Lab 3 Inversion count Analysis - Algorithms

Alberto Nicolai Romero Martinez

October 8, 2018

1 Minimum number of inversions

The minimum number of inversions in any array of distinct elements is 0. It is a increasing order sorted array, where:

$$\forall\ x[i],\ x[j]\ \ |\ \ i < j : x[i] < x[j] \tag{1}$$

2 Maximum number of inversions

The maximum number of inversions in any array of distinct elements is $\frac{(n-1)(n-2)}{2}$. It is a decreasing order sorted array, where:

$$\forall\ x[i], x[j]\ \ |\ \ i < j : x[i] > x[j] \tag{2}$$

The result is the sum of the number of inversions of an index involving every previous index in the array. This is:

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n-2)}{2} \tag{3}$$

3 Complexity (worst case number of comparisons) of the brute force inversion counting

The brute force algorithm used to determine the number of inversions in an array consists in iterating over the elements of the array and compare them with every previous element, and counting the amount of them larger than the element.

Brute force - Array inversion count

$count = 0$
for $j = 1$ **to** $A.length - 1$ **do**
  for $i = 0$ **to** $j - 1$ **do**
    if $A[i] > A[j]$ **then**

count+ = 1
end if
end for
end for
return *count*

In any case, the algorithm performs $\frac{(n-1)(n-2)}{2}$ comparisons, because for the element $A[i]$ it evaluates the i previous elements, for $1 < i < n-1$.

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n-2)}{2} \tag{4}$$

4 Complexity (worst case number of comparisons) of the divide and conquer inversion counting

The divide and conquer algorithm used to determine the number of inversions in an array consists in sorting the array using the merge sort method, which divides the array in 2 partitions, that are sorted recursively using the same dividing method (until one partition becames 1-length) and then merges them again comparing the first elements of each partition. In order to find the number of inversions, when one element of the right side of a merge operation is selected to enter the merged array, we then know that it is lower than every single remaining element in the left arrray. So it have an inversion with every of these elements, and we count the sum of the size of the right array in every merge operation involving 1 element of the left array.

Divide and conquer (Merge-Sort) - Array inversion count

count = 0
mergedList = []
while $left.length > 0$ **and** $right.length > 0$ **do**
if left[0]≤ $right$[0] **then**
mergedList.append(left.removeFirst())
else
mergedList.append(right.removeFirst())
count+ = **left.length**

We need to sort the entire array before we know the total number

of comparisons. Thus, the number of comparisons of this problem is given by the worst case number of comparisons of **Merge Sort** which is in $O(n(log(n)))$.

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n-2)}{2} \tag{5}$$