# CSCI-UA-102-011-Spring-2025

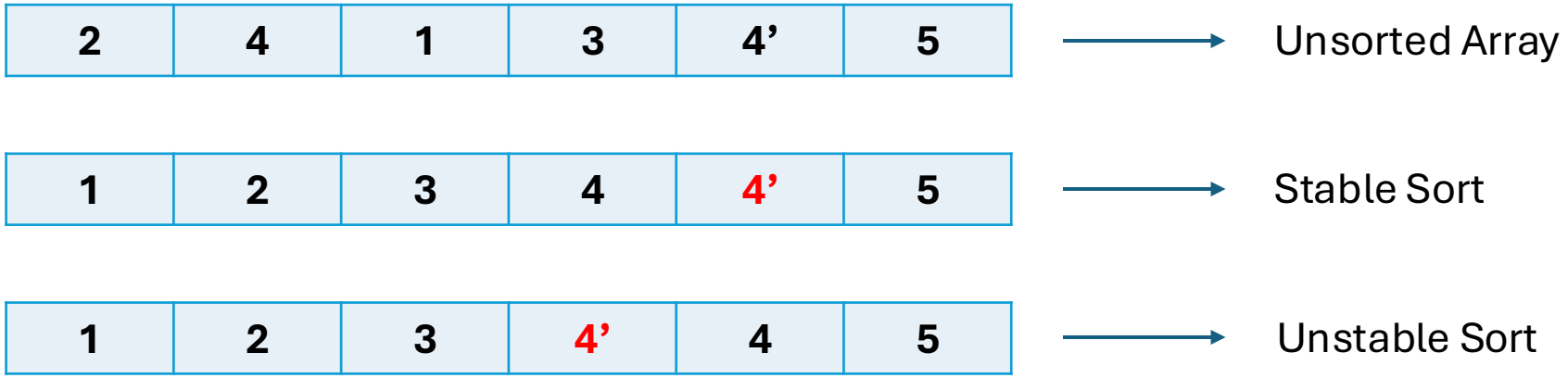
Recitation - 11

- Rasmika Billa
(03/11/2025)

# Agenda

- Stability in Sorting Algos
- Q12.6
- Q12.7
- Q12.26

# Stability in Sorting Algorithms




A **stable sorting algorithm** maintains the **relative order** of records with **equal keys**. This means, if two elements a and b have the same key and a comes before b in the input!

Making Bubble Sort Algorithm Stable →

```
public static void bubbleSort(int[] A) {
    int n = A.length;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n - i - 1; j++) {
          if (A[j] > A[j + 1]) {.        → Only swap if strictly greater
          int temp = A[j];
          A[j] = A[j + 1];
          A[j + 1] = temp;
        }
    }
}
```

- A **straggling** algorithm is the *opposite of stable* — if two equal-key entries appear as a before b in the input, then a must appear *after* b in the output.
- To make **Merge Sort straggling**, you can **modify the merge step** like this:

```
public static Entry[] mergeStraggling(Entry[] left, Entry[] right) {
   int i = 0, j = 0, k = 0;
   Entry[] result = new Entry[left.length + right.length];

   while (i < left.length && j < right.length) {
      if (left[i].key < right[j].key) {
         result[k++] = left[i++];
      } else if (left[i].key > right[j].key) {
         result[k++] = right[j++];
      } else {
         result[k++] = right[j++];      → Equal keys – bias towards right to make it straggling
      }
   }
   while (i < left.length) result[k++] = left[i++];
   while (j < right.length) result[k++] = right[j++];

   return result;
}
```

12.7 Given two sorted lists A and B, find the sorted union A ∪ B without duplicates in **O(n)** time:

```
public static List<Integer> sortedUnion(int[] A, int[] B) {
    int i = 0, j = 0;
    List<Integer> result = new ArrayList<>();
    while (i < A.length && j < B.length) {
      if (A[i] < B[j]) {
        result.add(A[i]);
        i++;
      } else if (A[i] > B[j]) {
        result.add(B[j]);
        j++;
      } else {
        result.add(A[i]);      → A[i] == B[j]
        i++;
        j++;
      }
    }
    while (i < A.length) {     → Add remaining elements
      result.add(A[i++]);
    }
    while (j < B.length) {
      result.add(B[j++]);
    }
    return result;
}
```

12.26

```
public class RemoveDuplicates {
   public static List<Integer> removeDuplicates(int[] A) {
      Set<Integer> seen = new HashSet<>();
      List<Integer> result = new ArrayList<>();

      for (int x : A) {
         if (!seen.contains(x)) {
            seen.add(x);
            result.add(x);
         }
      }

      return result;
   }
   public static void main(String[] args) {
      int[] A = {4, 2, 4, 5, 2, 3, 1, 5};
      List<Integer> unique = removeDuplicates(A);

      for (int num : unique) {
         System.out.print(num + " ");
      }
   }
}
```



Use a **hash set** to track seen elements and output only unique ones!