Hanifi Demirel,150120039

**Report**



|  |  |  |  |
| --- | --- | --- | --- |
|  | Quick Sort | Radix Sort | Counting Sort |
| 1k-10k | 0.000283 | 0.000283 | 0.000213 |
| 1k-10m | 0.000304 | 0.000468 | 0.071606 |
| 100k-10k | 0.034464 | 0.022588 | 0.013406 |
| 100k-10m | 0.013406 | 0.039372 | 0.089849 |

2. There two main method to choose pivot: Either we choose last element every time as pivot or we choose a random element. If we choose last element in every iteration, in the worst case the algorithm performs in O(n^2) time. On the other hand, if we choose pivot randomly in each iteration, the expected running time of the algorithm becomes O(nlogn) even with the most unbalanced split possible. The prove to this can be found on the book “Introduction to Algorithms”, Chapter 7.
3. The worst case behaviour of quicksort for my implementation is that the situation when array is already sorted in decreasing or increasing order. Because, in such situation, running time for the function would be:

T(n) = T(n-1) + Θ(n)

Consequently, the running time of the algorithm would be Θ(n^2).

1. *k* stands for the element with the highest value in the input array. We have to specify it because in the Counting Sort algorithm, there are two for loops which take time O(k). And the overall time for the algorithm is O(n+k).

Also, most of the time, counting sort is used when k=O(n) which lead to O(n) time for the

algorithm. If k is asymptotically bigger than n, running time is O(k)

1. The worst case is when the longest integer is longer than the array size. Time complexity is O(k), *k* asymptotically bigger than *n*.