Ruslan Bessarab

Big-O analysis

1. This algorithm does best when the input elements are evenly distributed. Explain why this is an important observation.

First of all, this algorithm is fast because it is not based on comparison between elements but instead compare each element against a group and put it into (here it uses insertion sort which is good for small inputs or small arrays). Second, this algorithm does best when the input elements are evenly distributed because we it keeps each group of elements small enough to do a quick insertion sort.

2. Give a big-O estimate for your sort routine. Your estimate should take into account both the input array size (n) and the number of groups (k).

3. Give a justification (explanation) for your big-O estimate.

First, I would like to break down my algorithm and then put it all together.

a. Find low – n (worst case and depends only on input)

b. Find high – n (worst case and depends only on input)

c. Find group size – 1 (constant because it always does same calculation and not depends on input or number of groups).

d. Create an array of thresholds – k (worst case and depends on number of groups - k).

With that saying, first part of algorithm takes about 2n + k.

e. Go over each element in the input and compare it against each group – depends on both input and number of groups. Two worst cases are – k = 1(then it will take quadratic runtime to sort elements in that array of nodes), other one if k = n (it will take quadratic runtime). So, the worst case would take O(n^2), in average case it would take O (n log n).

f. put each element from Node [] to original array – n.

If we put these all together it would be 2n + k + n log n + n -> O(n log n)