Session 2

UO283069

Activity 1: The measurements of sorting algorithms

Insertion

n	*sorted(t)*	*inverse(t)*	*random(t)*
10000	19	77	42
20000	7	289	163
40000	14	285	161
80000	31	1191	571
160000	19	4830	2265
320000	107	18879	9224
640000	251	75973	36909
1280000	458	327638	155589

The expected values are very similar when using an inversed vector, but for the sorted and random vectors the actual times are almost the double as the theoretical ones until a high size for n is reached (calculating these theoretical values for its worst-case complexity O(n^2)).

The times do also not match the ones calculated for its O(n) complexity on the best-case scenario.

Selection

n	*sorted(t)*	*inverse(t)*	*random(t)*
10000	24	53	28
20000	63	188	64
40000	283	616	275
80000	940	2352	1023
160000	3511	9350	3964
320000	13510	36643	15661
640000	52801	145582	59705
1280000	225250	602398	232082

The values make sense for a O(n^2) complexity and the theoretical and actual values for this algorithm are very similar, only differing in a couple of units, except for the randomly sorted vector, whose theoretical times differ significantly until it reaches a size of 320000.

Bubble

n	*sorted(t)*	*inverse(t)*	*random(t)*
10000	62	66	91
20000	210	309	277
40000	236	465	565
80000	3318	3528	3040
160000	13121	14046	18018
320000	52545	56774	119061
640000	210506		
1280000			

Once again, the calculated theoretical values are very similar for the sorted and inversed vector, but the random one differs from its theoretical values, being the actual ones twice as big as the theoretical.

The other values make sense, since Bubble has a O(n^2) complexity (since we are not using a sentinel).

Quicksort with central element

n	*sorted(t)*	*inverse(t)*	*random(t)*
10000	32	15	81
20000	29	16	175
40000	58	55	335
80000	96	108	676
160000	195	206	1714
320000	439	481	6343
640000	1180	1425	10523
1280000	3362	3698	12783
Until Exc			

The times for the algorithm are very similar to those calculate theoretically for its complexity of O(nlogn) and, obviously, differ from its worst-case scenario complexity O(n^2).

Activity 2: Quicksort Fateful

The criteria that the Quicksort Fateful algorithm follows in order to select the pivot of the class is always selecting the first element of the array.

This might work for a randomly ordered vector where the median of three or the center element could not be an appropriate selection, which would mean that the calculation of the pivot would have been not as effective as it needs to be, and thus using the first element of the array would solve this problem, but whenever we have an already sorted array or an array with inversed sorted elements, the ‘fateful’ selection would not be a great choice.

Therefore, Quicksort Fateful would be a bad decision for arrays that are ordered or inversely ordered, since the pivot would always be greater than one of the two values that we are checking to swap.