Mikel Fernández Esparta

UO275688

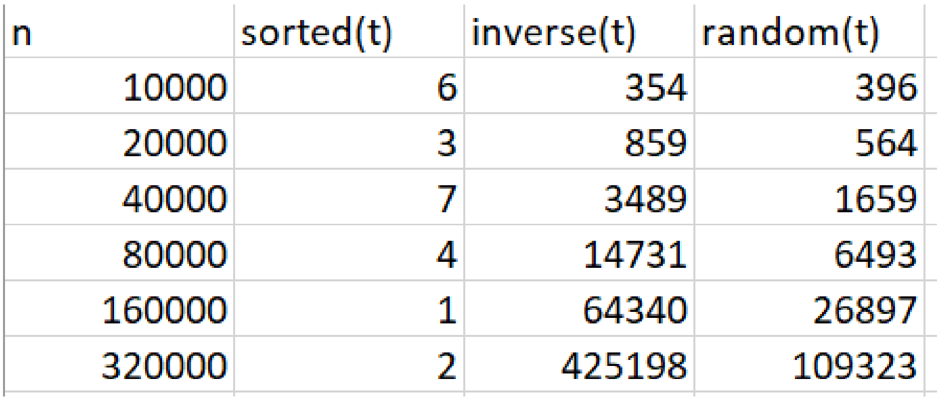
Algorithmics

LAB I-3

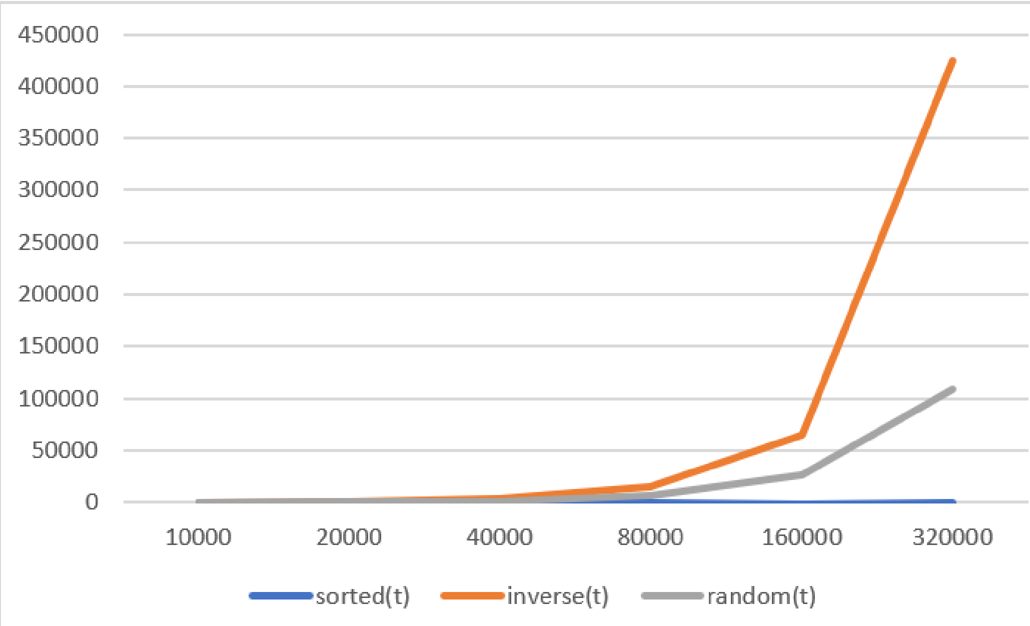
Activity 1. Time measurements for sorting algorithms.

Four tables with times for each of the algorithms (Insertion, Selection, Bubble and Quicksort with the central element as the pivot). An example of one of the tables is below:

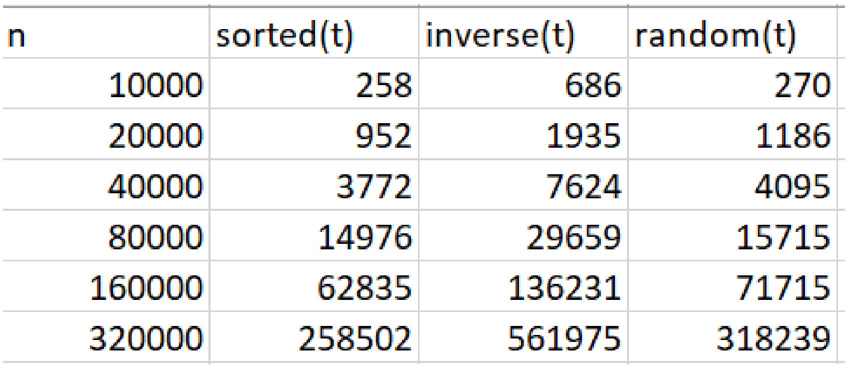
- **Insertion**:
  The values make sense, the lower the time, the faster and better the algorithm is as we can see when calculating the complexity for sorting, however on both the inverse and random, it keeps on iterating which is bad, so the complexity is worse.



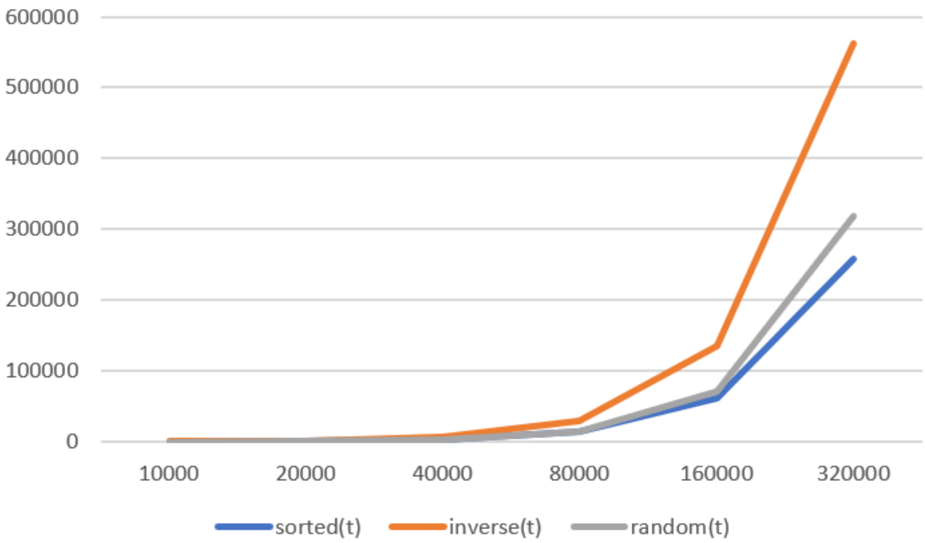
n	sorted(t)	inverse(t)	random(t)
10000	6	354	396
20000	3	859	564
40000	7	3489	1659
80000	4	14731	6493
160000	1	64340	26897
320000	2	425198	109323



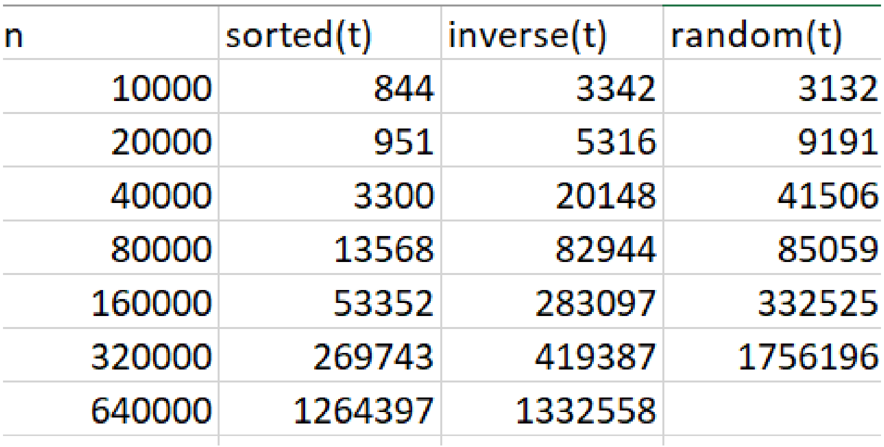
- **Selection**:
  Consists on selecting the lowest element and exchanging it with the first element, as we can see, it is evident that this algorithm is worse since it grows higher, therefore it is slower. Both best and worst case are O(n^2), the number of comparisons is very high.



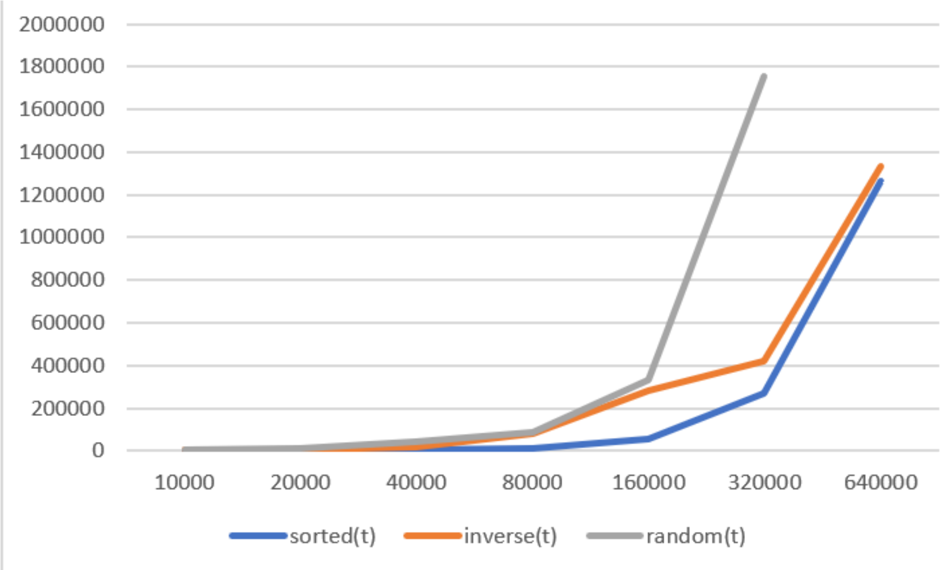
n	sorted(t)	inverse(t)	random(t)
10000	258	686	270
20000	952	1935	1186
40000	3772	7624	4095
80000	14976	29659	15715
160000	62835	136231	71715
320000	258502	561975	318239



- Buble:
  In this case, the worst case is the random algorithm, since it grows a lot, and it keeps on iterating the array, which is not what we strive for.

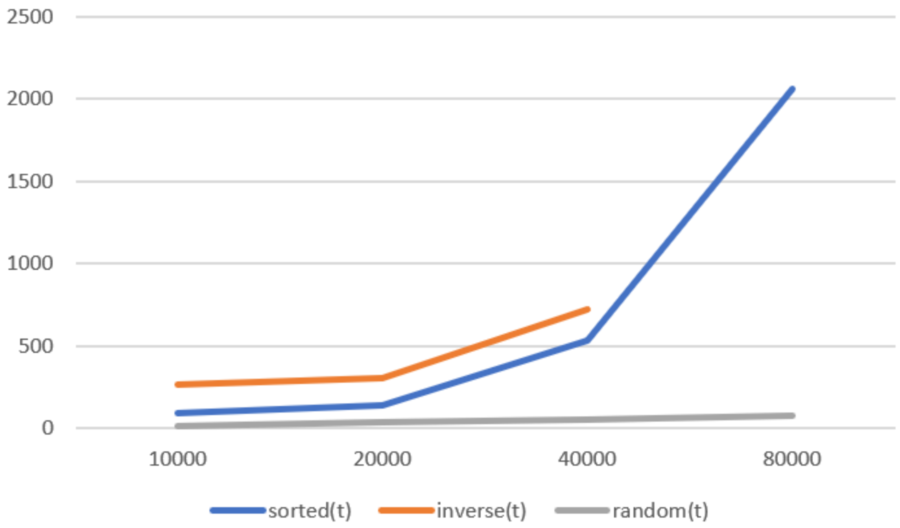


n	sorted(t)	inverse(t)	random(t)
10000	844	3342	3132
20000	951	5316	9191
40000	3300	20148	41506
80000	13568	82944	85059
160000	53352	283097	332525
320000	269743	419387	1756196
640000	1264397	1332558	



- **QuicksortCentralElement**:
  The values for both sorted and inverse stop almost at the beginning of the execution because there is stack overflow. However the random seems to work fine and the values are low so that's what we are looking for in order to have a better complexity than the previous algorithms, such as Bubble that is really bad.

n	sorted(t)	inverse(t)	random(t)
10000	91	265	16
20000	139	302	36
40000	533	725	57
80000	2060		81
160000			262
320000			493
640000			820
1280000			1562
2560000			3275
5120000			6901



Activity 2. QuicksortFateful

Briefly explain what the criteria is for selecting the pivot in that class. Indicate when that idea can work and when that idea will not work.

Quicksort is based on portioning the array of values, we choose a pivot to partition and with it, we start creating a tree with the lower values to its left and the higher ones to the right. This is a recursive method which means it keeps on iterating. The ideal choices are either the median of the elements or the central ones