**Merge binary insertion sort**

The algorithm we wrote takes advantage of merge sort and binary insertion sort, two distinct sorting algorithms. Depending on the k value from user input the algorithm will decide which algorithm to use between the two. In other words k value indicates subset of the total number of elements on which we call binary insertion sort, otherwise we call merge sort.

When we are provided with an array of elements. We generally prefer to merge sort. It is quite beneficial when the number of elements is large O(nlogn). But for the cases when the numbers of elements are less, we must do a binary insertion sort. The binary Insertion Sort can be most efficiently used when the number of elements are less.

Binary insertion sort is simply an insertion sort. It is just implemented using binary search instead of linear search. Changing the type of search improves the time complexity of the sorting algorithm. It is because the comparison we do is reduced for one element from O(n) to O(logn).

It will work faster when the array is already sorted (the best case) O(nlogn). Thus, making it an adaptive algorithm. The worst case is when we have an array that is opposite of a sorted array O(n^2).

| **Integer field** | | Chart |
| --- | --- | --- |
| **k** | **time** |
| 10 | 7.155 |
| 20 | 7.339 |
| 30 | 7.203 |
| 40 | 7.359 |
| 50 | 7.37 |
| 60 | 7.4 |
| 70 | 7.392 |
| 80 | 7.626 |
| 90 | 7.713 |
| 100 | 9.426 |
| 200 | 15.393 |
| 300 | 9.763 |
| 400 | 19.927 |
| 500 | 21.078 |
| 600 | 19.764 |
| 700 | 19.014 |
| 800 | 22.071 |
| 900 | 19.305 |
| 1000 | 15.662 |