Report

The assignment was dived in three parts. The first part asked us to write code in the main class so we could test the sorting algorithm that were developed in the sorting class from the program. In the main class I implemented two methods in order to print the sorted list and to unsort the.

Part two instead asked to use the same sorting algorithm, but to sort them by telephone number instead by last name how we did in the previous part. This was a very simple task to accomplish. I modified the class where the comparison code was implemented, so that I would compare the telephone numbers. Actually I copy and paste the method and renamed the method. So I had both copies of them. The other task that was required in part two was to change the quick sort method so it would use the first element of the array as a partition element. That was a simple task to perform. I just

Initialized the partition variable with the array[min], with min being the minimum value passed in the method.

The third part was a little bit challenging. It asked to perform a bubble sort and a quick-sort on the same random numbers stored in an array of size 100. I achieved the task by copying the same random numbers in two distinct arrays. This was done because I could not reference the two arrays.

Part two and three asked to calculate the process time used by the computer to sort using bubble and quick-sort on the same set of random numbers in an array that varied it size by 10^n and to calculate the time of computing of the sorts used previously in an array already sorted. The results are illustrated in the table.

|  |  |  |
| --- | --- | --- |
| Same unsorted list | The quick sort time ns | The bubble sort time ns |
| Unsorted list Size = 100 | 7148624 | 9613401 |
| Unsorted list Size = 1000 | 5016759 | 37735462 |
| Unsorted list Size = 10000 | 5161413 | 1375480772 |

The data shows the time in ns for the same unsorted list having the same data. We notice that while the size increases the bubble is very inefficient while the quick sort is very efficient.

|  |  |  |
| --- | --- | --- |
| **with the initSorted** | The quick sort time ns | The bubble sort time ns |
| Size = 100000 | 42725546 | 56386723 |

This is the data with the initsorted method. This method returns already a sorted array and it point to the second array in which the two sorts are performed. As in the previous table here we notice still the efficiency of quick sort over bubble-sort. The data is not very surprising because it is line with what we have studied in class.