Activity 1. Direct exchange or Bubble algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| n | T ordered | T reverse | T random |
| 10000 | LoR | 66 | LoR |
| 2\*10000 | 101 | 241 | 331 |
| 2\*\*2\*10000 | 113 | 963 | 1962 |
| 2\*\*3\*10000 | 1557 | 3857 | 7425 |
| 2\*\*4\*10000 | 6265 | 15637 | 31561 |

The bubble algorithm doesn’t use divide and conquer so it will always iterate through the same amount regardless of how the matrix is ordered. So, it will always have complexity O(n^2). The difference in time comes from the if, which is called less times when the matrix is ordered.

Activity 2. Selection algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| n | T ordered | T reverse | T random |
| 10000 | LoR | LoR | LoR |
| 2\*10000 | LoR | 77 | 60 |
| 2\*\*2\*10000 | 110 | 265 | 270 |
| 2\*\*3\*10000 | 459 | 1209 | 1050 |
| 2\*\*4\*10000 | 1771 | 4918 | 4130 |

O(n^2 ) in all cases

Activity 3. Insertion algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| n | T ordered | T reverse | T random |
| 10000 | LoR | LoR | LoR |
| 2\*10000 | LoR | 133 | 72 |
| 2\*\*2\*10000 | LoR | 144 | 68 |
| 2\*\*3\*10000 | LoR | 553 | 266 |
| 2\*\*4\*10000 | LoR | 2237 | 1058 |
|  |  |  |  |
| 2\*\*13\*10000 | LoR | OoT | OoT |

Best case(n), Worst and average case(n^2)

Activity 4. Quicksort algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| n | T ordered | T reverse | T random |
| 250000 | LoR | LoR | LoR |
| 2\*250000 | LoR | LoR | LoR |
| 2\*\*2\*250000 | LoR | LoR | 79 |
| 2\*\*3\*250000 | LoR | LoR | 154 |
| 2\*\*4\*250000 | LoR | LoR | 326 |
| 2\*\*5\*250000 | 68 | 84 | 721 |
| 2\*\*6\*250000 | 144 | 169 | 1618 |

Best and average case O(n \* log(n)), worst case O(n^2)

Activity 5. Quicksort + Insertion algorithm

|  |  |
| --- | --- |
| n | T random |
| Quicksort | 1618 |
| Quicksort + Insertion  (k = 5) | 1845 |
| Quicksort + Insertion  (k = 10) | 1782 |
| Quicksort + Insertion  (k = 20) | 1812 |
| Quicksort + Insertion  (k = 30) | 1885 |
| Quicksort + Insertion  (k = 50) | 1805 |
| Quicksort + Insertion  (k = 100) | 1573 |
| Quicksort + Insertion  (k = 200) | 1283 |
| Quicksort + Insertion  (k = 500) | 1207 |
| Quicksort + Insertion  (k = 1000) | 1470 |

We can see that the time that it takes to execute the algorithm is at first slower than the normal Quicksort, this is because the parameter k is not optimized, so we call Insertion when simply continuing with the Quicksort will be more effective. It is approximately around k = 100 that this algorithm becomes faster than the regular Quicksort and it is the fastest at k = 500. However, when k = 1000 we can start seeing that it becomes slower again (although still faster than regular Quicksort), so this could indicate that for very big parameters k it also becomes less optimized.