Analysis results text

Selection sort

This algorithm is not the best for sorting numbers with length [50, 300], and with values from all input types, but it is not the worse one, so you can say, its execution time has medium value compared to the remaining algorithms.

But we can note that when input length in interval [50, 300], its execution time varies with the type of the input, so it gives slower execution time with input values from the first type (random values from 0 to 5), and speeder execution time with input values from the third type (numbers almost sorted in the required order)

As for sorting numbers with length [100, 4100] and with values from all input types, it is considered one of the slowest algorithms, so in this case you must forget about using it.

But for input length in interval [100, 4100], its execution time varies with the type of the input, so it gives slower execution time with input values from the last type (reverse sorted (descending) numbers from 4100 to 1), and almost the same execution time for the rest input types.

Simple inserts sort

This algorithm is works very well for sorting numbers with length [50, 300], and with values from all input types (it’s the best choice when input type is the third (numbers almost sorted in the required order)), but for reverse sorted (descending) numbers, almost it gives slow speed value compared to the remaining algorithms (Radix sort - counting sort – Quick sort Binary inserts sort).

But we can note that when input length in interval [50, 300], its execution time varies with the type of the input, so it gives slower

execution time with input values from the last type (reverse sorted (descending) numbers from 4100 to 1), and speeder execution time with input values from the third type (numbers almost sorted in the required order)

As for sorting numbers with length [100, 4100] and with values almost from all input types, it is the second slower algorithm, so in this case you must forget about using it, but you must think firstly about it when you know that your input is numbers almost sorted, because it gives very good results.

But for input length in interval [100, 4100], its execution time varies with the type of the input, so it gives slower execution time with input values from the last type (reverse sorted (descending) numbers from 4100 to 1), and speeder execution time with input values from the third type (numbers almost sorted in the required order).

Binary inserts sort

It is almost no different from the previous algorithm and gives results very close to it.

Counting sort

This algorithm is works very great for sorting numbers with length [50, 300], and with values from all input types (it’s the best choice when input type is the first (random values from 0 to 5)), but for reverse sorted (descending) numbers, almost it gives not very good speed value compared to the remaining algorithms (Radix sort – Simple inserts sort – Quick sort Binary inserts sort).

But we can note that when input length in interval [50, 300], its execution time varies with the type of the input, so it gives slower

execution time with input values from the second type (random values from 0 to 4000), and speeder execution time with input values from the first type (random values from 0 to 5)

As for sorting numbers with length [100, 4100] and with values from all input types, it is absolutely the best choice you can think about.

But for input length in interval [100, 4100], its execution time varies with the type of the input, so it gives slower execution time with input values from the second type (random values from 0 to 4000), and speeder execution time with input values from the first type (random values from 0 to 5)

Radix sort

This algorithm is works wonderfully for sorting numbers with length [50, 300], and with values from all input types, but for almost sorted numbers, still “Simple inserts sort” and “Binary inserts sort” speeder.

But we can note that when input length in interval [50, 300], its execution time varies with the type of the input, so it gives slower execution time with input values from the first type (random values from 0 to 5), and speeder execution time with input values from the last type (reverse sorted (descending) numbers from 4100 to 1), especially when input length between [150 -300].

As for sorting numbers with length [100, 4100] and with values from all input types, it is as Counting sort good choice.

But for input length in interval [100, 4100], its execution time varies with the type of the input, so it gives slower execution time with input values from the first type (random values from 0 to 5), and speeder execution time with the rest of input types.

Merge sort

This algorithm is one of the worse algorithms after “Heapsort” for sorting numbers with length [50, 300], and with values from all input types.

we can note that when input length in interval [50, 300], its execution time almost the same for all input types.

As for sorting numbers with length [100, 4100] and with values from all input types, it gives good results, but still “Counting sort”, “Radix sort” and “Quick sort”, better than it in this case.

But for input length in interval [100, 4100], its execution time varies with the type of the input, so it gives slower execution time with input values from the first type (random values from 0 to 5), and speeder execution time with the rest of input types.

Quicksort

This algorithm is not the best for sorting numbers with length [50, 300], and with values from all input types, but it is not the worse one, so you can say, its execution time has medium value compared to the remaining algorithms, but it gives good results for reverse sorting.

But we can note that when input length in interval [50, 300], its execution time varies with the type of the input, so it gives slower execution time with input values from the first type (random values from 0 to 5), and speeder execution time with input values from the third type (numbers almost sorted in the required order).

As for sorting numbers with length [100, 4100] and with values from all input types, it is considered one of the best algorithms, but still “Counting sort” and “Radix sort” better in this case.

But for input length in interval [100, 4100], its execution time varies with the type of the input, so it gives slower execution time with input values from the first type (random numbers from 0 to 5), and speeder execution time with input values from the third type (numbers almost sorted in the required order).

Heapsort

This algorithm is absolutely the worse algorithm for sorting numbers with length [50, 300], and with values from all input types.

But we can note that when input length in interval [50, 300], its execution time varies with the type of the input, so it gives slower execution time with input values from the third type (numbers almost sorted in the required order), and speeder execution time with input values from the first type (random numbers from 0 to 5).

As for sorting numbers with length [100, 4100] and with values from all input types, it gives good results, but still “Counting sort”, “Radix sort”, “Merge sort” and “Quick sort”, better than it in this case.

But for input length in interval [100, 4100], its execution time varies with the type of the input, so it gives slower execution time with input values from the third type (numbers almost sorted in the required order), and speeder execution time with input values from the first type (random numbers from 0 to 5).

Conclusion

Finally, we can provide a compare table of all sorting algorithms, which will help you to make a correct choice in depend on input length and type:

	random numbers from 0 to 5	random numbers from 0 to 4000	numbers almost sorted in the required order	Reverse sorted (descending) numbers from 4100 to 1
[50 - 300]	Radix - counting	Radix	Simple inserts	Radix
[100 - 4100]	Radix - counting	Radix - counting	Simple inserts – counting – Binary inserts - Radix	Radix - counting