Data Structures and Algorithms TICT2113

Brief Overview of Sorting Techniques

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Etc.

Selection Sort Algorithm

- Selection Sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.
- Time Complexity: O(n^2)
- Space Complexity: O(1)

Selection Sort Step-by-Step Explanation

- Step-by-Step Explanation:
  - Find the smallest element in the unsorted portion.
  - Swap it with the first element.
  - Repeat the process for the remaining unsorted portion.

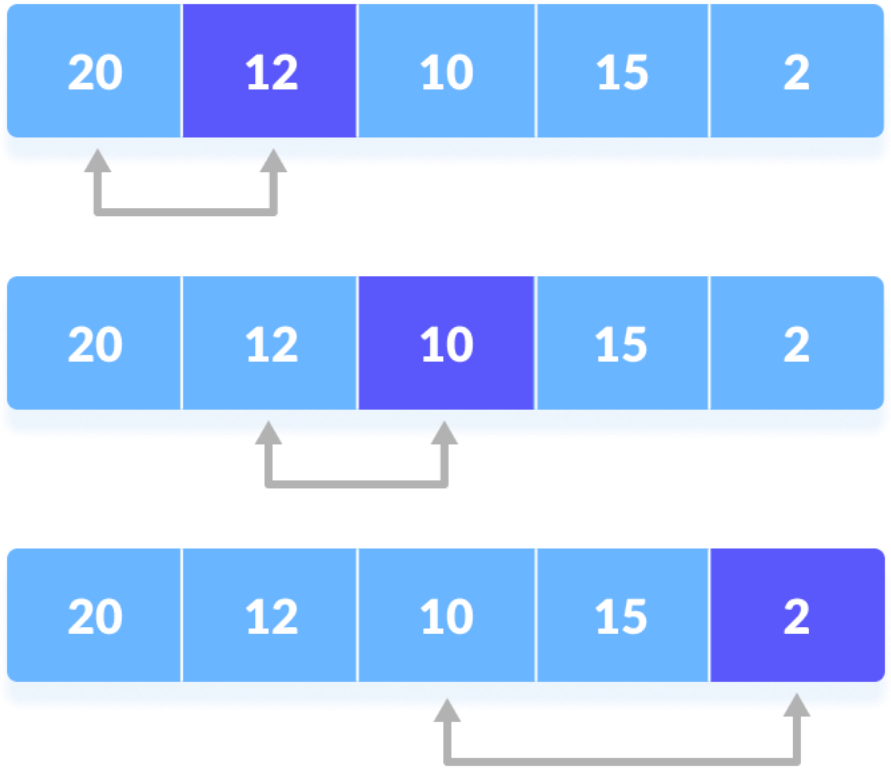
Working of Selection Sort

- Set the first element as minimum.

- Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.
- Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.

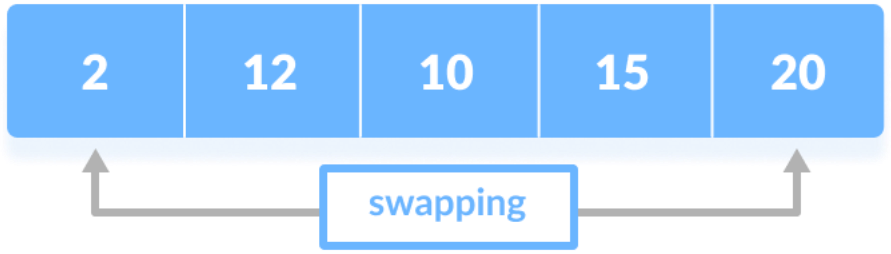
Working of Selection Sort (Cont.)

- Compare minimum with the remaining elements



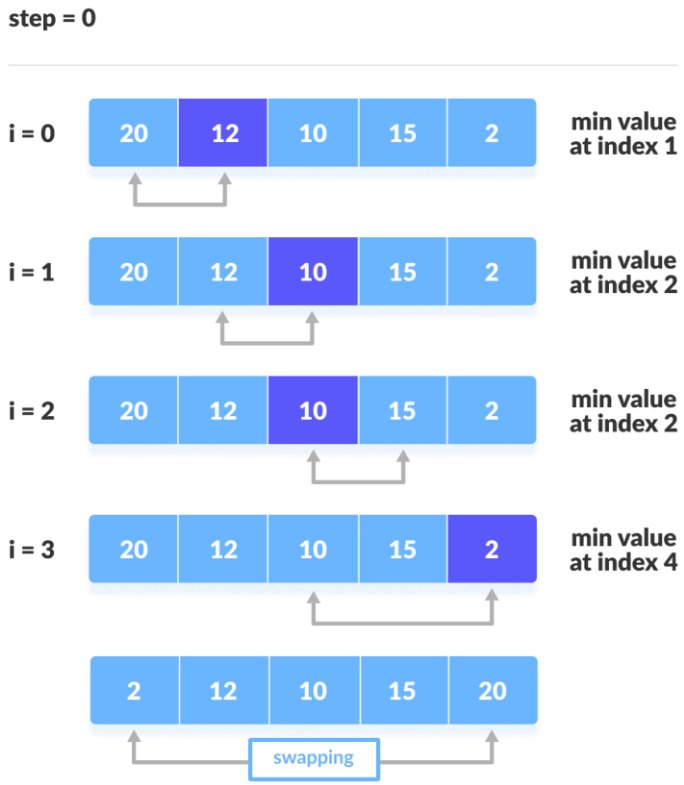
Working of Selection Sort (Cont.)

- After each iteration, minimum is placed in the front of the unsorted list.
- Swap the first with minimum.



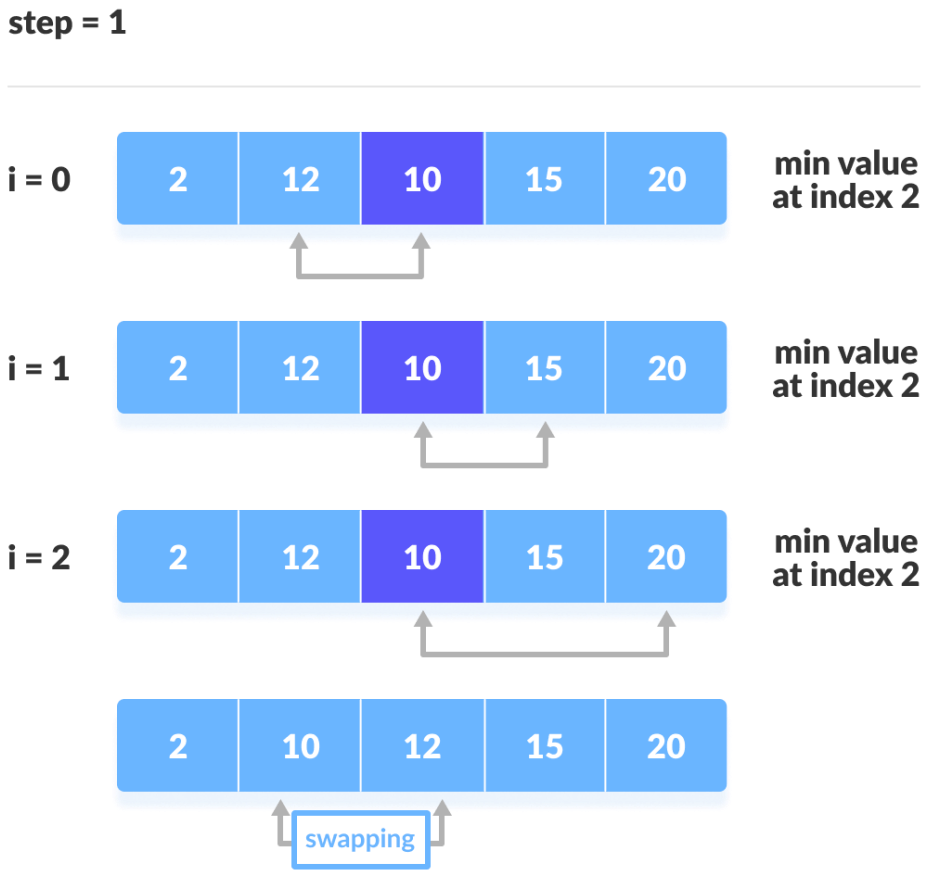
Working of Selection Sort (Cont.)

- For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.



Working of Selection Sort (Cont.)

- The second iteration.

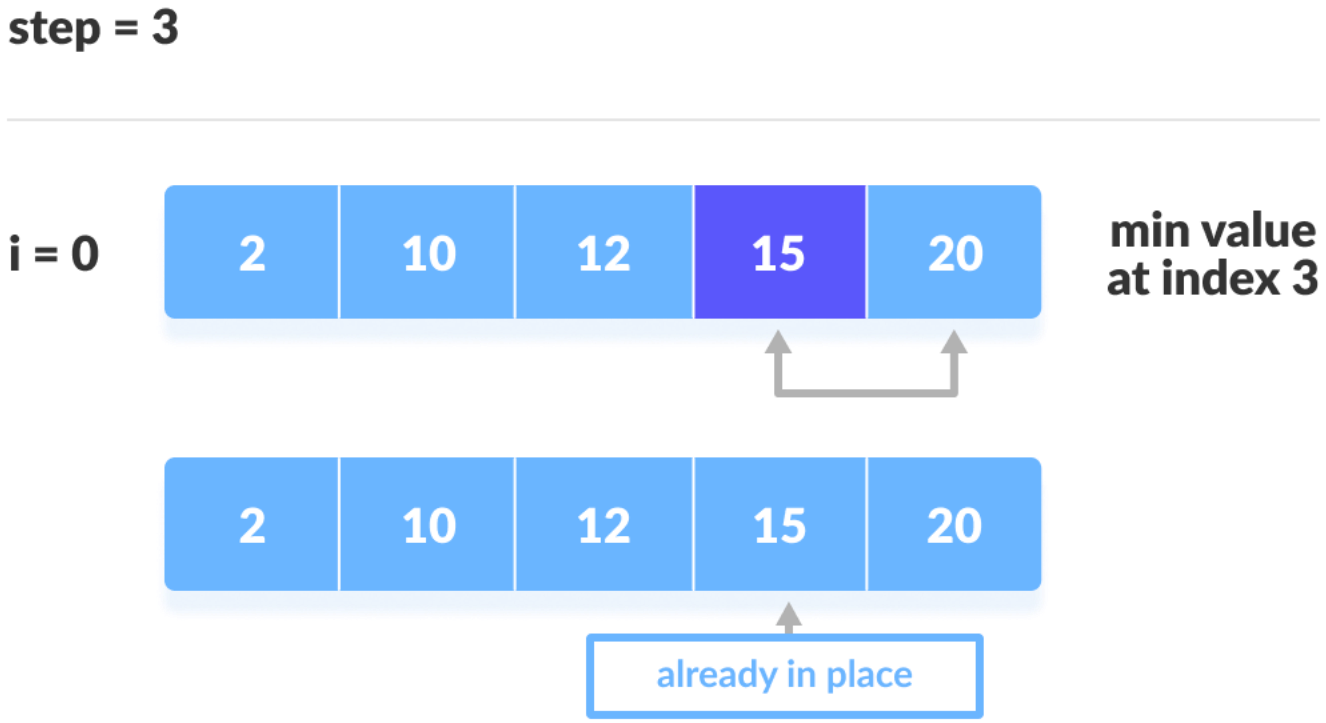


Working of Selection Sort (Cont.)

- The third iteration.

Working of Selection Sort (Cont.)

- The fourth iteration.



Selection Sort Applications

- The selection sort is used when:
  - a small list is to be sorted
  - cost of swapping does not matter
  - checking of all the elements is compulsory
  - cost of writing to a memory matters like in flash memory (number of writes/swaps is O(n) as compared to O(n2) of bubble sort)

Selection Sort Example in Java

SelectionSort.java

```java
// Selection sort in Java

import java.util.Arrays;

class SelectionSort {
  // Method to perform selection sort on the given array
  void selectionSort(int array[]) {
    int size = array.length;

    // Iterate through the array
    for (int step = 0; step < size - 1; step++) {
      int min_idx = step;

      // Find the index of the minimum element in the unsorted portion
      for (int i = step + 1; i < size; i++) {
        // To sort in descending order, change > to < in this line.
        // Select the minimum element in each loop.
        if (array[i] < array[min_idx]) {
          min_idx = i;
        }
      }

      // Swap the minimum element with the first element of the unsorted portion
      int temp = array[step];
      array[step] = array[min_idx];
      array[min_idx] = temp;
    }
  }

  // Driver code to test the selection sort algorithm
  public static void main(String args[]) {
    int[] data = { 20, 12, 10, 15, 2 };
    SelectionSort ss = new SelectionSort();
    ss.selectionSort(data);
    System.out.println("Sorted Array in Ascending Order: ");
    System.out.println(Arrays.toString(data));
  }
}
```

Insertion Sort Algorithm

- Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.
- Insertion sort works similarly as we sort cards in our hand in a card game.
- We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.
- A similar approach is used by insertion sort.
- Time Complexity: O(n^2) in the worst case, but O(n) in the best case (when the array is already sorted).
- Space Complexity: O(1)

Insertion Sort Step-by-Step Explanation

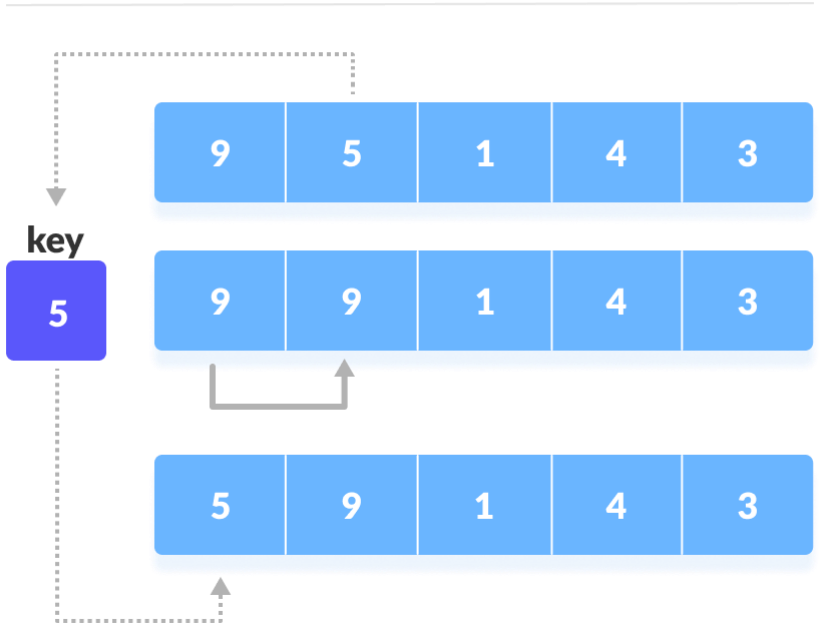
- Step-by-Step Explanation:
  - Iterate through the array, starting from the second element.
  - Compare each element with the elements on its left and insert it into the correct position.

Working of Insertion Sort

- Suppose we need to sort the following array.

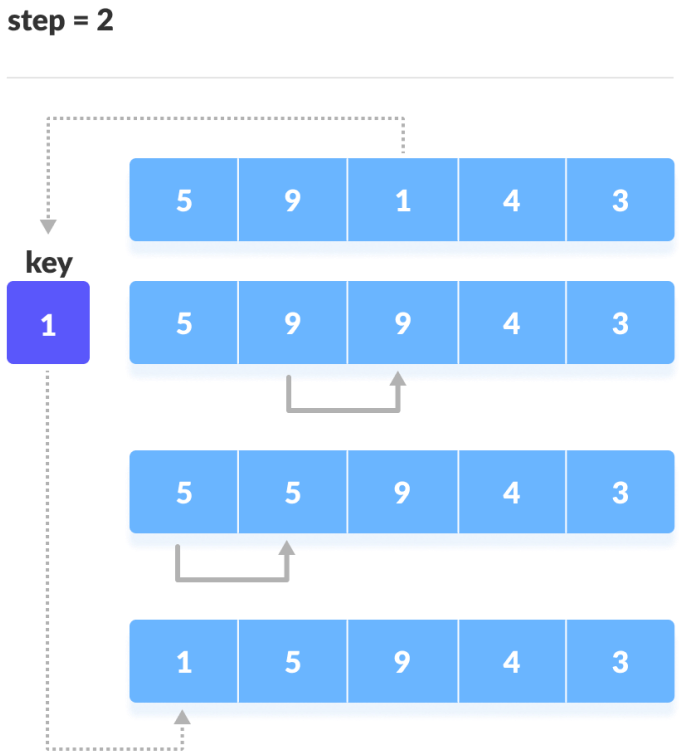
Working of Insertion Sort (Cont.)

- The first element in the array is assumed to be sorted. Take the second element and store it separately in key.
- Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.



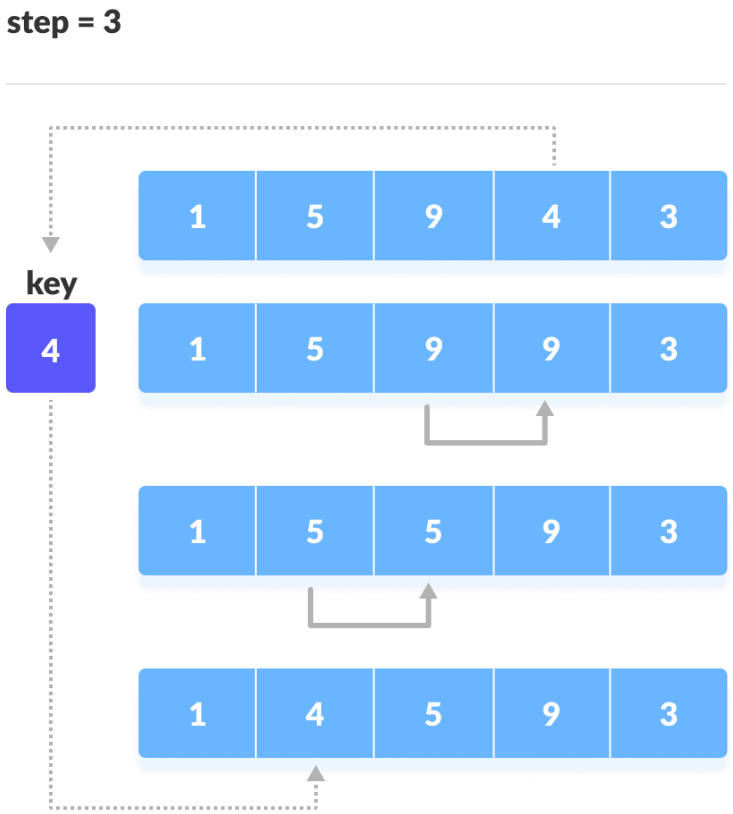
Working of Insertion Sort (Cont.)

- Now, the first two elements are sorted.
- Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.



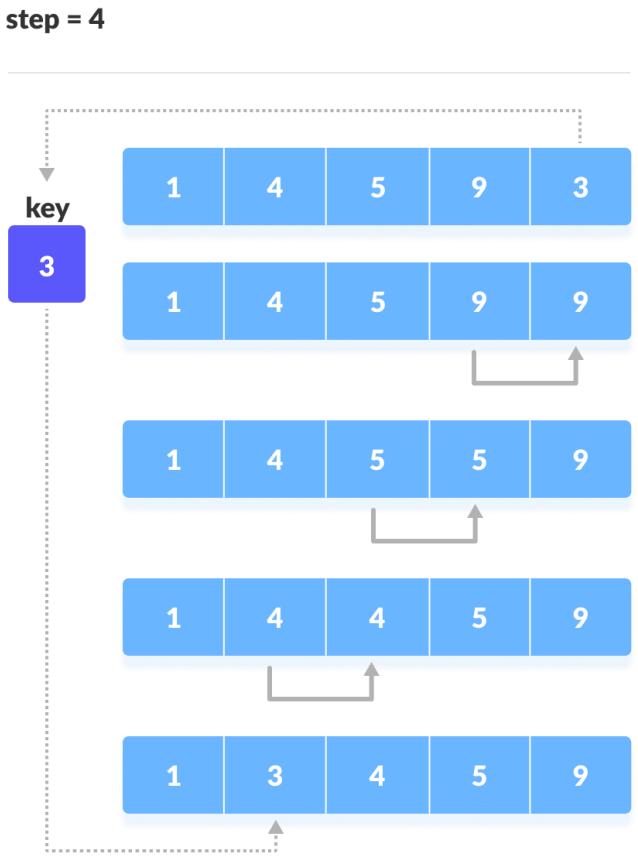
Working of Insertion Sort (Cont.)

- Similarly, place every unsorted element at its correct position.



Working of Insertion Sort (Cont.)

- Place 3 behind 1 and the array is sorted.



Insertion Sort Applications

- The insertion sort is used when:
  - the array is has a small number of elements
  - there are only a few elements left to be sorted

Insertion Sort Example in Java

InsertionSort.java

```java
// Insertion sort in Java

import java.util.Arrays;

class InsertionSort {
  // Method to perform insertion sort on the given array
  void insertionSort(int array[]) {
    int size = array.length;

    // Iterate through each element in the array
    for (int step = 1; step < size; step++) {
      int key = array[step];
      int j = step - 1;

      // Compare key with each element on the left of it until an element smaller than it is found.
      // For descending order, change key<array[j] to key>array[j].
      while (j >= 0 && key < array[j]) {
        array[j + 1] = array[j];
        --j;
      }

      // Place key after the element just smaller than it.
      array[j + 1] = key;
    }
  }

  // Driver code to test the insertion sort algorithm
  public static void main(String args[]) {
    int[] data = { 9, 5, 1, 4, 3 };
    InsertionSort is = new InsertionSort();
    is.insertionSort(data);
    System.out.println("Sorted Array in Ascending Order: ");
    System.out.println(Arrays.toString(data));
  }
}

```