# Homework #2 Solution, CpSc 2120, April 12, Fall 2022
# Total 30 points

**Question 1 (6 + 4 = 10 points)**

(a) Hannah gives clues about her six-digit secret number:
   a. Clue 1: It is the same number if you read it from right to left.
   b. Clue 2: The number is a multiple of 9.
   c. Clue 3: Cross off the first and last digits. The only prime factor of the remaining four-digit number is 11.

   What is Hannah's six-digit number?

(b) The only way that 10 can be written as the sum of 4 different counting numbers is 1 + 2 + 3 + 4. In how many different ways can 15 be written as the sum of 4 different counting numbers? Enumerate those.

Answer:

- **Strategy:** Consider the clues one at a time, starting with the most restrictive. **Clue 3**: The only prime factor of the 4-digit number is 11, so the number = 11×11 or 11×11×11 or 11×11×11×11, etc. Of these, only 11×11×11= 1331 has 4 digits, so the middle 4 digits are 1331. **Clue 1**: The number reads the same right to left, so the first and last digits are the same. Call the number A1331A. **Clue 2:** The number is a multiple of 9, so the sum of its digits is a multiple of 9. A + 1 + 3 + 3 + 1 + A = A + 8 + A must equal 9 or 18. No digit A satisfies A + 8 + A = 9, but if A + 8 + A = 18, A = 5. Hannah's number is 513315.
- List the ways four different numbers can add to 15. Starting with the largest number reduces the number of trials necessary. Because 3 + 2 + 1 = 6 is the least possible sum for 3 of the numbers, the greatest can't exceed 15 – 6 = 9.
  - 15 = 9 + 6 = 9 + 3 + 2 + 1
  - 15 = 8 + 7 = 8 + 4 + 2 + 1
  - 15 = 7 + 8 = 7 + 5 + 2 + 1 or 7 + 4 + 3 + 1
  - 15 = 6 + 9 = 6 + 5 + 3 + 1 or 6 + 4 + 3 + 2

  Thus, 15 can be written as the sum of four different counting numbers is 6

**Question 2 (8 + 12 = 20 points)**

**Problem:** Given an array A[], if i < j and A[i] > A[j], then the pair (i,j) is called a reversal of the array A. [Example: Input: arr[] = {8, 4, 2, 1}, Output: 6; Why? Because, the given array has 6 reversals: (8, 4), (4, 2), (8, 2), (8, 1), (4, 1), (2, 1)] Provide a detailed algorithm pseudocode (and its complexity analysis) to compute the total number of reversal pairs in a given arbitrary array of size n. Then, provide an efficient algorithm pseudocode (and its complexity analysis) [**8 points for a simple-minded algorithm; the other 12 points for the most efficient algorithm**]

**Answer:** [A reversal pair is also known as an inversion pair]

- **Approach:** Traverse through the array and for every index find the number of smaller elements on its right side of the array. This can be done using a nested loop. Sum up the counts for all indexes in the array and print the sum.
- **Algorithm :**

1. Traverse through the array from start to end
2. For every element find the count of elements smaller than the current number up to that index using another loop.
3. Sum up the count of inversion for every index.
4. Print the count of inversions.

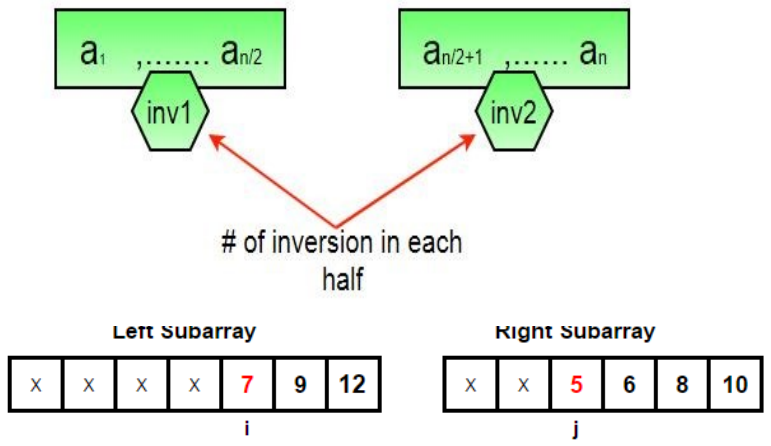
**Complexity Analysis:**

- **Time Complexity:** $O(n^2)$, Two nested loops are needed to traverse the array from start to end so the Time complexity is $O(n^2)$
- **Space** Complexity**:** O (1), No extra space is required.

**Efficient Approach:** Think of merge-sort of an array of integers.
Suppose the number of reversals in the left half and right half of the array (let be inv1 and inv2), what kinds of reversals are not accounted for in Inv1 + Inv2? The answer is – the reversals that need to be counted during the merge step. Therefore, to get a number of reversals, that needs to be added a number of reversals in the left subarray, right subarray, and merge ().

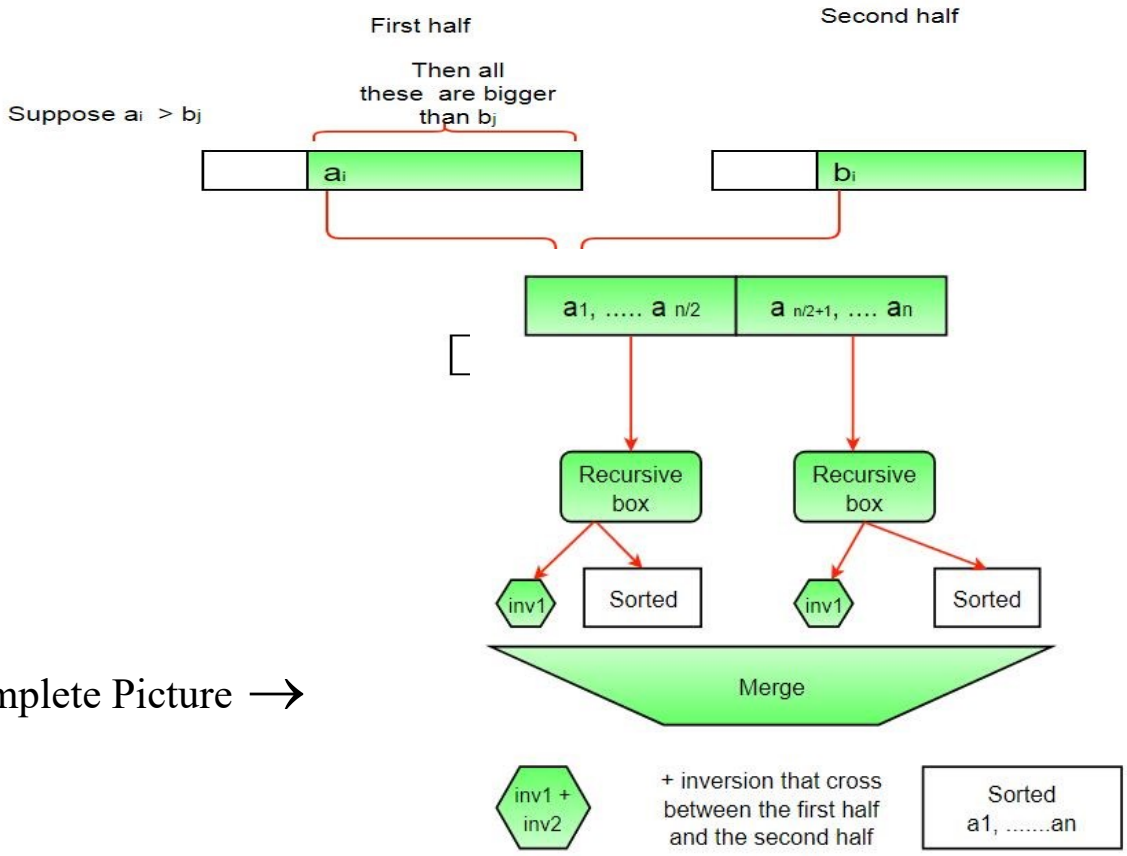
How to get the number of reversals in merge()?

- In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if a[i] is greater than a[j], then there are (mid – i) inversions. Since left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j]




**As 7 > 5, (7, 5) forms an inversion pair.**

Also as left subarray is sorted, it is obvious that elements 9 and 12 will also form inversion with element 5. i.e (9, 5) and (12, 5). So we can say that for element 5, total number of inversions are 3 which is exactly equal to number of elements left in the left subarray.




omplete Picture →

## Algorithm:

1. The idea is similar to merge sort, divide the array into two equal or almost equal halves in each step until the base case is reached.
2. Create a function merge that counts the number of inversions when two halves of the array are merged, create two indices i and j, i is the index for first half and j is an index of the second half. if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j].
3. Create a recursive function to divide the array into halves and find the answer by summing the number of inversions is the first half, number of inversion in the second half and the number of inversions by merging the two.
4. The base case of recursion is when there is only one element in the given half.
5. Print the answer

**Complexity Analysis:**

1. **Time Complexity:** O(n log n), The algorithm used is divide and conquer, So in each level one full array traversal is needed and there are log n levels so the time complexity is O(n log n).

**Space Complexity**: O (n), Temporary array of O(n) size.