# Activity 1. [TABLE 1 = BUBBLE ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| ***n*** | ***t ordered*** | ***t reverse*** | ***t random*** |
| 10000 | 310 | 1429 | 991 |
| 2\*10000 | 1216 | 5713 | 3942 |
| 2\*\*2\*10000 | 4895 | 22721 | 15943 |
| 2\*\*3\*10000 | 19877 | OoT | OoT |
| 2\*\*4\*10000 | OoT | OoT | OoT |

t\_ordered = O(n^2)

Escuela de Ingeniería Informática



t\_reverse = O(n^2)

t\_random = O(n^2)

Execution times differences happened because of the execution of the “Vector.*interchange*(a, j-1, j); //swap” . Although it is a O(1) method, it execution takes place in each iteration of the t\_reverse loop, and some random times in t\_random, while it is never executed for t\_ordered (it is executed **if** (a[j-1] > a[j]))

Activity 2 [TABLE 2 = SELECTION ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| ***n*** | ***t ordered*** | ***t reverse*** | ***t random*** |
| 10000 | 310 | 283 | 307 |
| 2\*10000 | 1221 | 1128 | 1217 |
| 2\*\*2\*10000 | 4898 | 4517 | 4885 |
| 2\*\*3\*10000 | 19463 | 18006 | 19409 |
| 2\*\*4\*10000 | OoT | OoT | OoT |

`

They are all the same complexity O(n^2) . Also, they are the same complexity as previous methods, but here “Vector.interchange(a, j-1, j); //swap” is executed once per loop (always), instead of inside the nested loop.

Activity 3 [TABLE 3 = INSERTION\_ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| ***n*** | ***t ordered*** | ***t reverse*** | ***t random*** |
| 10000 | LoR | 291 | 148 |
| 2\*10000 | LoR | 1137 | 572 |
| 2\*\*2\*10000 | LoR | 4618 | 2301 |
| 2\*\*3\*10000 | LoR | 18235 | 9207 |
| 2\*\*4\*10000 | LoR | OoT | 36770 |
| 2\*\*5\*10000 | LoR | OoT | OoT |
| 2\*\*6\*10000 | LoR | OoT | OoT |
| 2\*\*7\*10000 | LoR | OoT | OoT |
| 2\*\*8\*10000 | 50 | OoT | OoT |
| 2\*\*9\*10000 | 99 | OoT | OoT |
| 2\*\*10\*10000 | 185 | OoT | OoT |
| 2\*\*11\*10000 | 373 | OoT | OoT |
| 2\*\*12\*10000 | 744 | OoT | OoT |
| 2\*\*13\*10000 | 1484 | OoT | OoT |

They are all O(n) but t\_ordered has a massive advantage when it comes to time-efficiency because of the while loop inside of the for loop **while** (j >= 0 && pivot < a[j]) only taking place for t\_reverse and t\_random (in t\_reverse it takes the maximum possible iterations).

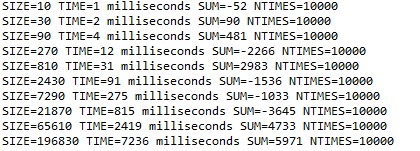
# Activity 3.1 [TABLE 4 = QUICKSORT ALGORITHM]

|  |  |  |  |
| --- | --- | --- | --- |
| ***n*** | ***t ordered*** | ***t reverse*** | ***t random*** |
| 250000 | LoR | LoR | 100 |
| 2\*250000 | 62 | 71 | 199 |
| 2\*\*2\*250000 | 127 | 143 | 441 |
| 2\*\*3\*250000 | 260 | 291 | 939 |
| 2\*\*4\*250000 | 534 | 604 | 2007 |
| 2\*\*5\*250000 | 1089 | 1234 | 4516 |
| 2\*\*6\*250000 | 2244 | 2554 | 11073 |

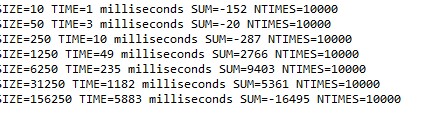
# Activity 3.2 [What happens with time if the problem size is multiplied by a value k other than 2? (try it, for example, for k=3 and k=4 and check the times obtained)

K=3

Escuela de Ingeniería Informática



K=5



# Activity 3.3 [Tables MAXIMUM)

CPU1: 12th Gen Intel(R) Core(TM) i5-12400 2.50 GHz

RAM1: 16,0 GB

Escuela de Ingeniería Informática

|  |  |  |
| --- | --- | --- |
| **Size** | **T(max)** | **T(sum)** |
| 10000 | 0.058 | 0.0377 |
| 20000 | 0.113 | 0.0743 |
| 40000 | 0.220 | 0.1482 |
| 80000 | 0.445 | 0.2927 |
| 160000 | 0.890 | 0.5873 |
| 320000 | 1.769 | 1.1747 |
| 640000 | 3.566 | 2.3510 |
| 1280000 | 7.095 | 4.7113 |
| 2560000 | 14.125 | 9.4374 |

# Activity 3.4 [Tables MATCHES)

CPU1: 12th Gen Intel(R) Core(TM) i5-12400 2.50 GHz

RAM1: 16,0 GB

**Siza T(matches2) T(matches1)**

Escuela de Ingeniería Informática

|  |  |
| --- | --- |
| 10000 | 0.06 511 |
| 20000 | 0.1 2026 |
| 40000 | 0.21 8065 |
| 80000 | 0.42 32244 |
| 160000 | 0.85 130349 |
| 320000 | 1.67 - |
| 640000 | 3.35 - |
| 1280000 | 6.7 - |
| 2560000 | 13.4 - |
| 5120000 | 27.05 - |
| 10240000 | 54 - |
| 20480000 | 110 - |
| 40960000 | 218 - |

T(Sum) is linear O(n)

T(Max) is linear O(n)

T(matches1) is quadratic O(n^2)

T(matches2) is linear O(n)