Mhammed Alhayek

RUID: 156-00-6919

Programming Methodology Homework 3

Question 1: code submitted on Sakai.

Question 2: also in same file for code submitted on Sakai.

Question 3:

Data:

|  |  |  |  |
| --- | --- | --- | --- |
| Unsorted Inputs | | | |
| Number of Elements | Selection Sort | Insertion Sort | Bubble Sort |
| 4 | 12 | 7 | 7 |
| 16 | 150 | 160 | 188 |
| 64 | 2142 | 1667 | 2631 |
| 256 | 33150 | 33126 | 48338 |
| 1024 | 525822 | 505484 | 775857 |
| 4096 | 8394750 | 8364499 | 12564138 |
| 65536 | 2147581950 | 2159176561 | 3226992031 |
| 1048576 | Computer Can't Handle These Large Inputs | | |
| 16777216 |

Plots:

Discussion:

All the plots were plotted on a graph with a logarithmic horizontal and vertical axis with a base of 2. The fact that all three graphs are linear shows that the correlation between run time and number of inputs n is roughly n^2. For the data set with only 4 elements, Bubble Sort and Insertion Sort were actually slightly faster than selection sort, however, as the data size increased, bubble sort was significantly slower than the other two algorithms. Insertion sort was consistently faster than the other two algorithms except for once the data set got very large, 216 elements to be exact. At that point, selection sort was actually slightly faster. For the last two data sets, which each had over 1 million elements, my computer was never able to complete sorting the arrays. This shows why using an algorithm with O(n2) time complexity is not a good idea for large values of n.

Question 4:

Data:

|  |  |  |  |
| --- | --- | --- | --- |
| Sorted Inputs | | | |
| Number of Elements | Selection Sort | Insertion Sort | Bubble Sort |
| 4 | 12 | 3 | 3 |
| 16 | 150 | 15 | 15 |
| 64 | 2142 | 63 | 63 |
| 256 | 33150 | 255 | 255 |
| 1024 | 525822 | 1023 | 1023 |
| 4096 | 8394750 | 4095 | 4095 |
| 65536 | 2147581950 | 65535 | 65535 |
| 1048576 | Computer Can't Handle These Large Inputs | | |
| 16777216 |

Plots:

Discussion:

For selection sort with a sorted input, the run times do not change, because even if the elements are already sorted, the algorithm still needs to compare each selected item with every other item in the array. For insertion sort, the run time dropped dramatically because now all the algorithm does is it checks to see if each unsorted element is less than the previous element and since it is not, it moves on to the next element. This causes insertion sort to just make n-1 checks before terminating when it has a sorted array as an input. For bubble sort, the run time dropped dramatically as well. This is because now when bubble sort makes its first pass through all the elements, it sees that nothing was swapped, and thus it knows the array is sorted and terminates. Because when you make the first pass you make n-1 comparisons, the run time for bubble sort with a sorted array is n-1. As you can see, for selection sort, the worst case and the best case complexity are the same, however, for bubble sort and insertion sort, they each have a best case scenario of O(n).