Activity 1. Time measurements for sorting algorithms

All the times are measured in milliseconds and the specifications of the computer are: RAM: 8 GB, CPU: AMD Ryzen 7 2700X.

**Insertion**

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *sorted(t)* | *inverse(t)* | *random(t)* |
| 1000 | 4 | 197 | 114 |
| 2000 | 2 | 439 | 265 |
| 4000 | 0 | 1954 | 982 |
| 8000 | 0 | 7873 | 3952 |
| 16000 | 0 | 31657 | 15666 |
| 32000 | 0 |  |  |
| 64000 | 1 |  |  |
| 128000 | 5 |  |  |
| 256000 | 14 |  |  |
| 512000 | 51 |  |  |
| 1024000 | 98 |  |  |
| 2048000 | 212 |  |  |

The time complexity of the sort method in this class is , thus, let us take a measurement from the inverse column to check whether the obtained results make sense or not.

t1 = 439 ms, n1 = 2000 and n2 = 4000; then, t2 = n2^2/n1^2 \* t1 = 1756 ms

The measurement on the table is a bit bigger but the obtained results are the expected as they match a quadratic tendency.

**Selection**

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *sorted(t)* | *inverse(t)* | *random(t)* |
| 1000 | 87 | 260 | 203 |
| 2000 | 327 | 975 | 750 |
| 4000 | 1239 | 3906 | 2970 |
| 8000 | 4930 | 15660 | 11772 |
| 16000 | 19739 | 62457 | 47034 |

The time complexity of the sort method in this class is , thus, let us take a measurement from the inverse column to check whether the obtained results make sense or not.

t1 = 975 ms, n1 = 2000 and n2 = 4000; then, t2 = n2^2/n1^2 \* t1 = 3900 ms

The measurement on the table is practically the same obtained, so we can be certain that the results are correct and make sense.

**Bubble**

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *sorted(t)* | *inverse(t)* | *random(t)* |
| 1000 | 104 | 447 | 924 |
| 2000 | 370 | 1131 | 3560 |
| 4000 | 1426 | 4442 | 15742 |
| 8000 | 5690 | 17707 | 66574 |
| 16000 | 22808 | 71194 | 273418 |

The time complexity of the sort method in this class is , thus, let us take a measurement from the inverse column to check whether the obtained results make sense or not.

t1 = 1131 ms, n1 = 2000 and n2 = 4000; then, t2 = n2^2/n1^2 \* t1 = 4524 ms

The measurement on the table is, again, practically the same obtained, so we can be certain that the results are correct and make sense.

**Quicksort**

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *sorted(t)* | *inverse(t)* | *random(t)* |
| 1000 | 4 | 4 | 11 |
| 2000 | 6 | 11 | 21 |
| 4000 | 4 | 21 | 27 |
| 8000 | 7 | 41 | 57 |
| 16000 | 13 | 21 | 115 |
| 32000 | 28 | 47 | 231 |
| 64000 | 65 | 93 | 490 |
| 128000 | 161 | 182 | 1010 |
| 256000 | 281 | 396 | 2132 |
| 512000 | 543 | 816 | 4312 |
| 1024000 | 1238 | 1682 | 8874 |
| 2048000 | 2362 | 3443 | 18525 |
| 4096000 | 5111 | 7261 | 39851 |
| 8192000 | 10078 | 14659 | 89520 |
| 16384000 | 22043 | 31218 | 218690 |

The time complexity of the sort method in this class is , thus, let us take a measurement from the inverse column to check whether the obtained results make sense or not.

t1 = 11 ms, n1 = 2000 and n2 = 4000; then, t2 = n2/n1 \* t1 = 21 ms

The measurement on the table is the same obtained, so we can be certain that the results are correct and make sense.

Activity 2. QuicksortFateful

The selected pivot is always the element located at the *left* index, the leftmost element; in most cases the first one. This choice of the pivot will work when the array is randomly or inversed sorted, and the worst scenario will be when the array is already sorted.