

```
In [ ]: import pandas as pd
import matplotlib.pyplot as plt
```

```
In [ ]: sorts = [
    "selection",
    "bubble",
    "bubble-iverson-1",
    "bubble-iverson-2",
    "insertion",
    "bin-insertion",
    "counting",
    "radix",
    "merge",
    "quick",
    "heap",
    "shell-ciura",
    "shell"
]

arrays = [
    "small-range",
    "big-range",
    "almost-sorted",
    "reversed"
]
```

```
In [ ]: data = pd.read_csv('../data/ops-small.csv', sep=';', header=None)
data.columns = ['sort', 'array', 'size', 'ops']
```

```
In [ ]: def print_sort(data, sort):
    sort_df = data[data['sort'] == sort]
    plt.figure(figsize=(20, 20))
    for array in arrays:
        df = sort_df[sort_df['array'] == array]
        plt.plot(df['size'], df['ops'], label=array)
    plt.title(sort)
    plt.xlabel('Array Size')
    plt.xticks(sort_df['size'].unique())
    plt.ylabel('Operations count')
    plt.legend(labelcolor='black', prop={'size': 15})
```

```
In [ ]: def print_array(data, array):
    array_df = data[data['array'] == array]
    plt.figure(figsize=(20, 20))
    for sort in sorts:
        df = array_df[array_df['sort'] == sort]
        plt.plot(df['size'], df['ops'], label=sort)
    plt.title(array)
    plt.xlabel('Array Size')
    plt.xticks(array_df['size'].unique())
    plt.ylabel('Operations count')
    plt.legend(labelcolor='black', prop={'size': 15})
```

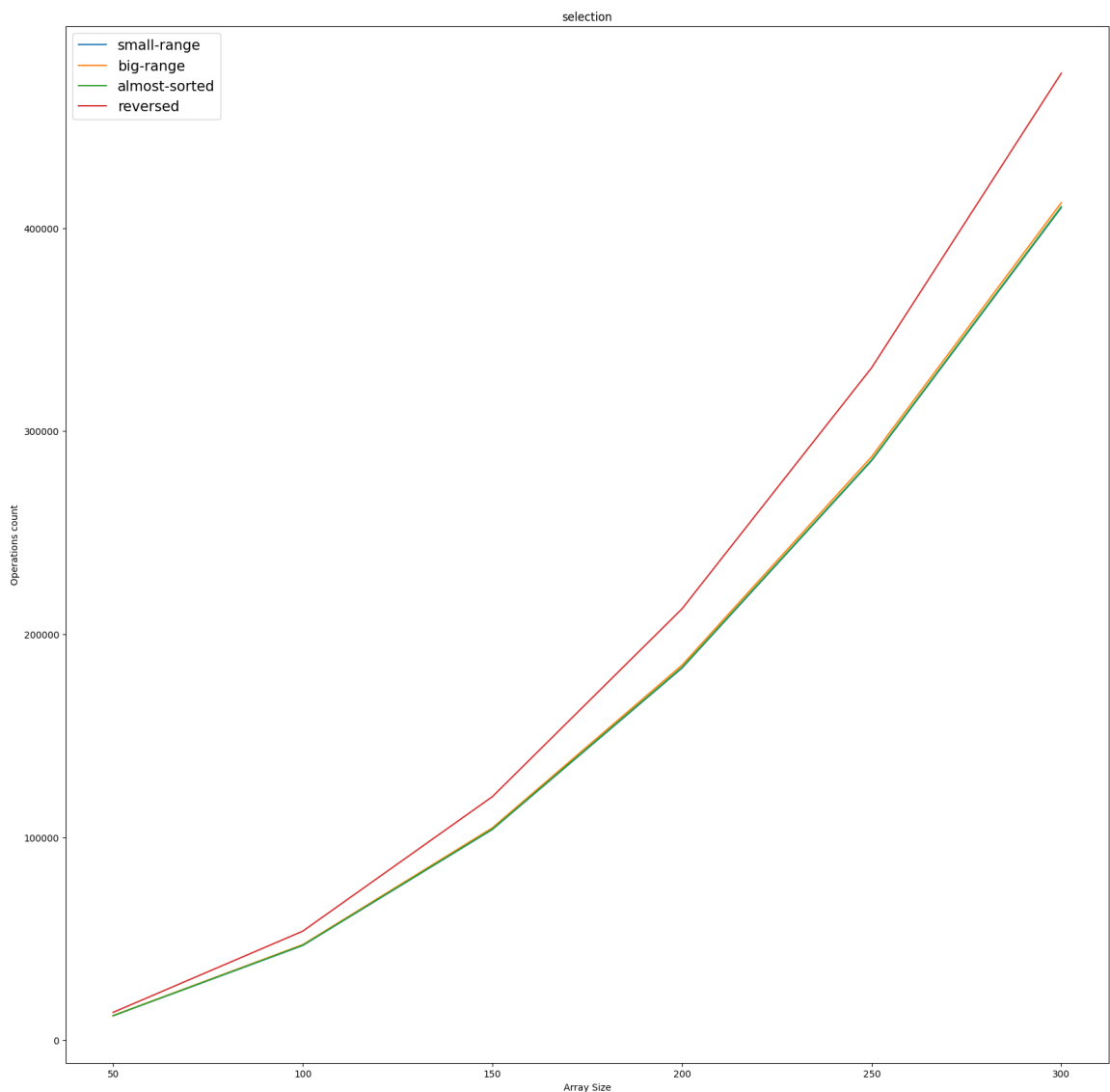
# Зависимость количества элементарных операций от размера массива

для размера массива от 100 до 4000, шаг 100

## По сортировкам

### 1. Выбором

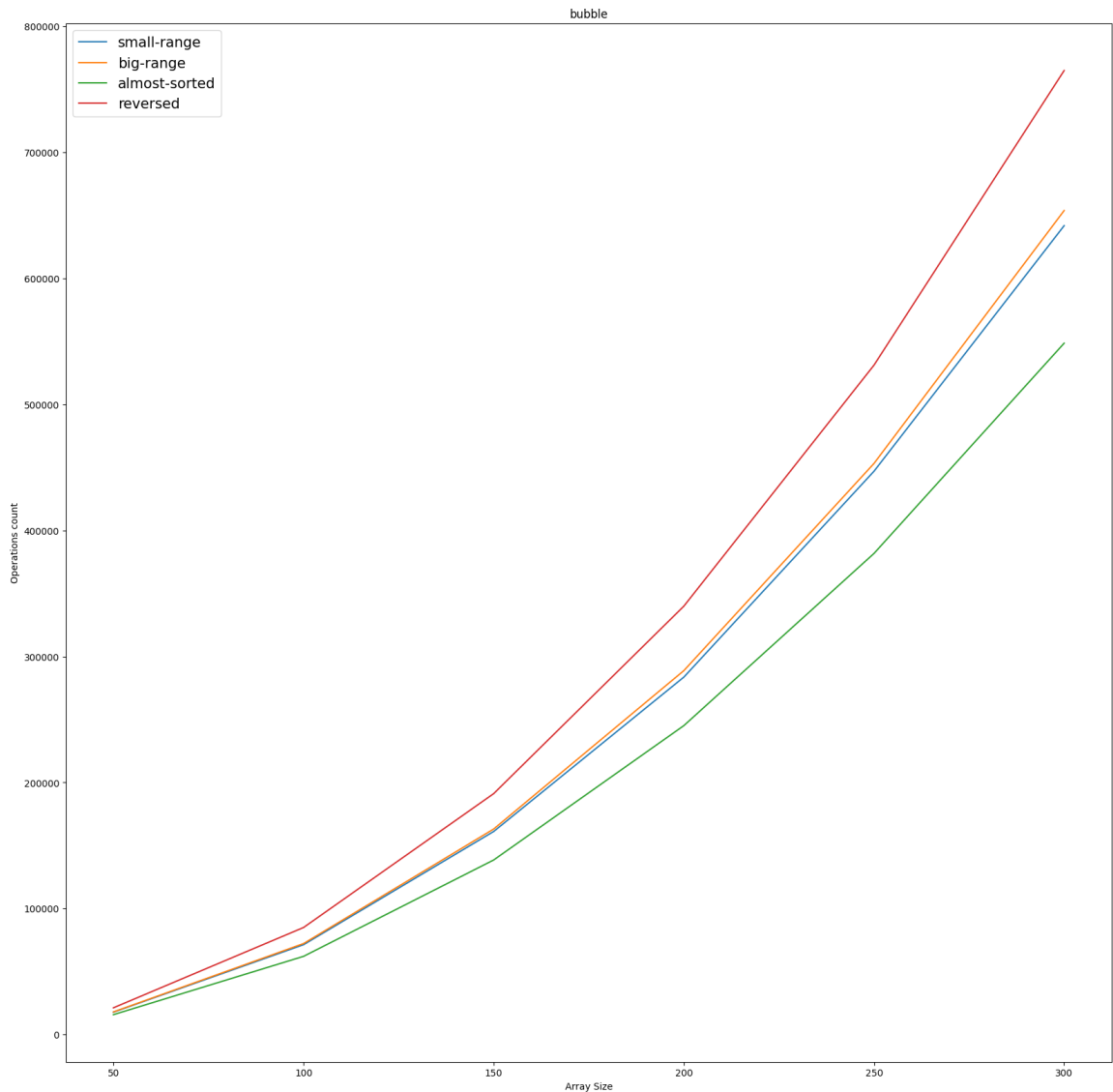
```
In [ ]: print_sort(data, sorts[0])
```



**Вывод:** очевидное дублирование вывода из отчёта по измерению времени: причина большего числа операций для `reversed` массива в дополнительном присваивании минимального элемента

### 2. Пузырьком

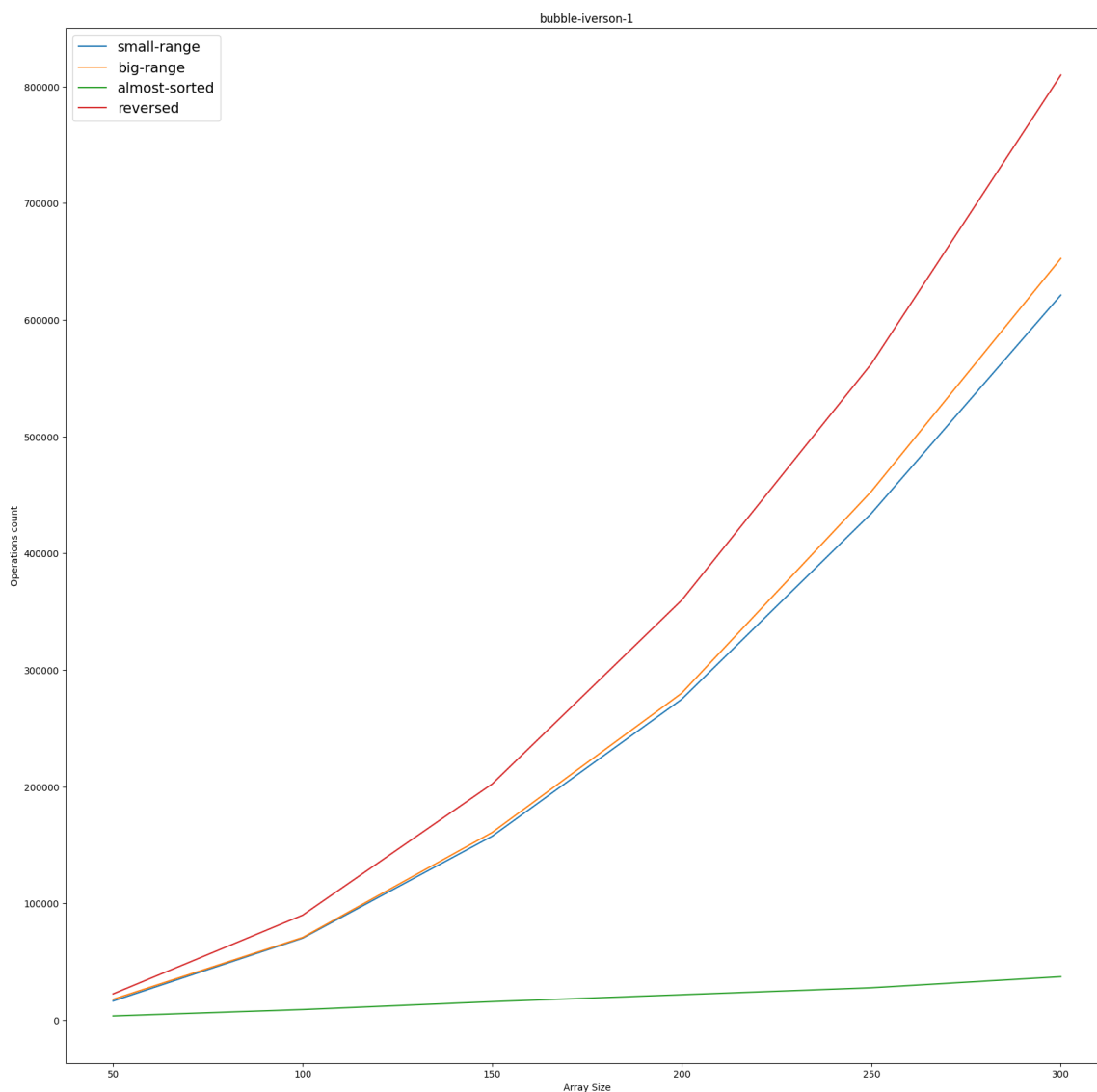
```
In [ ]: print_sort(data, sorts[1])
```



**Вывод:** повторяюсь, что алгоритм чувствителен к порядку элементов => от этого зависит количество свопов и суммарное число операций

### 3. Пузырьком с условием Айверсона 1

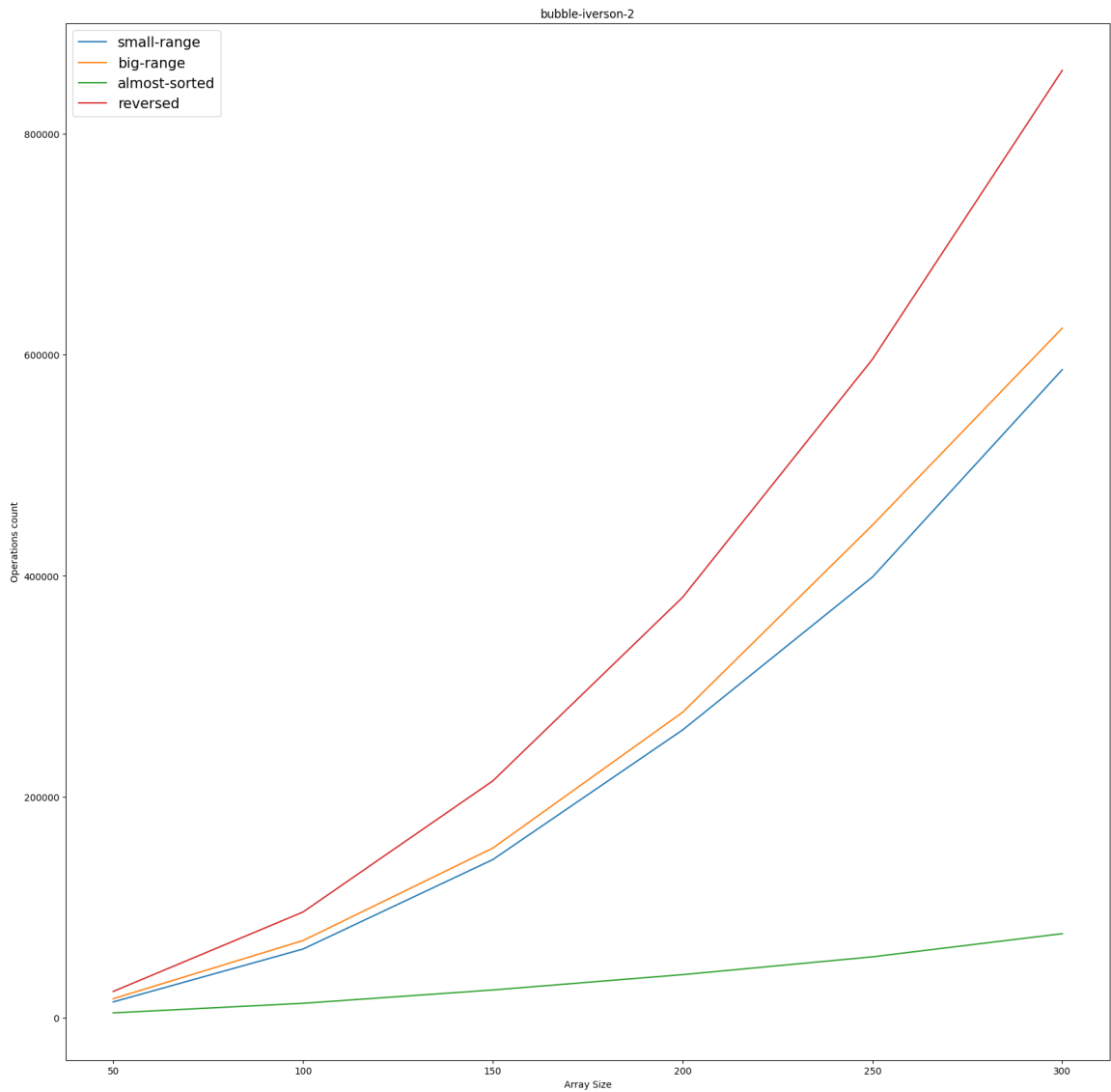
```
In [ ]: print_sort(data, sorts[2])
```



**Вывод:** видим, как алгоритм становится лучше для почти отсортированных массивов

#### 4. Пузырьком с условием Айверсона 1 + 2

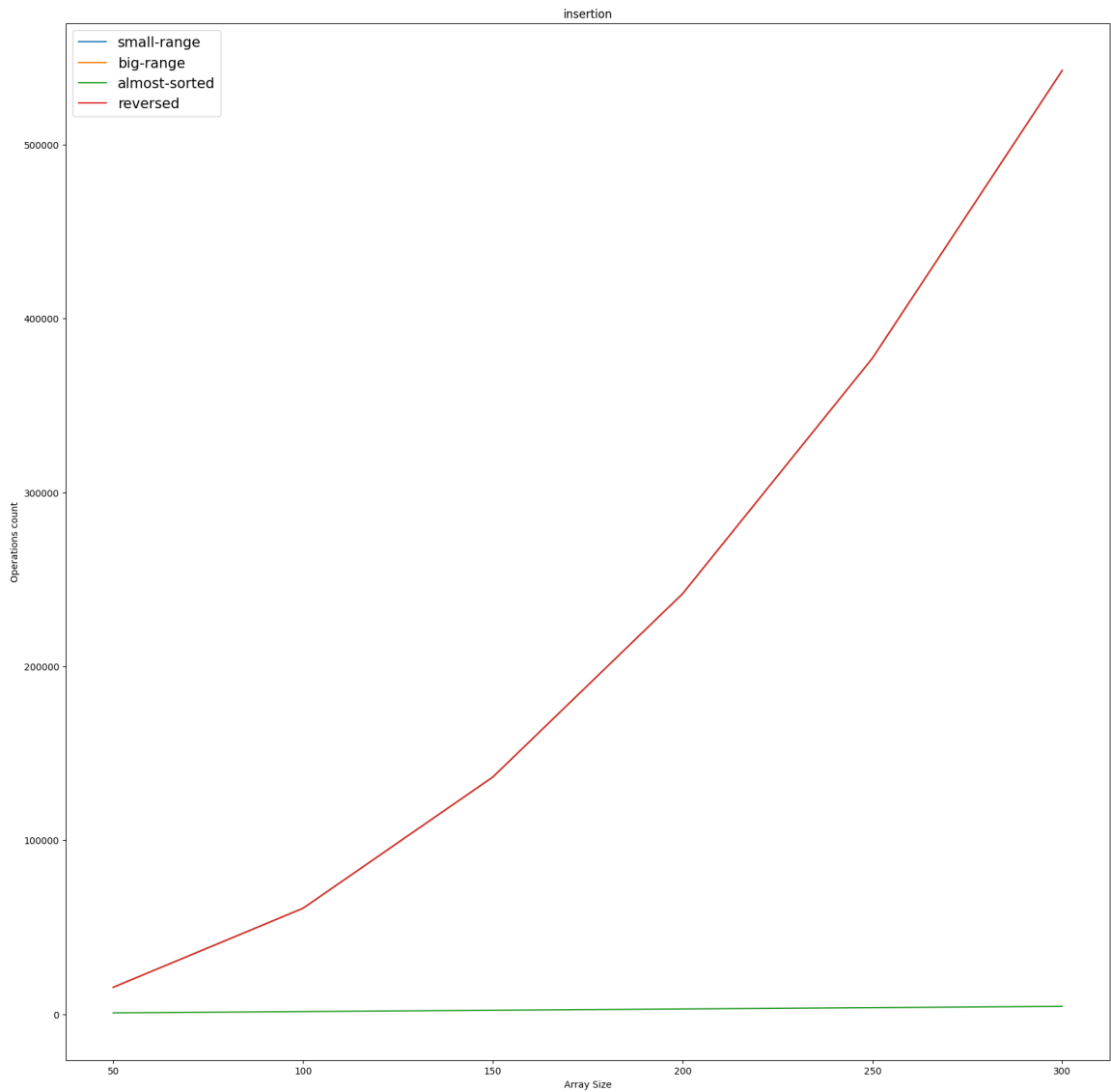
```
In [ ]: print_sort(data, sorts[3])
```



**Вывод:** аналогично предыдущему пункту

## 5. Простыми вставками

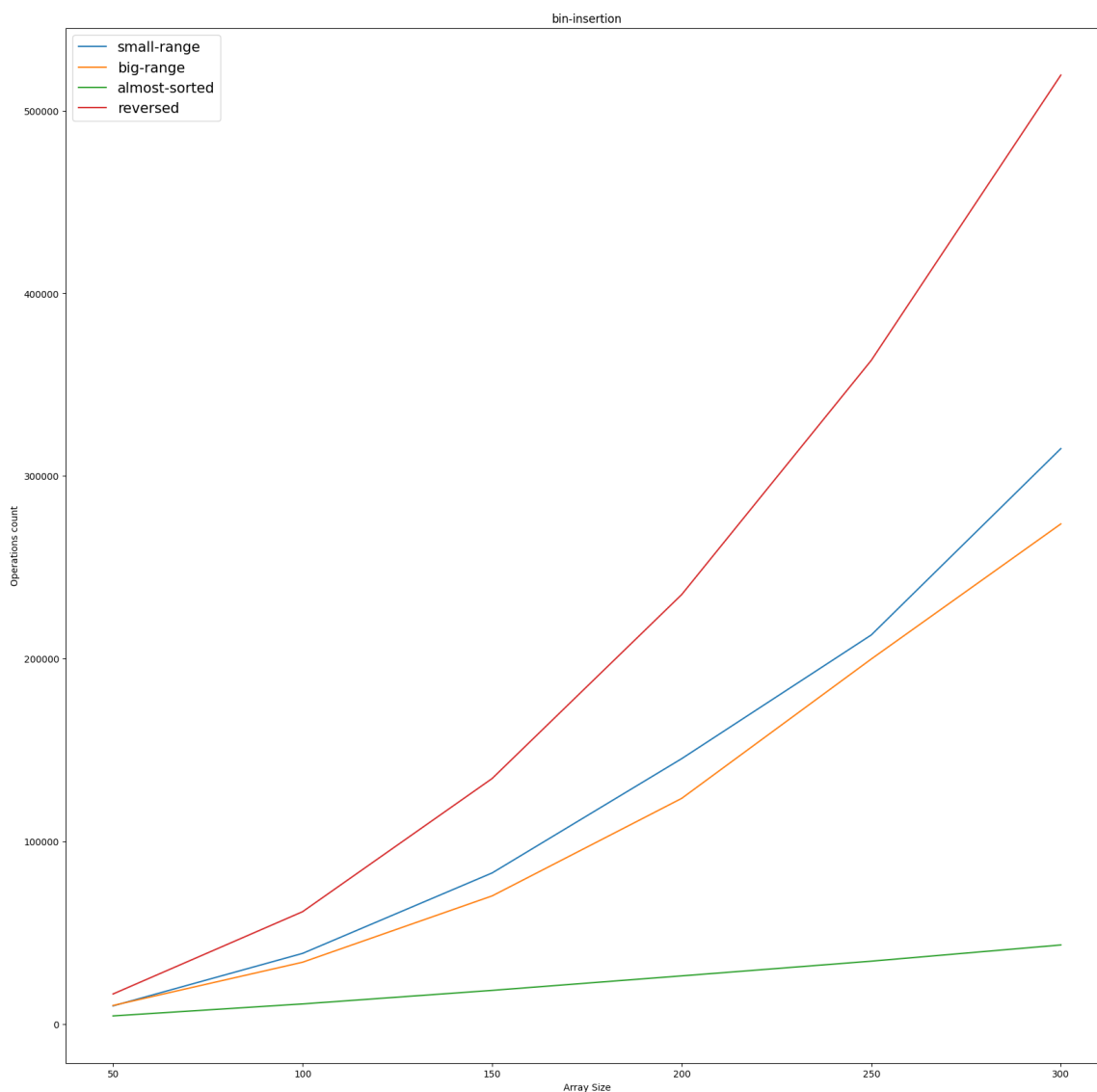
```
In [ ]: print_sort(data, sorts[4])
```



**Вывод:** аналогично случаю для небольших массивов

## 6. Бинарными вставками

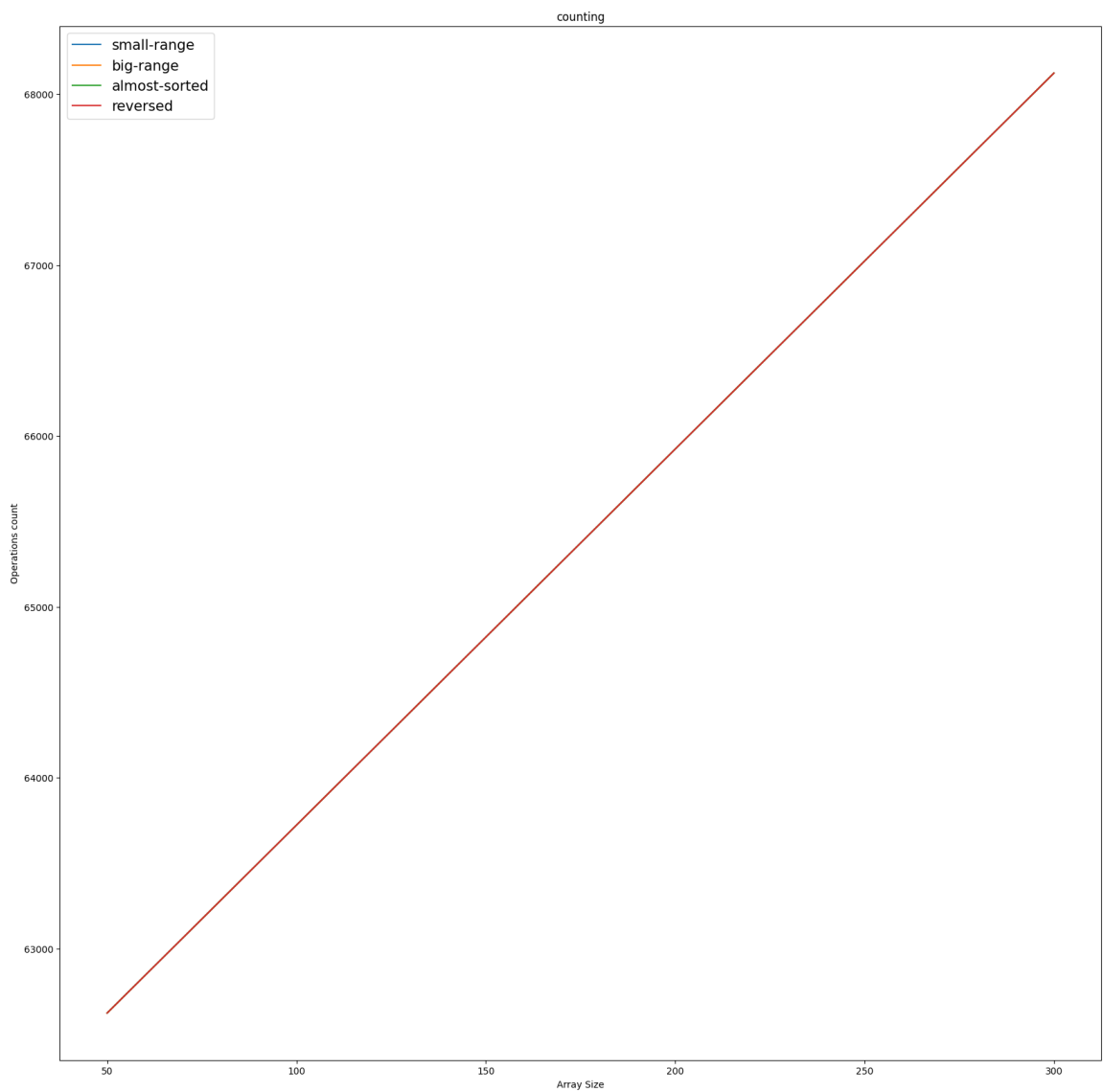
```
In [ ]: print_sort(data, sorts[5])
```



**Вывод:** тут ожидаемый результат для сортировки вставками: порядок элементов влияет на количество свопов

## 7. Подсчётом

```
In [ ]: print_sort(data, sorts[6])
```

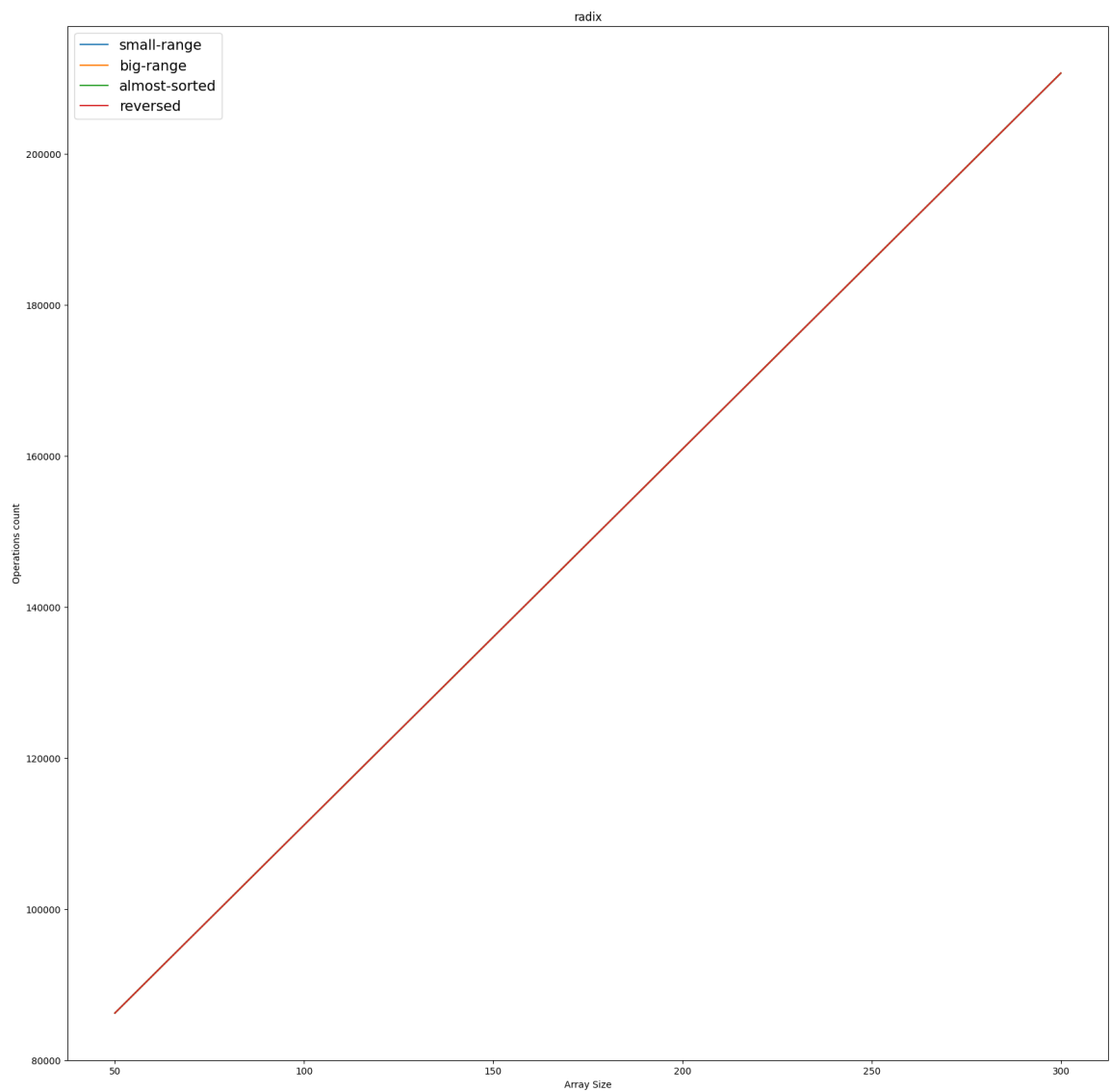


**Вывод:** ожидаемая линия для вне зависимости от набора элементов

## 8. Цифровая

```
In [ ]: print_sort(data, sorts[7])
```

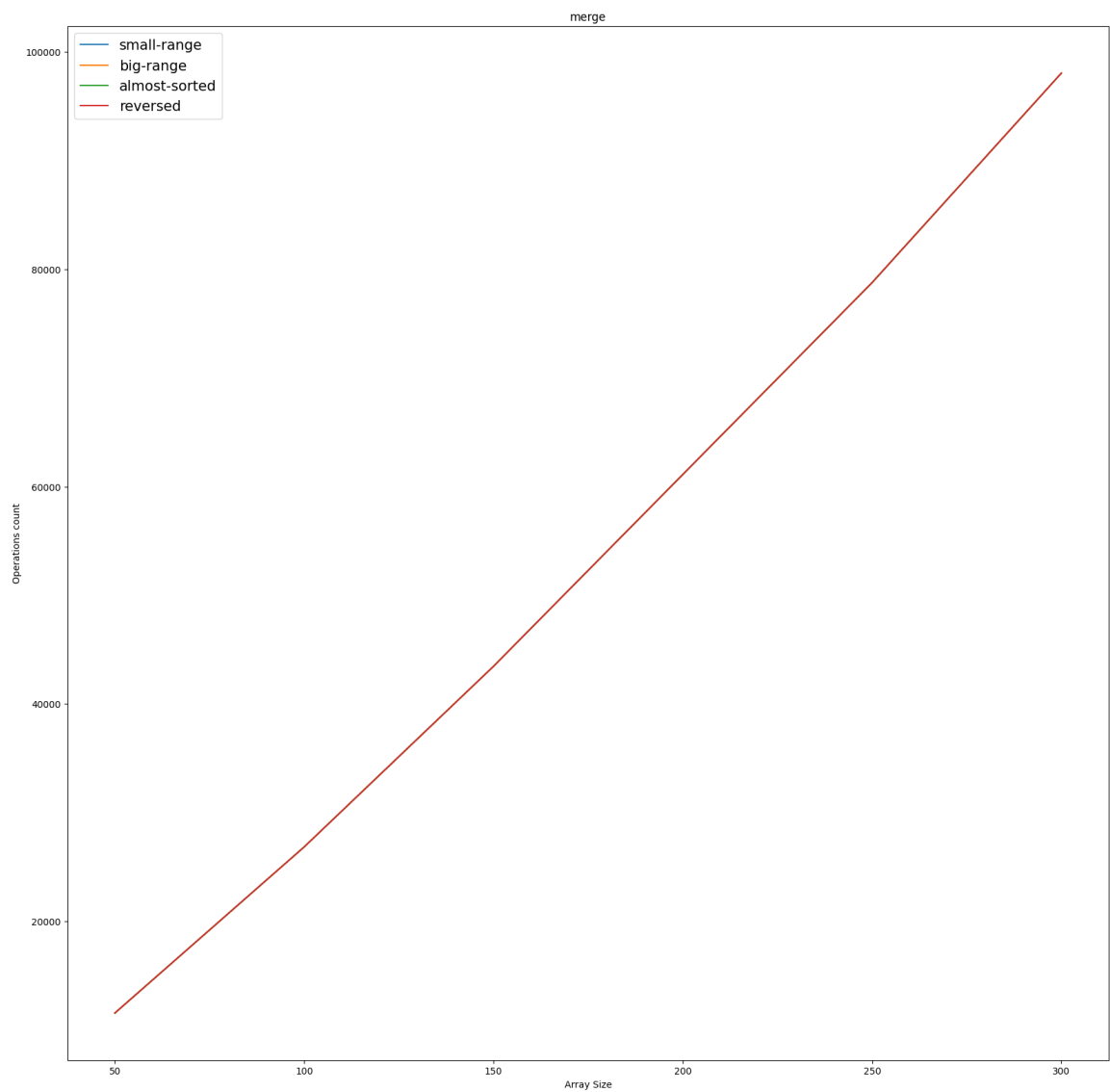




**Вывод:** тоже линия независимо от элементов

## 9. Слиянием

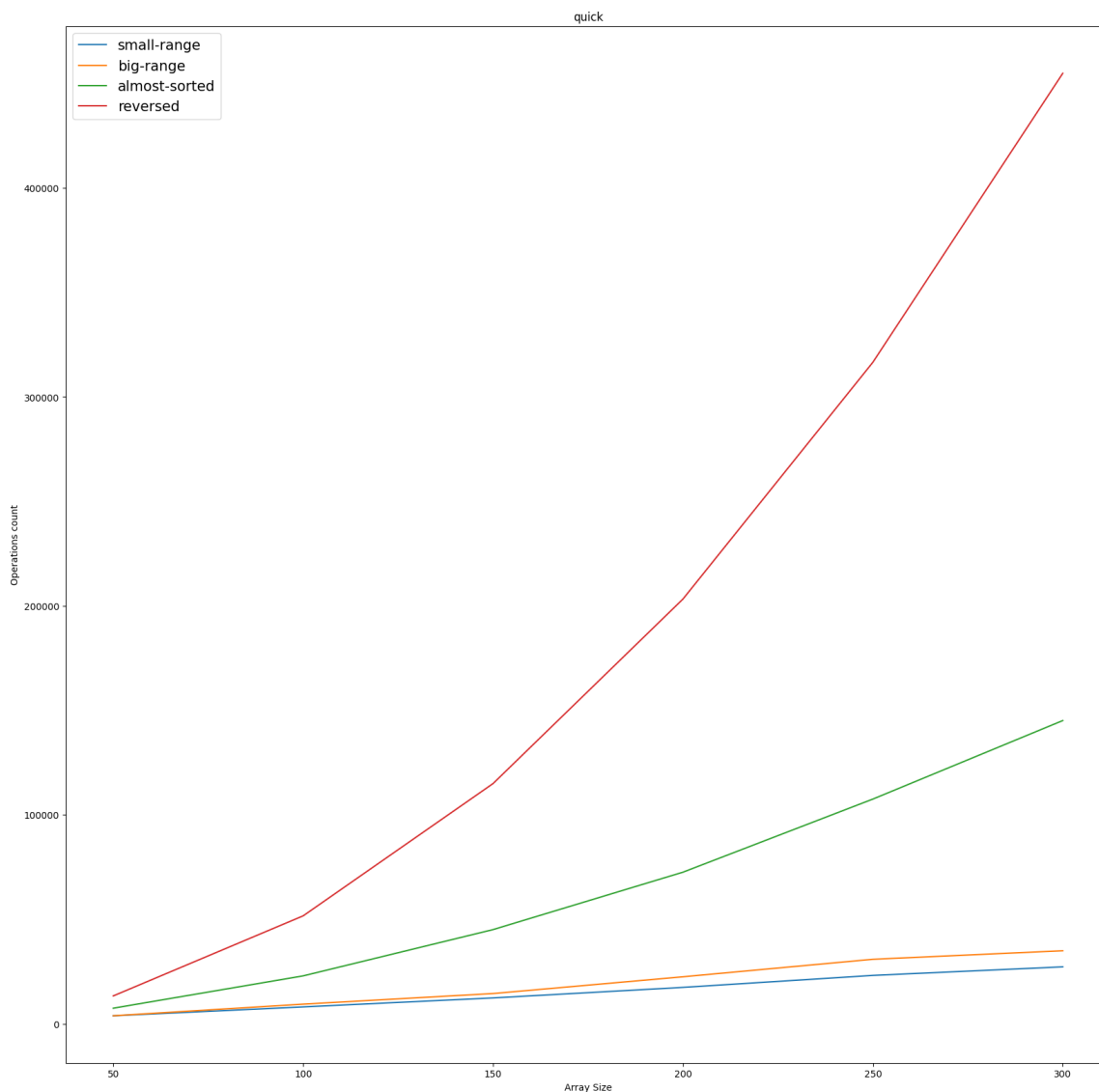
```
In [ ]: print_sort(data, sorts[8])
```



**Вывод:** так же очевидно, что любой массив будет обрабатываться одинаково

## 10. Быстрая

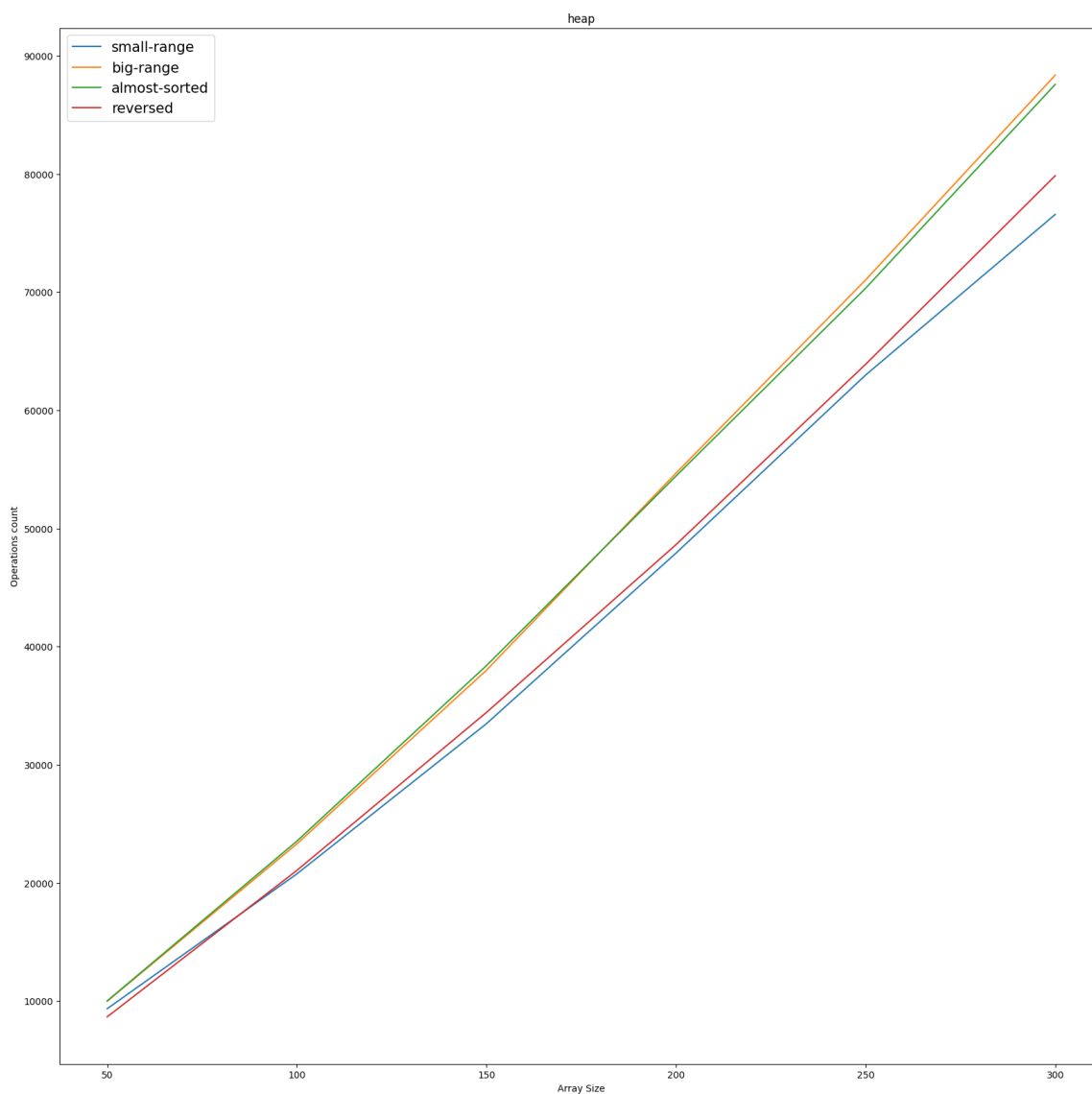
```
In [ ]: print_sort(data, sorts[9])
```



**Вывод:** снова видимо деградацию к квадрату из-за условия о выборе опорного элемента

## 11. Пирамидальная

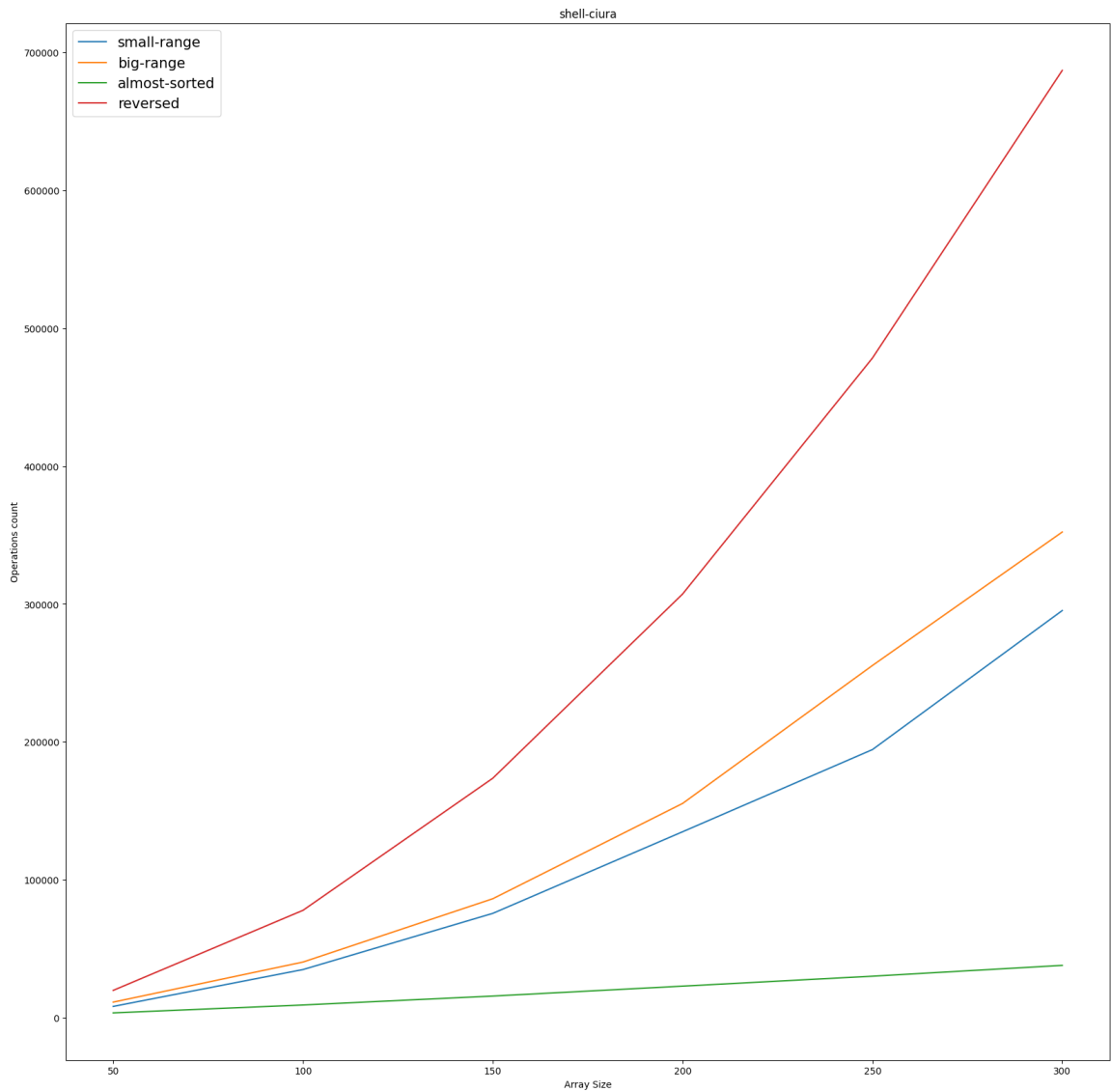
```
In [ ]: print_sort(data, sorts[10])
```



**Вывод:** больше диапазон => больше свопов => больше операций

## 12. Шелла (последовательность Циура)

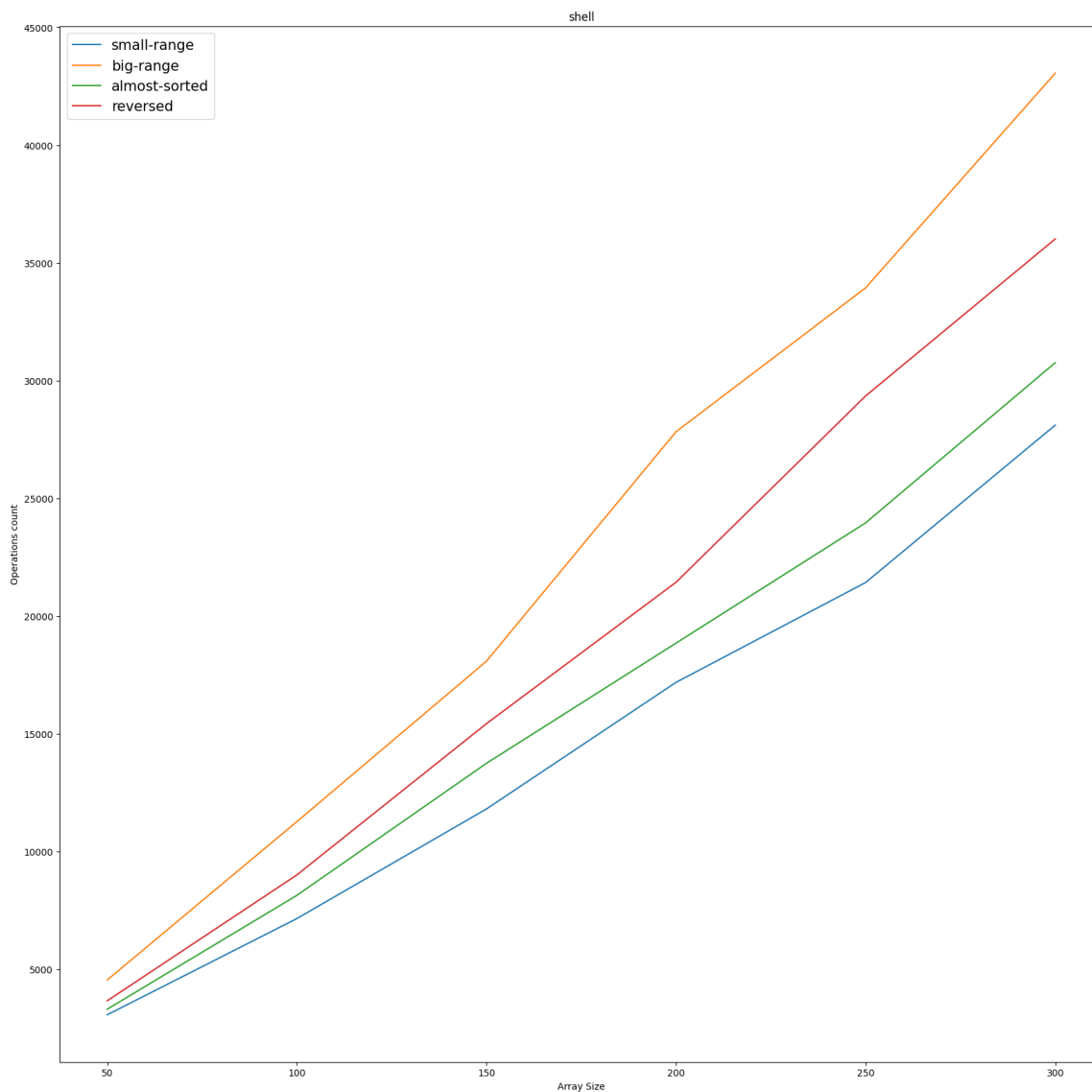
```
In [ ]: print_sort(data, sorts[11])
```



**Вывод:** видим влияние порядка элементов и их диапазона значений на количество операций

### 13. Шелла (последовательность Шелла)

```
In [ ]: print_sort(data, sorts[12])
```

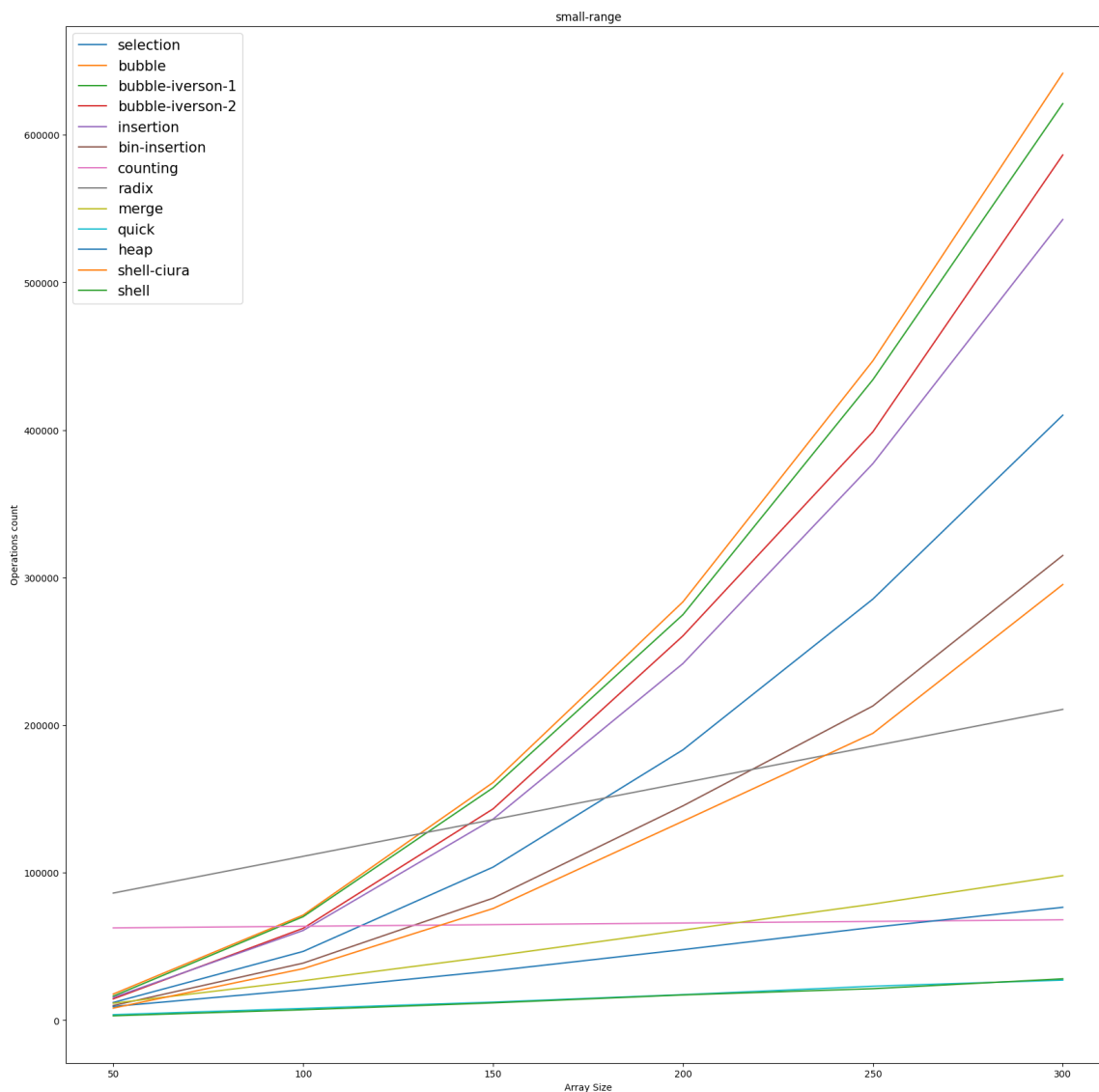


**Вывод:** схож с предыдущим пунктом, но тут результаты меньше отличаются для входных данных

## По массивам

### 1. Случайные числа от 0 до 5

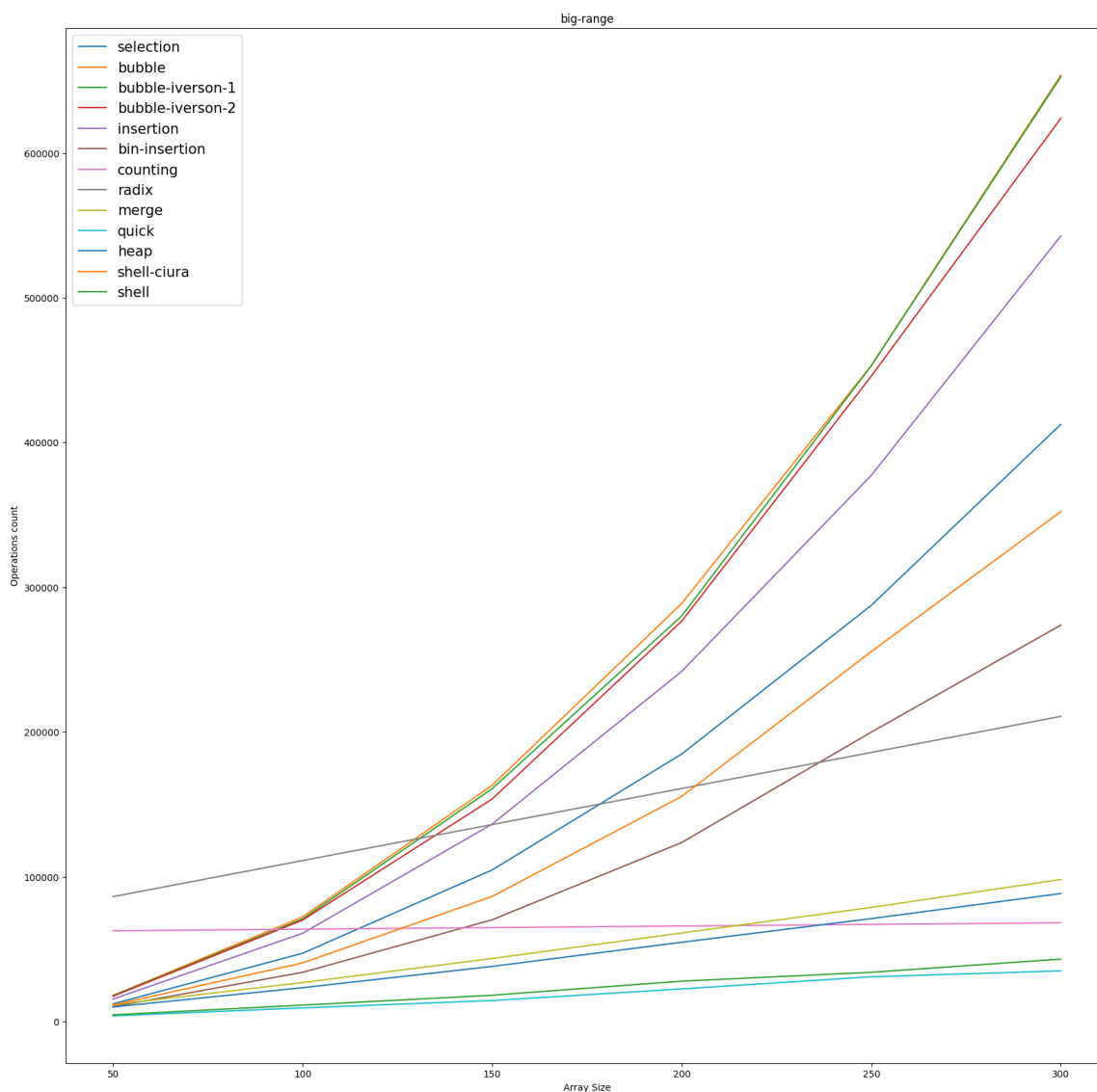
```
In [ ]: print_array(data, arrays[0])
```



**Вывод:** реализация сортировки за квадрат работает одинаково для всех видов массивов.

## 2. Случайные числа от 0 до 4000

```
In [ ]: print_array(data, arrays[1])
```

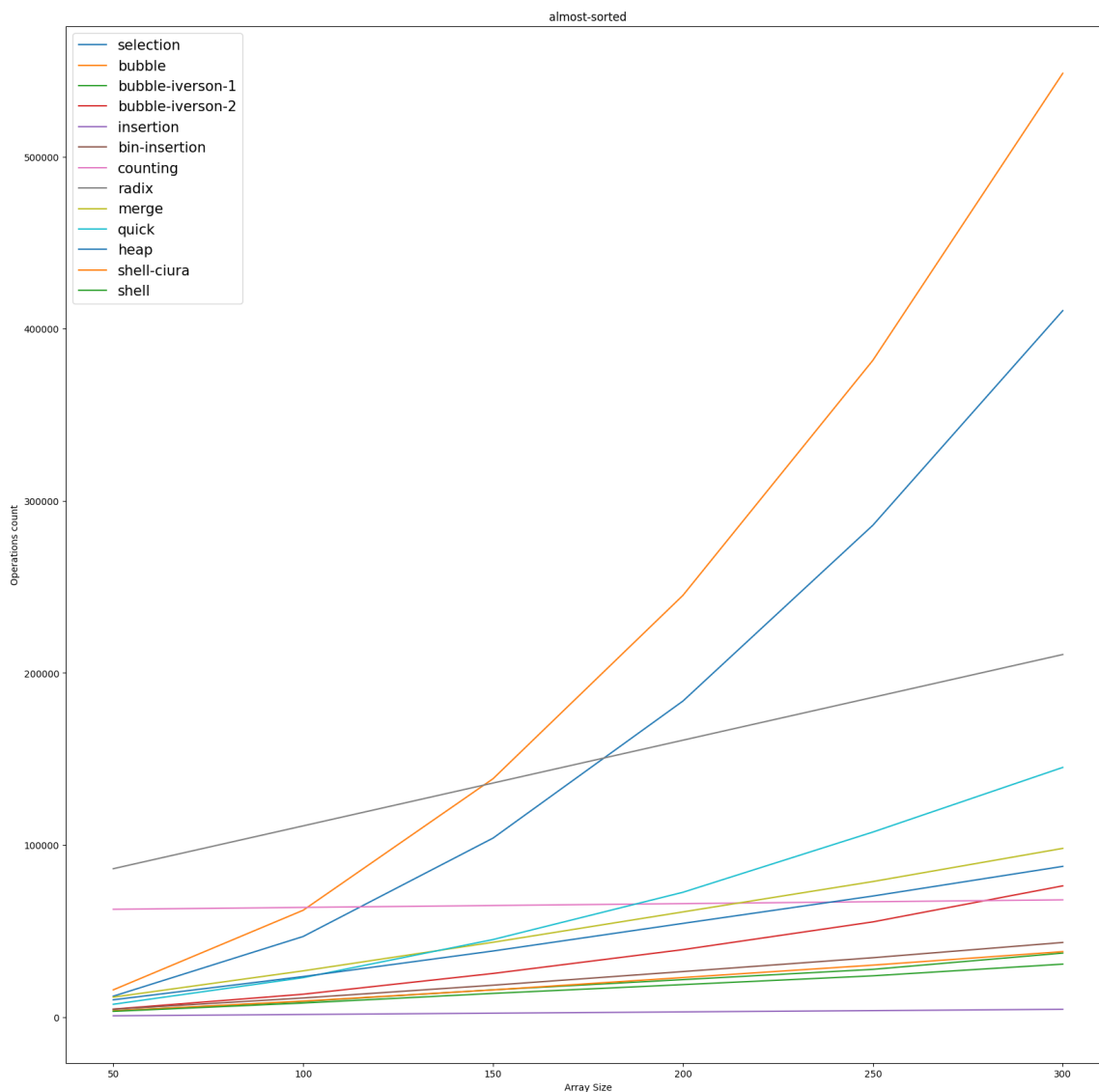


**Вывод:** реализация сортировки за квадрат работает одинаково для всех видов массивов.

### 3. Почти отсортированный массив

```
In [ ]: print_array(data, arrays[2])
```

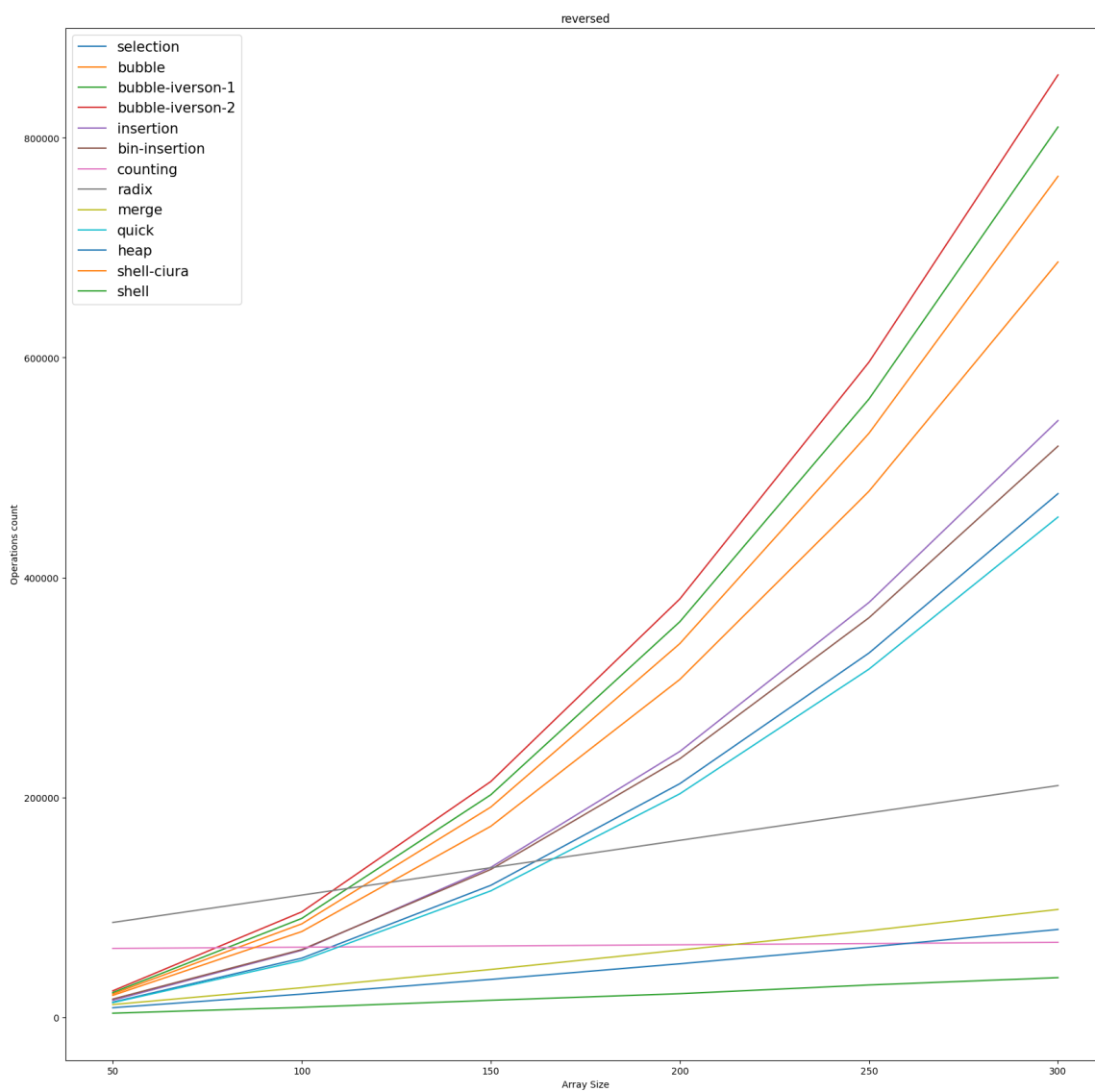




**Вывод:** отлично видна линейность `counting` и `radix`

#### 4. Отсортированный в обратном порядке массив

```
In [ ]: print_array(data, arrays[3])
```



**Вывод:** `quick` более заметно деградировал на полностью упорядоченном массиве