Activity 1. [BUBLE ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| Bubble | | | |
| n | tOrdered | tReverse | tRandom |
| 10000 | 315 | 1476 | 1030 |
| 20000 | 1217 | 5803 | 4071 |
| 40000 | 4814 | 23054 | 16195 |
| 80000 | 19601 | Oot | Oot |
| 160000 | Oot | Oot | Oot |

Theoretically, the complexity of this algorithm is O(n2), which fits perfectly with the results obtained, as we increase the size of the problem by a factor of 2, the time spent is the one of the previous calculation multiplied by 22, which is the size of the problem to the power of 2.

The difference between the columns is because of the number of operations we need to order the different arrays.

Activity 2. [SELECTION ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| Selection | | | |
| n | tOrdered | tReverse | tRandom |
| 10000 | 304 | 288 | 318 |
| 20000 | 1213 | 1124 | 1237 |
| 40000 | 4812 | 4506 | 4903 |
| 80000 | 19495 | 17895 | 19305 |
| 160000 | Oot | Oot | Oot |

Here we are in the same case as with the bubble algorithm, the complexity is O(n2) which is reflected in the times measured.

Activity 3. [INSERTION ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| Insertion | | | |
| n | tOrdered | tReverse | tRandom |
| 10000 | LoR | 298 | 151 |
| 20000 | LoR | 1156 | 587 |
| 40000 | LoR | 4646 | 2319 |
| 80000 | LoR | 18499 | 9323 |
| 160000 | LoR | Oot | 37076 |
| 320000 | LoR | Oot | Oot |
| 640000 | LoR | Oot | Oot |
| 1280000 | LoR | Oot | Oot |
| 2560000 | LoR | Oot | Oot |
| 5120000 | 92 | Oot | Oot |
| 10240000 | 181 | Oot | Oot |
| 20480000 | 360 | Oot | Oot |
| 40960000 | 726 | Oot | Oot |
| 81920000 | 1455 | Oot | Oot |

The theoretically complexity of the insertion algorithm is O(n2) for worst cases, and O(n) for best cases. This complexity agrees with the results obtained, as the best case is sorting an array already sorted, this times grows in a linear way because the algorithm just traverse the array trying to find a number disordered. In the other two cases, which are part of the “worst” cases, the complexity is clearly O(n2) because when we increase the size of the problem by a factor n, the time spent will be increase by a factor of n2.

Activity 4. [QUICKSORT ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| Quicksort | | | |
| n | tOrdered | tReverse | tRandom |
| 250000 | LoR | LoR | 95 |
| 500000 | 64 | 72 | 190 |
| 1000000 | 125 | 145 | 401 |
| 2000000 | 256 | 292 | 872 |
| 4000000 | 529 | 606 | 1886 |
| 8000000 | 1104 | 1231 | 4283 |
| 16000000 | 2280 | 2516 | 10359 |

Theoretically, the complexity of the quicksort algorithm (choosing the correct pivots) is O(nlogn). We can see this complexity observing the results obtained. Increasing the size of the problem by a factor of 2, will increase the time by a factor slightly bigger than 2 and by analyzing the algorithm we can know than this slightly difference is the log2.

n=16000000

* As n will be increasing the initial n of the measurements (I will call it b0) of the bubble algorithm by a factor of 160. The time spent with size n will be the time spent in b0\*1602.
* As n will be increasing the initial n of the measurements (I will call it s0) of the selection algorithm by a factor of 160. The time spent with size n will be the time spent in s0\*1602.
* As n will be increasing the initial n of the measurements (I will call it i0) of the insertion algorithm by a factor of 160. The time spent with size n will be the time spent in t0\*160\*log160.

Activity 5. [QUICKSORTINSERTION ALGORITHM]

|  |  |
| --- | --- |
| n = 16000000 | |
| alg | tRandom |
| Quicksort | 10304 |
| Quicksort+Insertion (k=5) | 10745 |
| Quicksort+Insertion (k=10) | 10640 |
| Quicksort+Insertion (k=20) | 10423 |
| Quicksort+Insertion (k=30) | 10328 |
| Quicksort+Insertion (k=50) | 9894 |
| Quicksort+Insertion (k=100) | 8980 |
| Quicksort+Insertion (k=200) | 7403 |
| Quicksort+Insertion (k=500) | 10171 |
| Quicksort+Insertion (k=1000) | 17949 |

Quicksort is the fastest algorithm between this four, so if we use it as much as we can the time spent will be reduced. However, the insertion algorithm