }

        // Output the sorted array after the completion of the sorting process
        System.out.println("After selection sort:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String args[]) {
        // Sample array to be sorted
        int arr[] = {13, 46, 24, 52, 20, 9};
        int n = arr.length;

        // Output the original array before sorting
        System.out.println("Before selection sort:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        // Call the selection_sort method to sort the array
        selection_sort(arr, n);
```

```
        }
    }
```

In summary, the Selection Sort algorithm works by dividing the array into two portions - a sorted portion and an unsorted portion. In each iteration of the outer loop, it finds the minimum element in the unsorted portion and swaps it with the first element of the unsorted portion. This process is repeated until the entire array is sorted.

Here's a step-by-step breakdown of the Selection Sort algorithm implemented in the given Java code:

1. **Outer Loop (i):**
    - Iterate over each element in the array except the last one (up to `n - 1`).
    - This loop represents the sorted portion of the array.
2. **Assume the Current Index as the Minimum (mini):**
    - Initialize a variable `mini` to the current index `i`, assuming it holds the minimum value in the unsorted portion.
3. **Inner Loop (j):**
    - Iterate over the unsorted portion of the array (from `i + 1` to `n`).
    - Compare each element with the element at the assumed minimum index (`mini`).
    - If a smaller element is found, update the index of the minimum (`mini`).
4. **Swap Minimum Element with First Element of Unsorted Portion:**
    - After the inner loop, swap the element at the assumed minimum index (`mini`) with the first element in the unsorted portion (`arr[i]`).
    - This effectively expands the sorted portion and reduces the unsorted portion.
5. **Output the Sorted Array:**
    - After the outer loop completes, output the sorted array.
6. **Main Method:**
    - Initialize an array (`arr`) with unsorted values.
    - Output the original array before sorting.
7. **Call `selection_sort` Method:**
    - Call the `selection_sort` method to perform the sorting on the array.

In summary, the Selection Sort algorithm repeatedly finds the minimum element in the unsorted portion and swaps it with the first element of the unsorted portion until the entire array is sorted.

The time complexity of the Selection Sort algorithm implemented in the given code is O(n^2), where 'n' is the number of elements in the array.

Explanation:

1. **Outer Loop:** The outer loop runs for (n-1) iterations, where 'n' is the number of elements in the array.
2. **Inner Loop:** The inner loop also runs for (n-1) iterations on average, since it starts from the current outer loop index and goes up to the end of the array.
3. **Total Comparisons:** In each iteration of the outer loop, the inner loop compares elements in the unsorted portion to find the minimum.
    - 1st iteration: (n-1) comparisons
    - 2nd iteration: (n-2) comparisons
    - 3rd iteration: (n-3) comparisons
    - ...
    - (n-2)th iteration: 1 comparison
    - (n-1)th iteration: 0 comparisons
4. **Total Comparisons (Sum):** The total number of comparisons is the sum of the first (n-1) natural numbers, which is approximately (n^2)/2.
5. **Time Complexity:** The time complexity of the Selection Sort algorithm is O(n^2) because the dominant term in the total number of comparisons is n^2.

While Selection Sort has a quadratic time complexity, it is not the most efficient sorting algorithm for large datasets. There are more efficient algorithms, such as Merge Sort or Quick Sort, which have better average and worst-case time complexities.