SCHOOL OF COMPUTING SCIENCE

SORTING

Budeanu, Daniel - Elliott, Jon - Hussain, Najam - Miron, Dragos - Prelic, Lukasz

TABLE OF CONTENTS

INTRODUCTION TO SORTING

Definitions

Sorting is any process of arranging items according to a certain sequence or in different sets. In computer science, this is a widely researched subject because of the need to speed up the operation of finding data in a large set.

A sorting algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order.

The time complexity of an algorithm measures the amount of time taken by an algorithm to run. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Thus the amount of time taken and the number of elementary operations performed by the algorithm differ by at most a constant factor. Since an algorithm's performance time may vary with different inputs of the same size, one commonly uses the worst-case time complexity of an algorithm, denoted as T(n), which is defined as the maximum amount of time taken on any input of size n.

The space complexity measures the amount of memory cells used by an algorithm in respect with the input size. Time complexity is composed of the space to store the data to be processed and the Auxiliary space which is the temporary space used by the algorithm.

QUICKSORT

Introduction

Quicksort is a divide and conquer algorithm which first divides a large list into two smaller sub-lists: the low elements and the high elements. Quicksort can then recursively sort the sub-lists. The quicksort algorithm was developed in 1960 by Tony Hoare while in the Soviet Union, as a visiting student at Moscow State University. At that time, Hoare worked in a project on machine translation for the National Physical Laboratory. He developed the algorithm in order to sort the words to be translated, to make them more easily matched to an already-sorted Russian-to-English dictionary that was stored on magnetic tape.

Algorithm explained

Pick an element, called a pivot, from the list. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation. Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values. The base cases of the recursion are lists of size zero or one, which never need to be sorted.

Example

Complexity

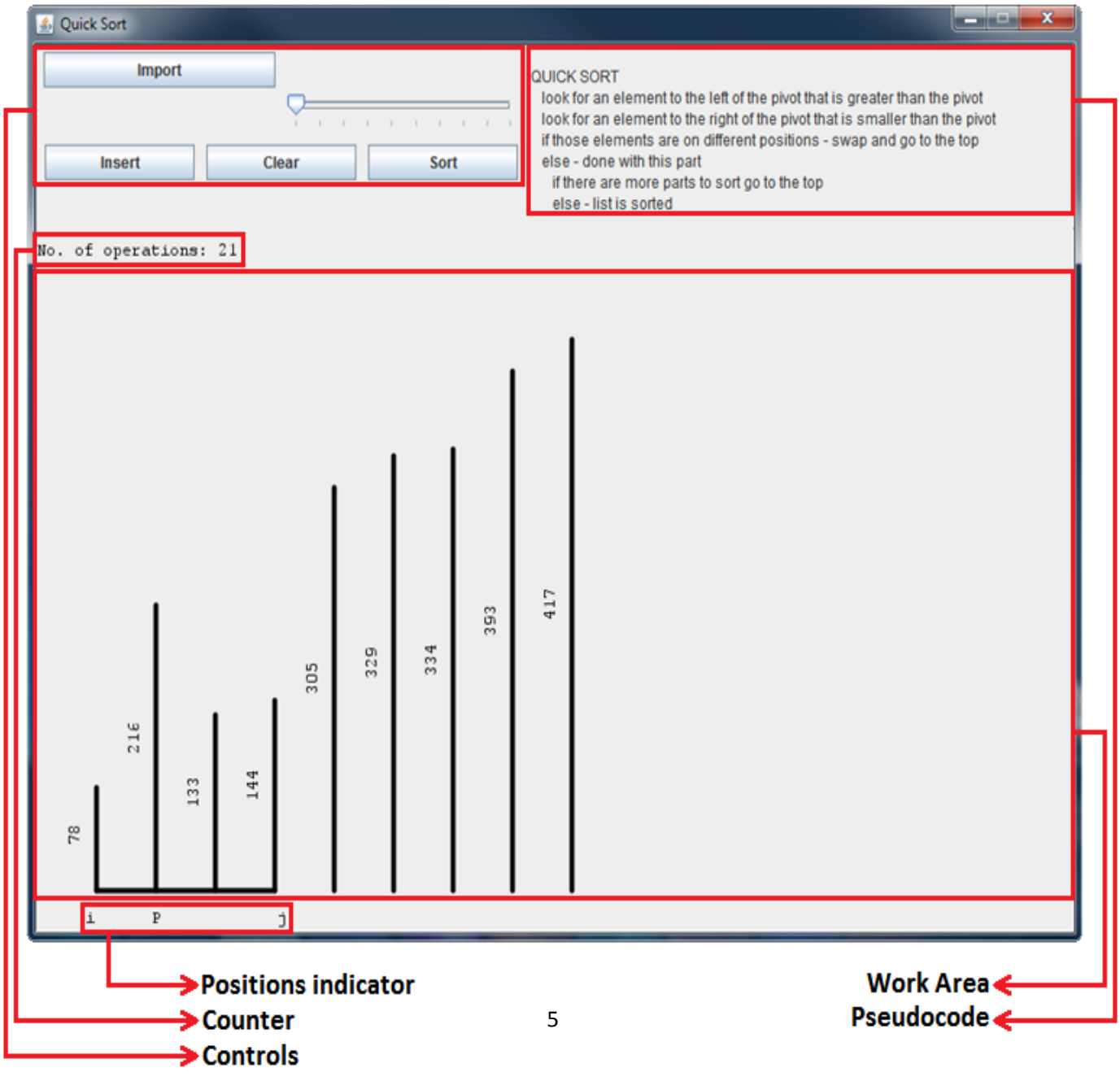
The time complexity of quicksort in the worst case scenario is $O(n^2)$ and this occurs when the list is already sorted or if it contains all elements equal to each other. The space complexity of the worst case is O(n) for the auxiliary space.

Improvements

Algorithm version	Improvement
In place	Reduces the space complexity to O(log n)
Fat partition	The case of repeated elements is no longer a problem
Selection algorithm for selecting the best pivot	Worst case time complexity reduced to O(n log n). However, in practice, this implementation is slower on average.

How to use the demonstrator

Run QuickSort.jar and the following screen will appear.



To insert an element to the list click the Insert button from the Controls area, then click once in the Work Area. As you move the mouse, you can decide the size of the element. When you decided on a size, click again and the element will be added to the list. Elements can also be added to the list using the Import button. The imported files need to have on separate lines numbers representing the height of the elements. The minimum size is 20 and the maximum is 500 and if numbers are out of this range they will be modified.

To clear the list, simply click the Clear button and to sort it, click Sort from the Controls area. Notice that the list has to have more than one element in order to be sorted.

The algorithm animation speed can be adjusted using the slider in the Controls area.

While the algorithms is running, the counter will keep track of the number of operations needed to sort the list and lines from the Pseudocode area will be highlighted in order to indicate which step of the algorithm is being executed.

The positions indicator will have 3 labels:

- i - looks for an element larger than the pivot in the sub-list to the left of the pivot.
- P - indicates the pivot position
- J - looks for an element smaller than the pivot in the sub-list to the right of the pivot.

The list that the algorithm is performing the partitioning will be underlined.

RADIX SORT

Introduction

Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. A positional notation is required, but because integers can represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers. Radix sort dates back as far as 1887 to the work of Herman Hollerith on tabulating machines. Two classifications of radix sorts are least significant digit (LSD) radix sorts and most significant digit (MSD) radix sorts.

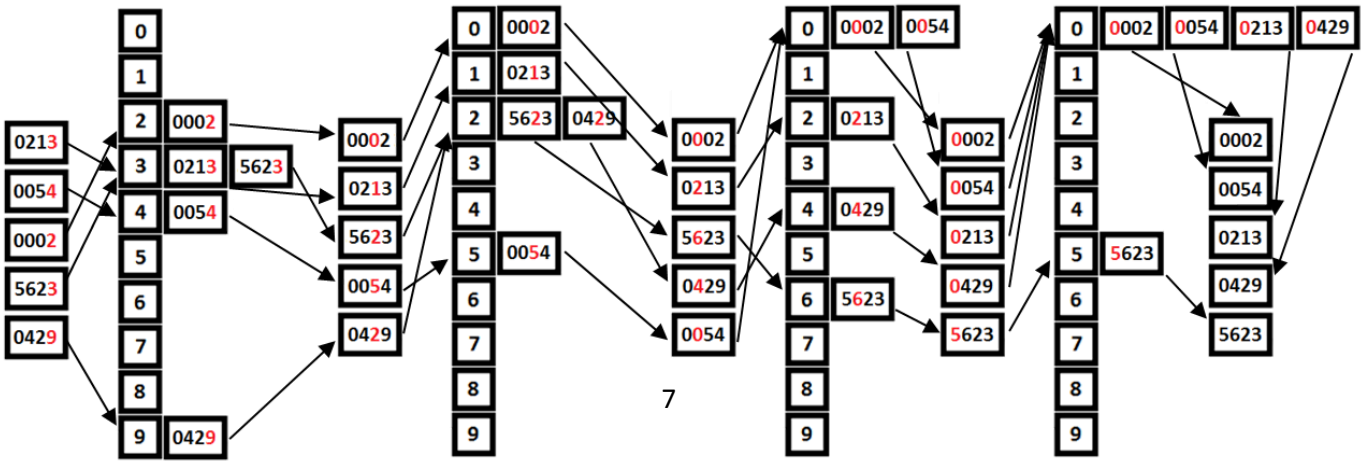
LSD radix sorts process the integer representations starting from the least digit and move towards the most significant digit. MSD radix sorts work the other way around. LSD radix sorts typically use the following sorting order: short keys come before longer keys, and keys of the same length are sorted lexicographically. This coincides with the normal order of integer representations, such as the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

MSD radix sorts use lexicographic order, which is suitable for sorting strings, such as words, or fixed-length integer representations. A sequence such as "b, c, d, e, f, g, h, i, j, ba" would be lexicographically sorted as "b, ba, c, d, e, f, g, h, i, j". If lexicographic ordering is used to sort variable-length integer representations, then the representations of the numbers from 1 to 10 would be output as 1, 10, 2, 3, 4, 5, 6, 7, 8, 9, as if the shorter keys were left-justified and padded on the right with blank characters to make the shorter keys as long as the longest key for the purpose of determining sorted order.

Algorithm explained

Considering a list of integers, LSD radix sort will first zero pad the integers such that all will have the same length. ({213, 54, 2, 5623, 429} --> {0213, 0054, 0002, 5623, 0429}) After, the least significant digit is selected (in this case the 4th). Integers are placed in “buckets” according to their selected digit. 0213 goes to bucket 3, 0054 goes to bucket 4 and so on. After this step, the new order is {0002, 0213, 5623, 0054, 0429}. Recursively, the same step is applied for each digit.

Example

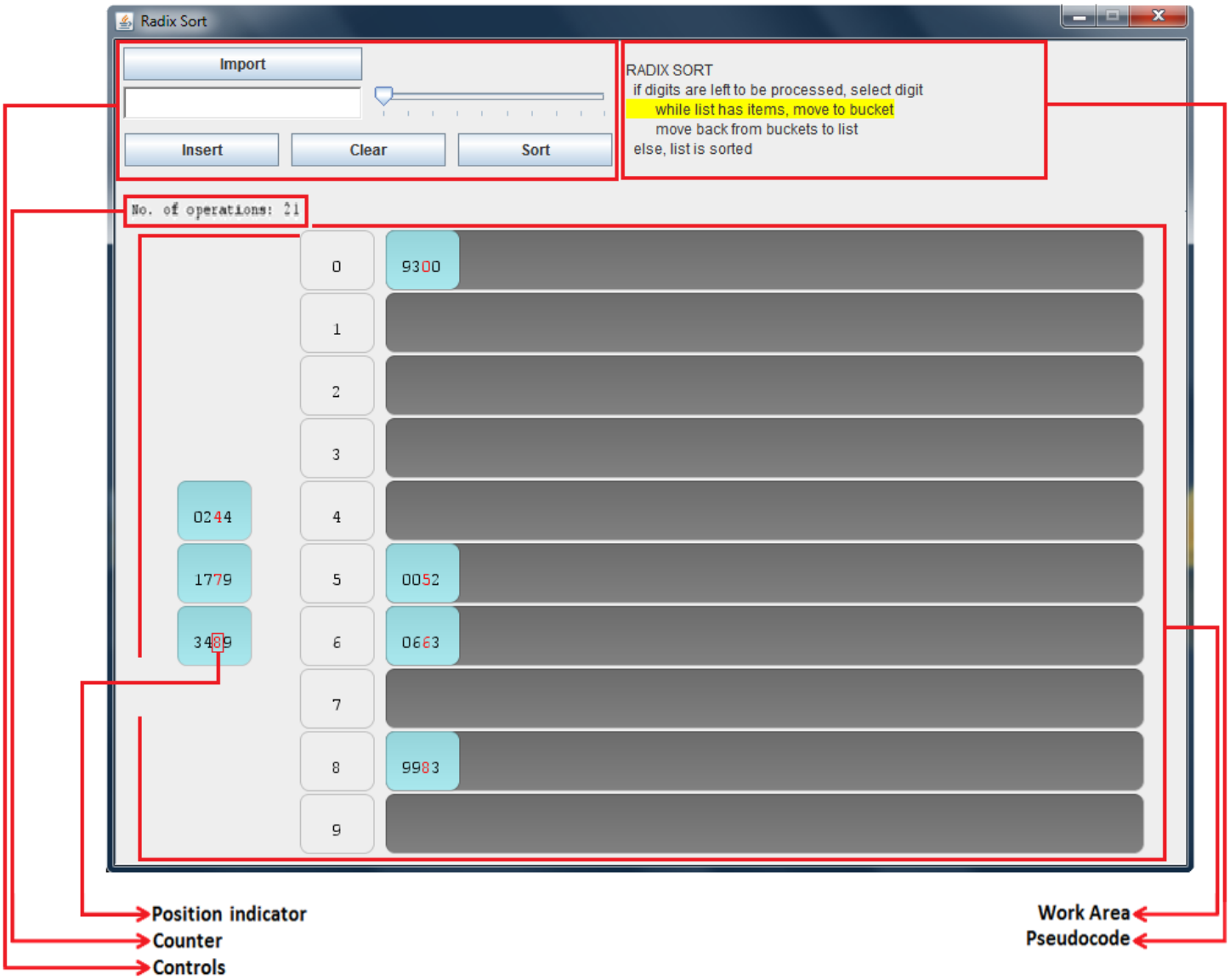


Complexity

The time complexity of radix sort is constant and is given by O(m*n), where m is the maximum length of the elements and n is the number of elements. The auxiliary space required to sort a list with n elements is equal to the space needed to implement the buckets.

How to use the demonstrator

Run RadixSort.jar and the following screen will appear:



To insert an element to the list, input in the text box in the Controls area and click the Insert button or hit Enter. Elements can also be added to the list using the Import button. The imported files need to have on separate lines numbers representing the height of the elements. All elements manually entered or imported must be numbers between 0 and 9999.

To clear the list, simply click the Clear button and to sort it, click Sort from the Controls area. Notice that the list has to have more than one element in order to be sorted.

The algorithm animation speed can be adjusted using the slider in the Controls area.

While the algorithms is running, the counter will keep track of the number of operations needed to sort the list and lines from the Pseudocode area will be highlighted in order to indicate which step of the algorithm is being executed.

The position indicator (digit is coloured red) will show according to which digit is the element moved to a bucket.

GNOME SORT

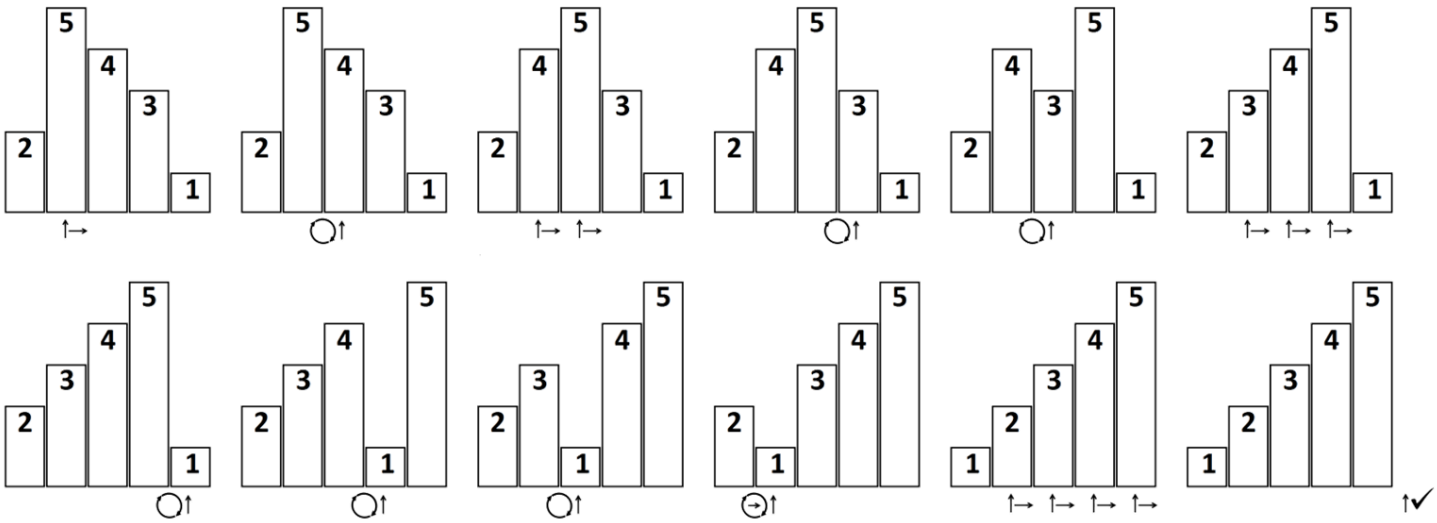
Introduction

Gnome sort (Stupid sort), originally proposed by Dr. Hamid Sarbazi-Azad (Professor of Computer Engineering at Sharif University of Technology) in 2000 and called Stupid sort, and then later on described by Dick Grune and named "Gnome sort", is a sorting algorithm which is similar to insertion sort, except that moving an element to its proper place is accomplished by a series of swaps, as in bubble sort. It is conceptually simple, requiring no nested loops.

Algorithm explained

Considering a list of integers, and the initial position pointing to the second element, a comparison is made between the current element and the previous one. In case of the current element being larger than the predecessor, the current position is incremented. This is done until the current element passes the last element of the list. Otherwise, if the current element is smaller than the predecessor, they are swapped and the current position is decreased, but it cannot be decreased so that it points to the first element.

Example




current element greater than predecessor - increment position

current elements smaller than predecessor - swap and decrease position


current elements smaller than predecessor - swap

Complexity

The worst case time complexity of radix sort is $O(n^2)$, but tends towards O(n) if the list is initially almost sorted. The average runtime is $O(n^2)$. The auxiliary space needed for radix sort is O(1).

Improvements

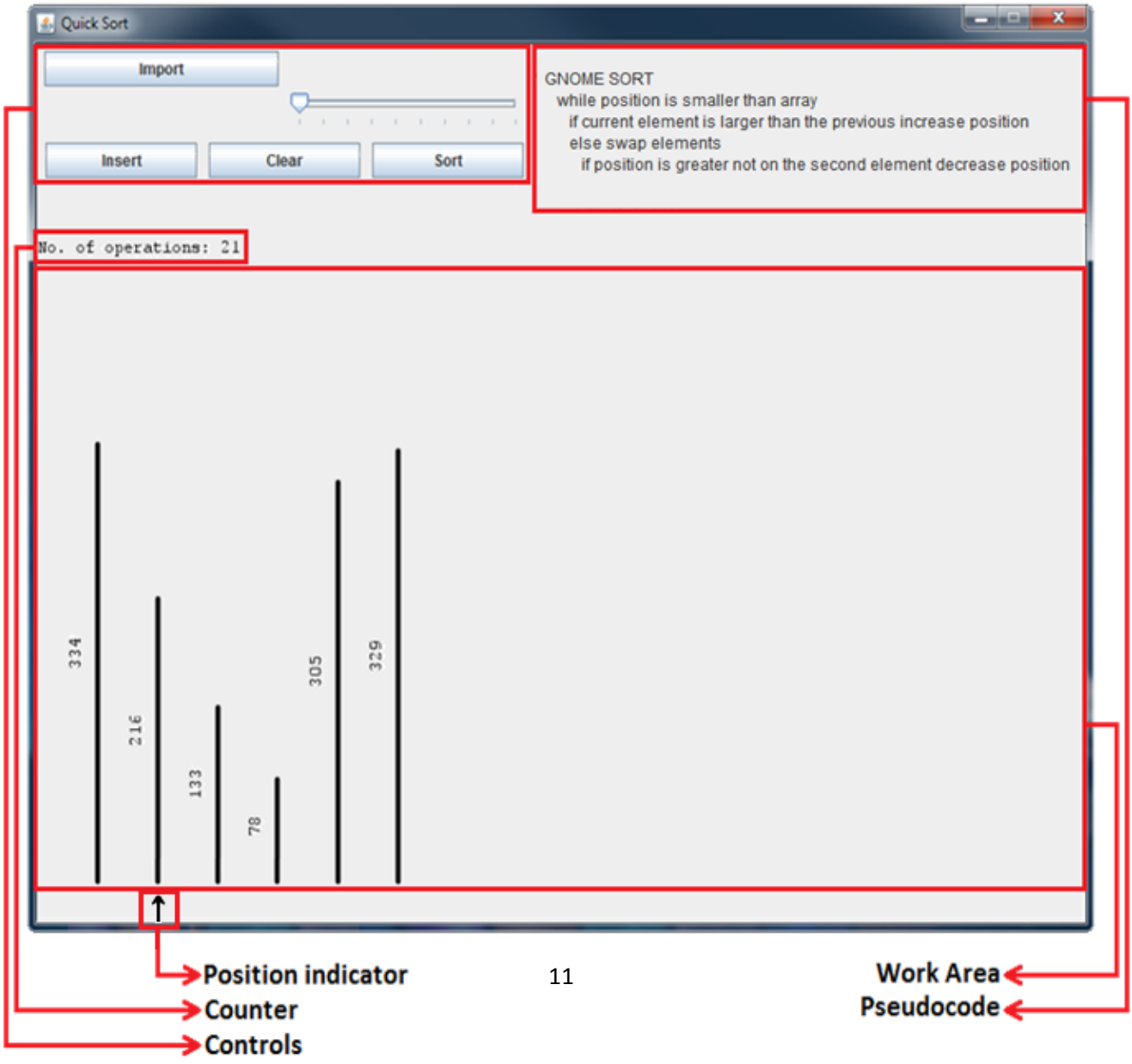
Consider this example:

List = {3, 4, 5, 6, 7, 8, 2, 1}. Radix sort will search until in finds element 2 less than 8 and then they will be swapped and then 2 will be swapped with 7 and so on until the list is {2, 3, 4, 5, 6, 7, 8, 1}. Now the search for an unordered element will begin from the second element and will go through the whole list to find element 1.

The improvement of radix sort consists in having a variable that saves the position of the previous unordered element and after that element is being swapped as many times until it is ordered, the search for the next unordered element can resume from that position.

How to use the demonstrator

Run GnomeSort.jar and the following screen will appear:



To insert an element to the list click the Insert button from the Controls area, then click once in the Work Area. As you move the mouse, you can decide the size of the element. When you decided on a size, click again and the element will be added to the list. Elements can also be added to the list using the Import button. The imported files need to have on separate lines numbers representing the height of the elements. The minimum size is 20 and the maximum is 500 and if numbers are out of this range they will be modified.

To clear the list, simply click the Clear button and to sort it, click Sort from the Controls area. Notice that the list has to have more than one element in order to be sorted.

The algorithm animation speed can be adjusted using the slider in the Controls area.

While the algorithms is running, the counter will keep track of the number of operations needed to sort the list and lines from the Pseudocode area will be highlighted in order to indicate which step of the algorithm is being executed.

The position indicator will show the current element.

REFERENCES

1. http://en.wikipedia.org/wiki/Quicksort
2. http://en.wikipedia.org/wiki/Radix_sort
3. http://en.wikipedia.org/wiki/Gnome_sort