Jade Yu Cheng
ICS 311
Homework 2
Sep 2, 2008

Question for lecture 3

Problem 2-4 on p. 39

Inversions

a. List the five inversions of the array {2, 3, 8, 6, 1}

Answer: {2, 1}, {3, 1}, {8, 1}, {6, 1}, {8, 6}

b. What array with elements from the set {1, 2,..., n} has the most inversions? How many does it have?

Answer: An array has the most inversions when the elements are sorted and in an order from the greatest to the smallest. The number of inversions is the number of combinations of any of the two elements, so it's:

$$C_n^2 = \frac{n!}{(n-2)! \cdot 2!}$$

c. What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify the answer.

Answer: The more inversions, the greater the running time of the insertion sort algorithm is. The number of inversions determines the actual running time, which varies from the best case $\Theta(n)$ to the worst case $\Theta(n^2)$.

For the former case, the elements are already sorted. Each element at index *j* is compared to the element at index *j – 1* and stay where it is. The number of operations involved in this process is:

$$1_1 + 1_2 + 1_3 + \cdots + 1_n = \Theta(n)$$

For the latter case, the elements are reversely sorted. Each element at index *j* is compared to all elements at index *1* to *j – 1* of the sorted section of the array and is inserted at the beginning of the list. The number of operations involved in this process is:

$$1 + 2 + 3 + \cdots + n = \Theta(n^2)$$

d. Give an algorithm that determines the number of inversions in any permutation on *n* elements in $\Theta(n \lg n)$ worst-case time.

Answer:

```
int inversions = 0;

int[] mergeSort(int[] array) {
        if (array.length <= 1)
                return array;

        int middle = array.length / 2;
        int[] left = new int[middle];
        for (int i = 0; i < middle; i++)
                left[i] = array[i];

        int[] right = new int[array.length - middle];
        for (int i = middle; i < array.length; i++)
                right[i - middle] = array[i];

        left = mergeSort(left);
        right = mergeSort(right);
        return merge(left, right);
}

int[] merge(int[] left, int[] right) {
        int[] array = new int[left.length + right.length];

        int leftIndex = 0;
        int rightIndex = 0;
        int i = 0;

        while (leftIndex < left.length && rightIndex < right.length) {
                if (left[leftIndex] <= right[rightIndex]) {
                        array[i++] = left[leftIndex++];
                } else {
                        inversions += left.length - leftIndex;
                        array[i++] = right[rightIndex++];
                }
        }

        while (leftIndex < left.length) {
                array[i++] = left[leftIndex++];
        }

        while (rightIndex < right.length) {
                array[i++] = right[rightIndex++];
        }

        return array;
}
```