Wyniki (podane w sekundach):

Mniejszy zbiór danych (15 000 wartości, zakres 99999)

15 000 values	bubblesort	insertion sort	counting-position sort	quicksort	total
random array	17,16563	7,17893	0,06096	0,04076	24,44627
sorted array	17,87495	7,16753	0,06305	0,04284	25,14837
reversed-sorted array	19,26524	7,28487	0,06213	0,03109	26,64333

Większy zbiór danych (1 000 000 wartości, zakres 99999)

1 000 000 values	counting-position sort	quicksort	total
random array	6,24012	5,31495	11,55507
sorted array	6,24044	5,03632	11,27676
reversed-sorted array	6,15077	5,05326	11,20403

Większy zbiór danych (1 000 000 wartości, zakres 999999)

1 000 000 values	counting-position sort	quicksort	total
random array	8,08755	4,82208	12,91605
sorted array	8,00896	4,80908	12,81804
reversed-sorted array	8,49697	4,9506	13,44757

Wnioski:

W przypadku mojej implementacji insertion sort radzi sobie dużo lepiej niż bubblesort (mimo podobnej złożoności obliczeniowej $O(n^2)$), ale oba te algorytmy są wielokrotnie wolniesze od sortowania pozycyjnego przez zliczanie oraz quicksorta.

Niestety z powodu ograniczenia sprzętowego wolniejsze algorytmy mogłem przetestować jedynie na niewielkich zbiorach danych, ale wyniki jakie osiągnąłem pozwalają zaobserwować wyraźne różnice w czasie wykonywania.

Zauważyłem, że w przeciwieństwie do sortowania pozycyjnego, quicksort działa wolniej dla zbiorów danych w których wielokrotnie pojawiają się zduplikowane wartości. Sortowanie pozycyjne gorzej radzi sobie ze zbiorami o dużym zakresie (dużej maksymalnej wartości).

Dla dużych danych o dużym zakresie najlepszym algorytmem sortującym okazał się quicksort.

Link do repozytorium:

https://github.com/marekk3301/pythonASD