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-large.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})
```

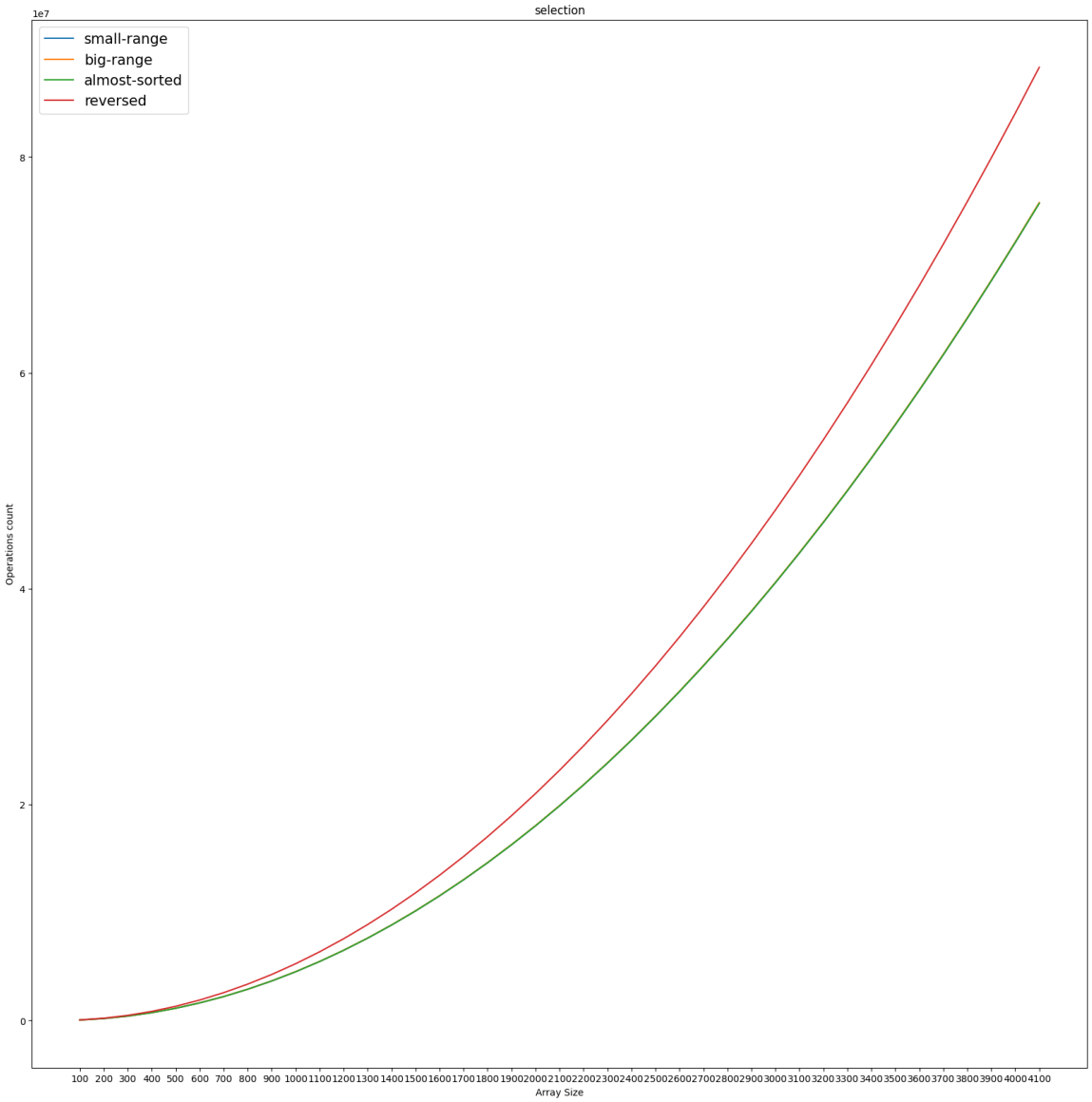
Зависимость количества элементарных операций от размера массива

для размера массива от 50 до 300, шаг 50

По сортировкам

1. Выбором

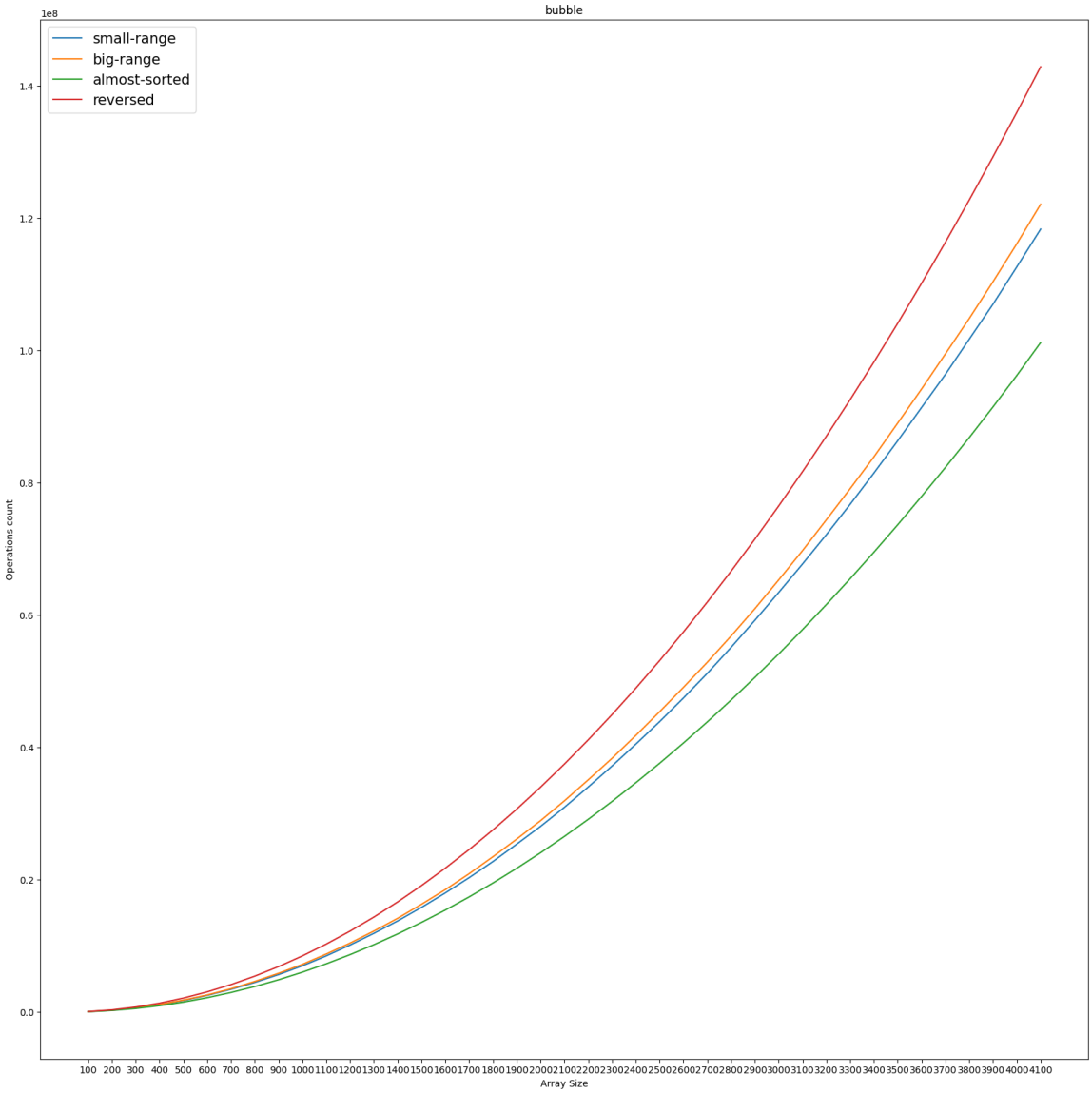
In [ ]:
```
print_sort(data, sorts[0])
```



Вывод: снова видим лишние операции для `reversed` массива

2. Пузырьком

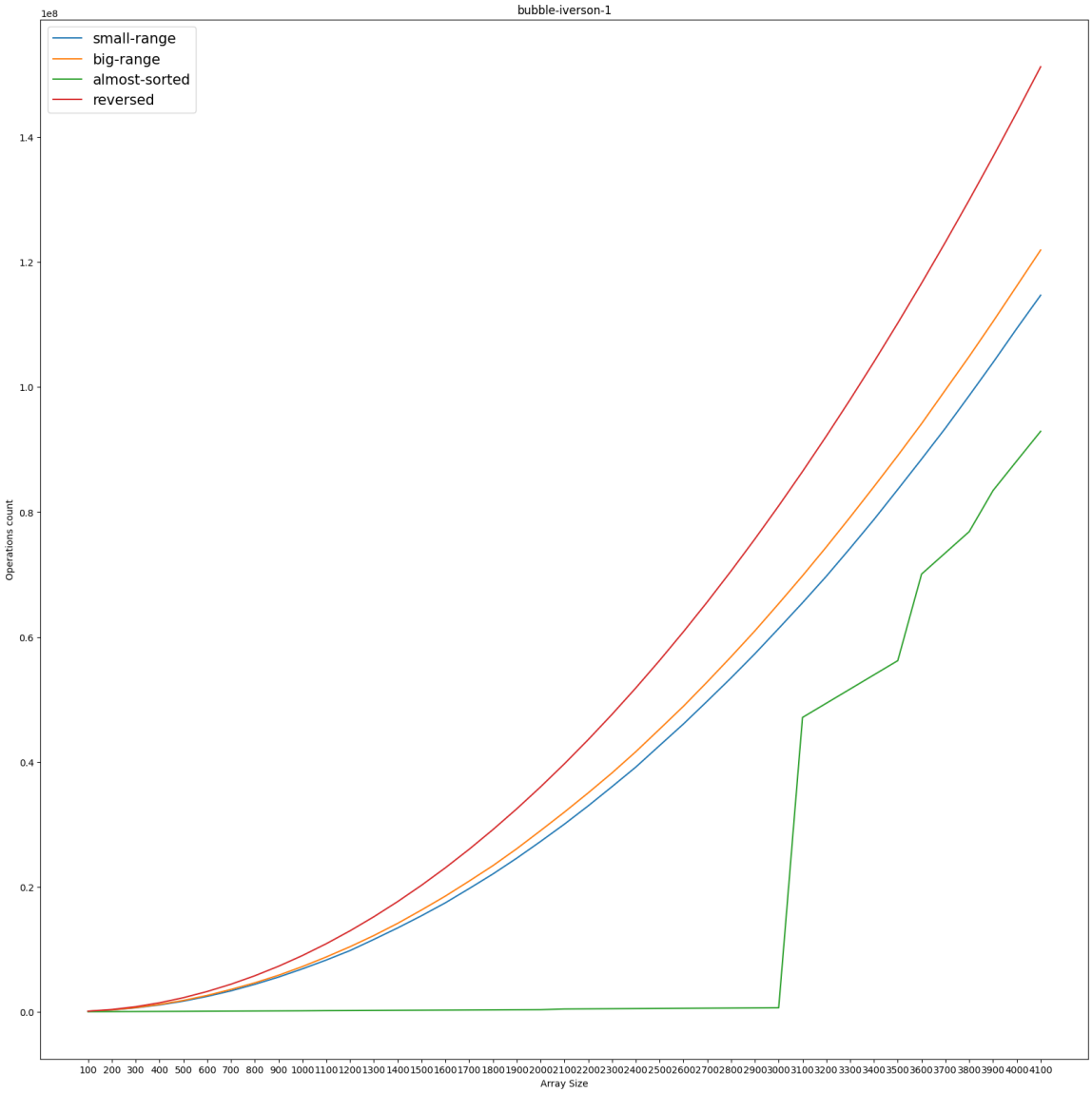
In [ ]:
```
print_sort(data, sorts[1])
```



Вывод: обычный чувствительный к порядку элементов пузырёк

3. Пузырьком с условием Айверсона 1

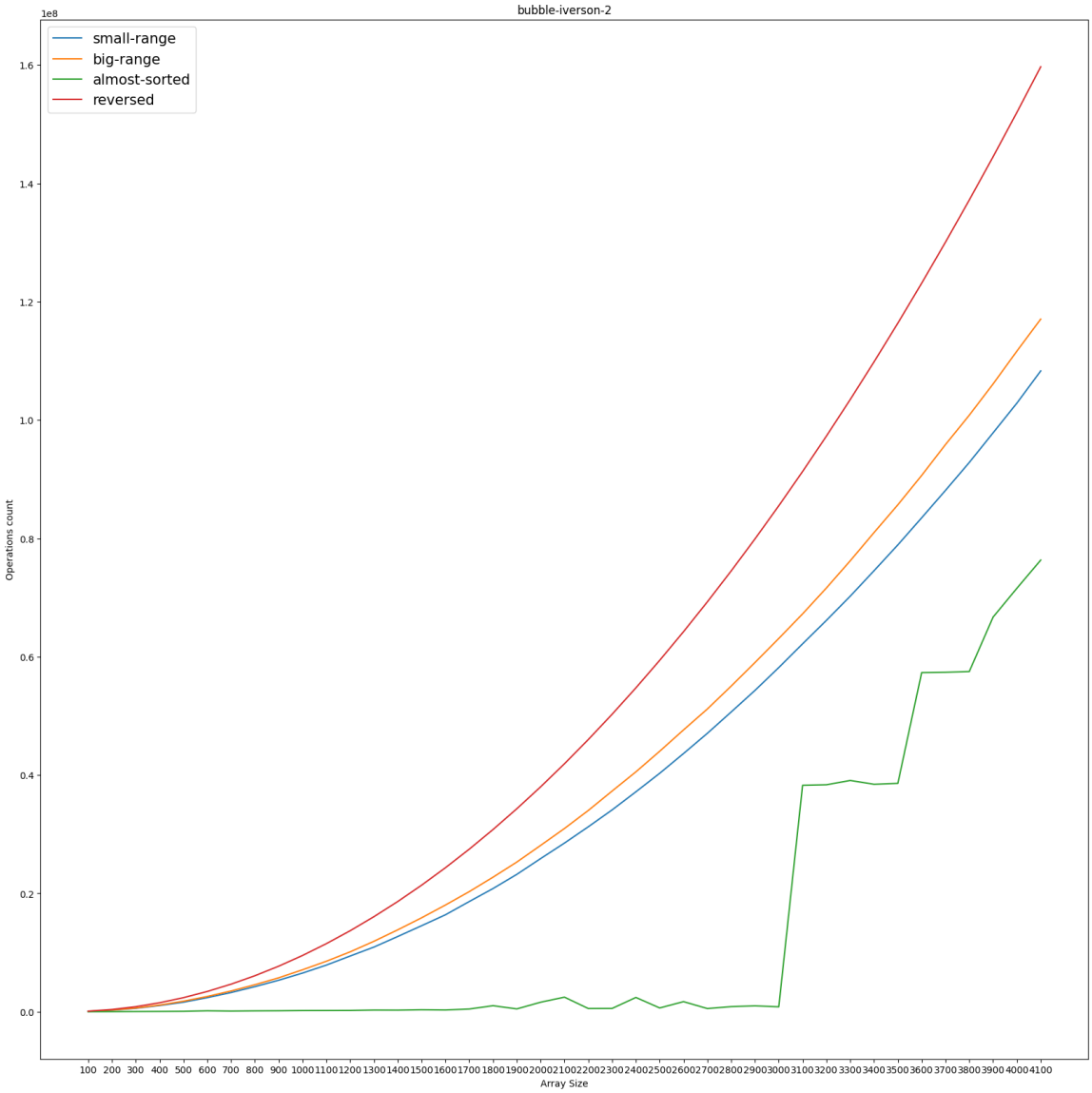
In [ ]:
```
print_sort(data, sorts[2])
```



Вывод: операций стало сравнительно меньше благодаря флагу с условием Айверсона 1!

4. Пузырьком с условием Айверсона 1 + 2

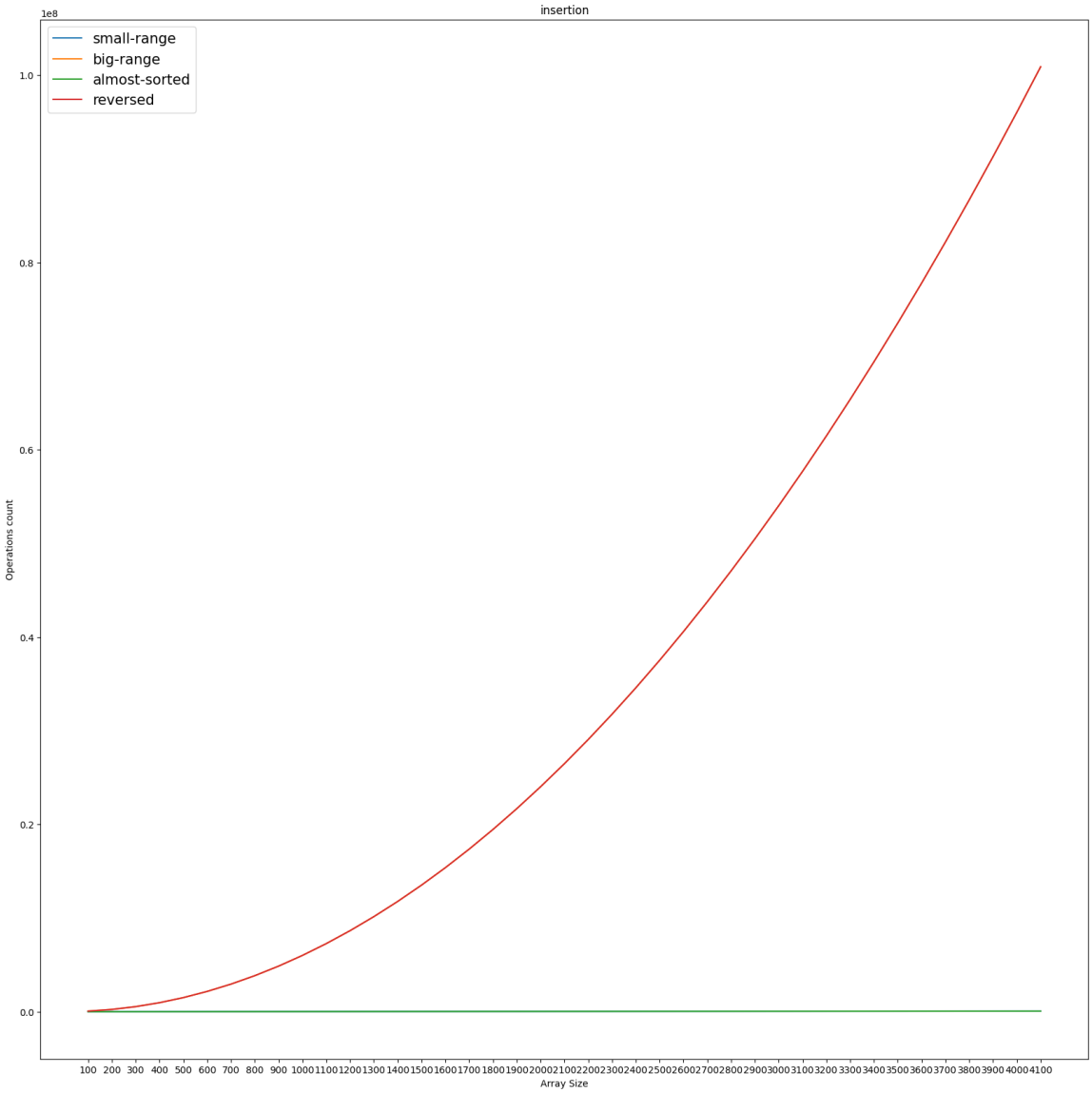
In [ ]:
```
print_sort(data, sorts[3])
```



Вывод: почти ничем не отличается от предыдущего

5. Простыми вставками

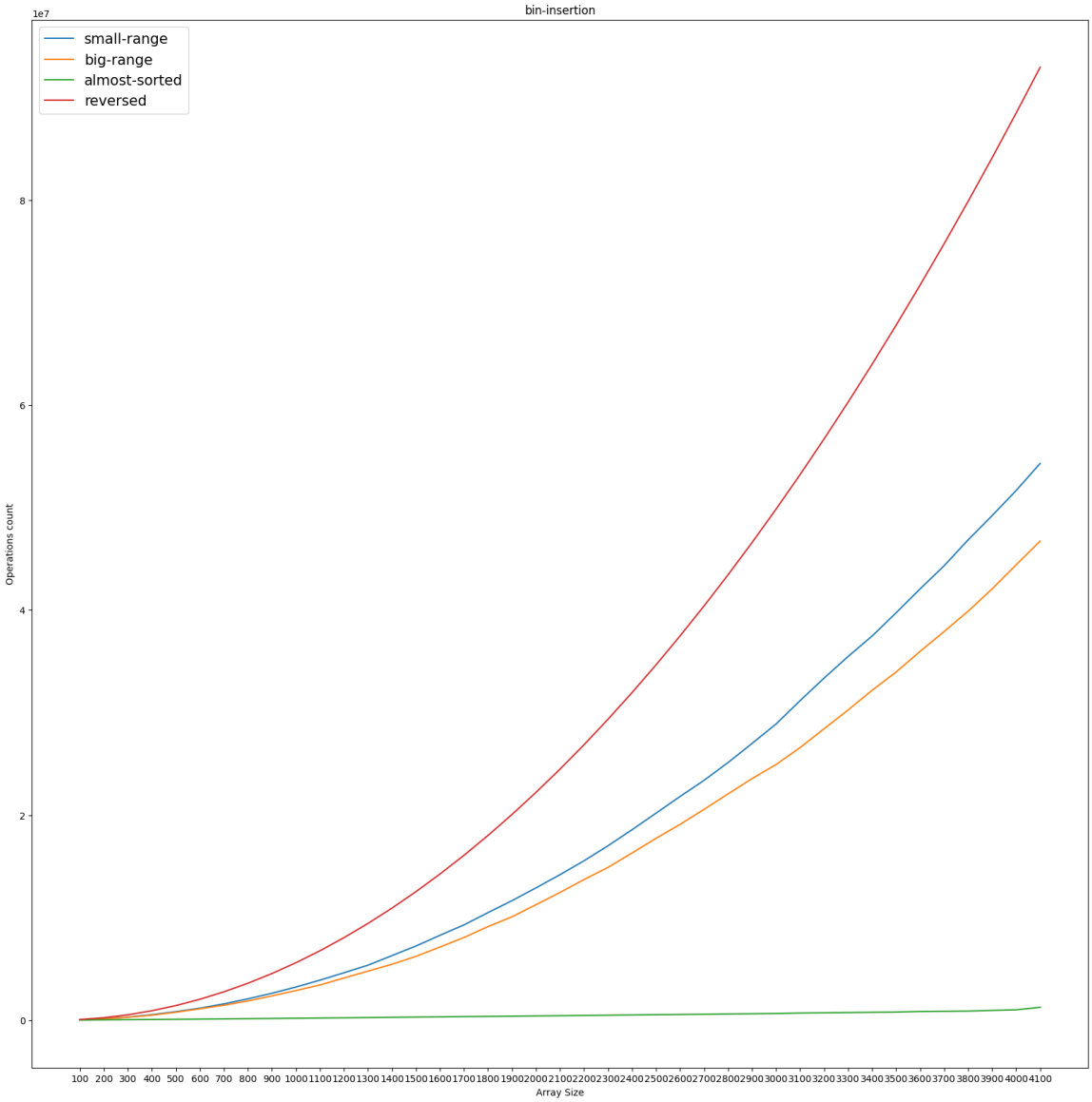
In [ ]:
```
print_sort(data, sorts[4])
```



Вывод: возможно, есть проблема со счётчиком операций (например, своп прибавляет несбалансированно большое число)

6. Бинарными вставками

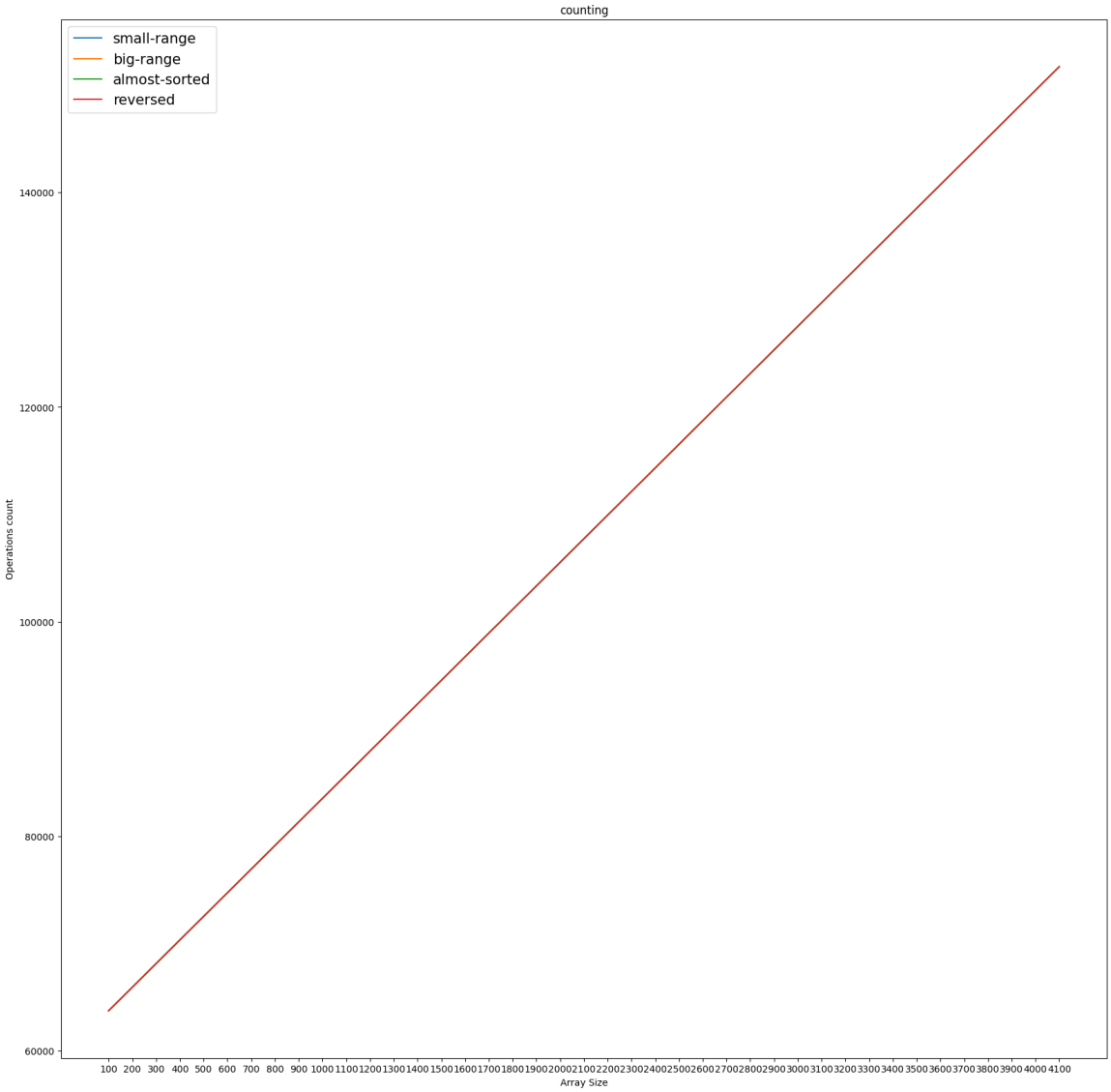
In [ ]:
```
print_sort(data, sorts[5])
```



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

7. Подсчётом

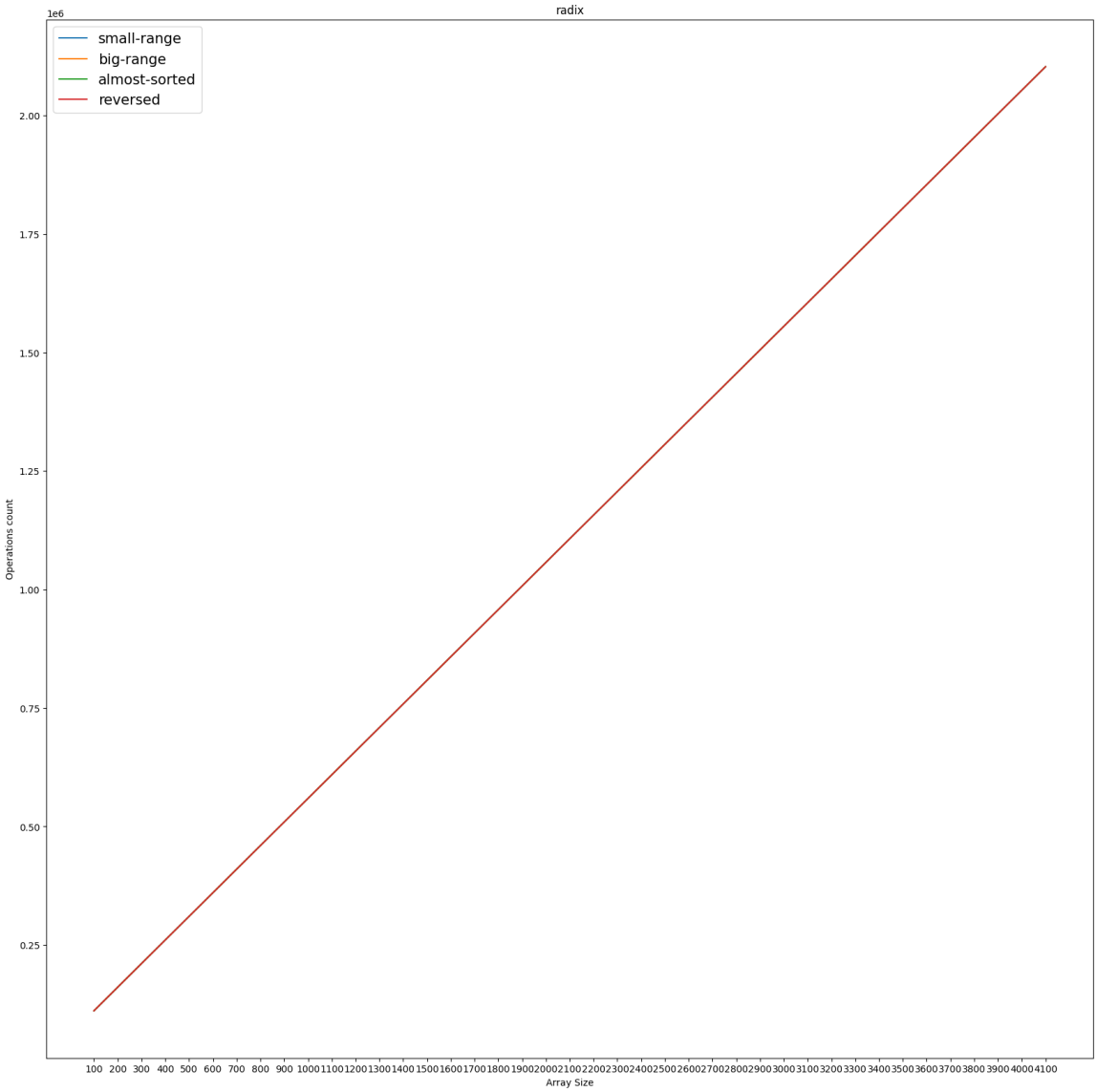
In [ ]:
```
print_sort(data, sorts[6])
```



Вывод: красивая линия)

8. Цифровая

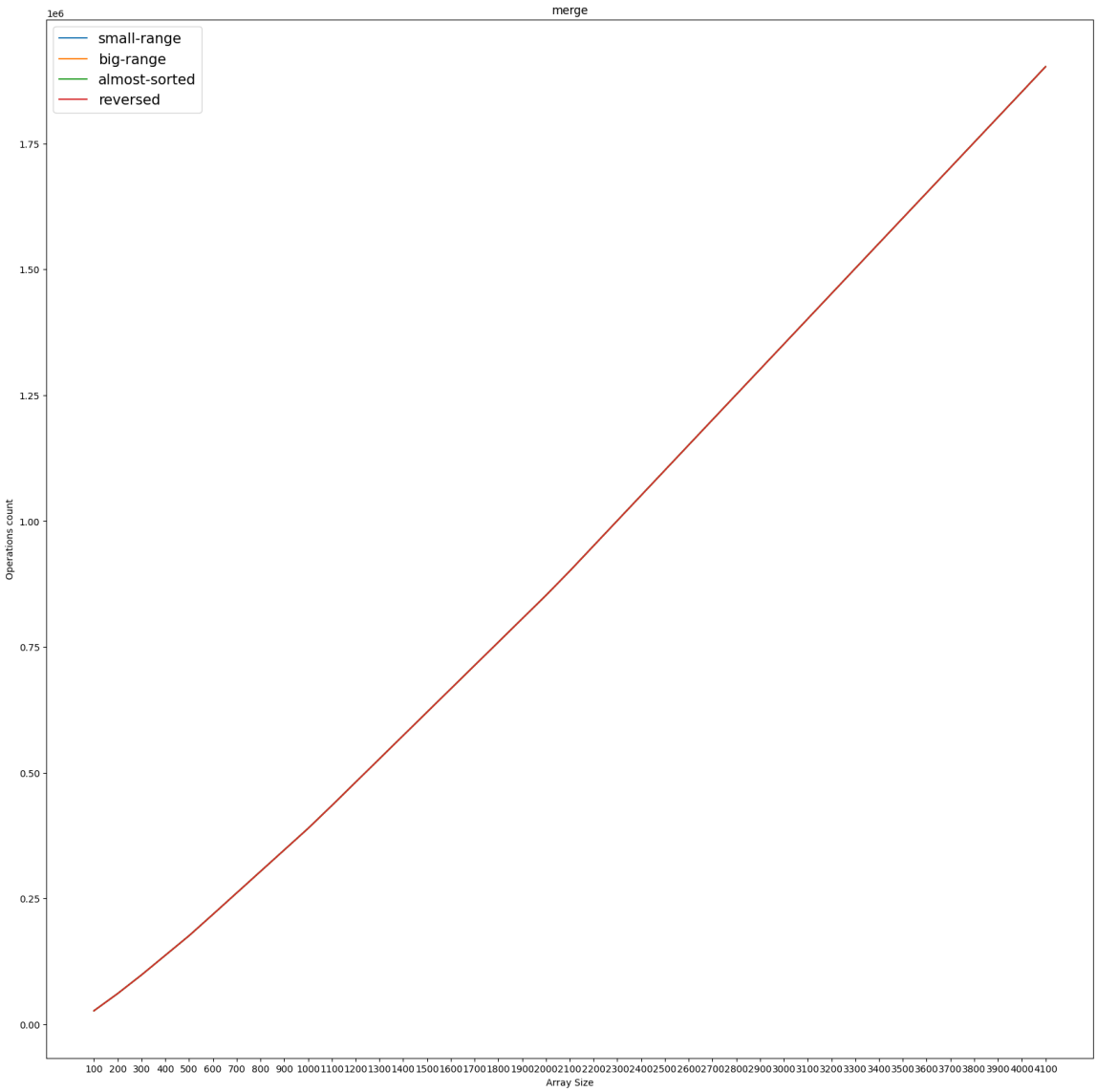
In [ ]:
```
print_sort(data, sorts[7])
```



Вывод: аналогично сортировке подсчёт подтвержает свою стабильно в отношении элементов

9. Слиянием

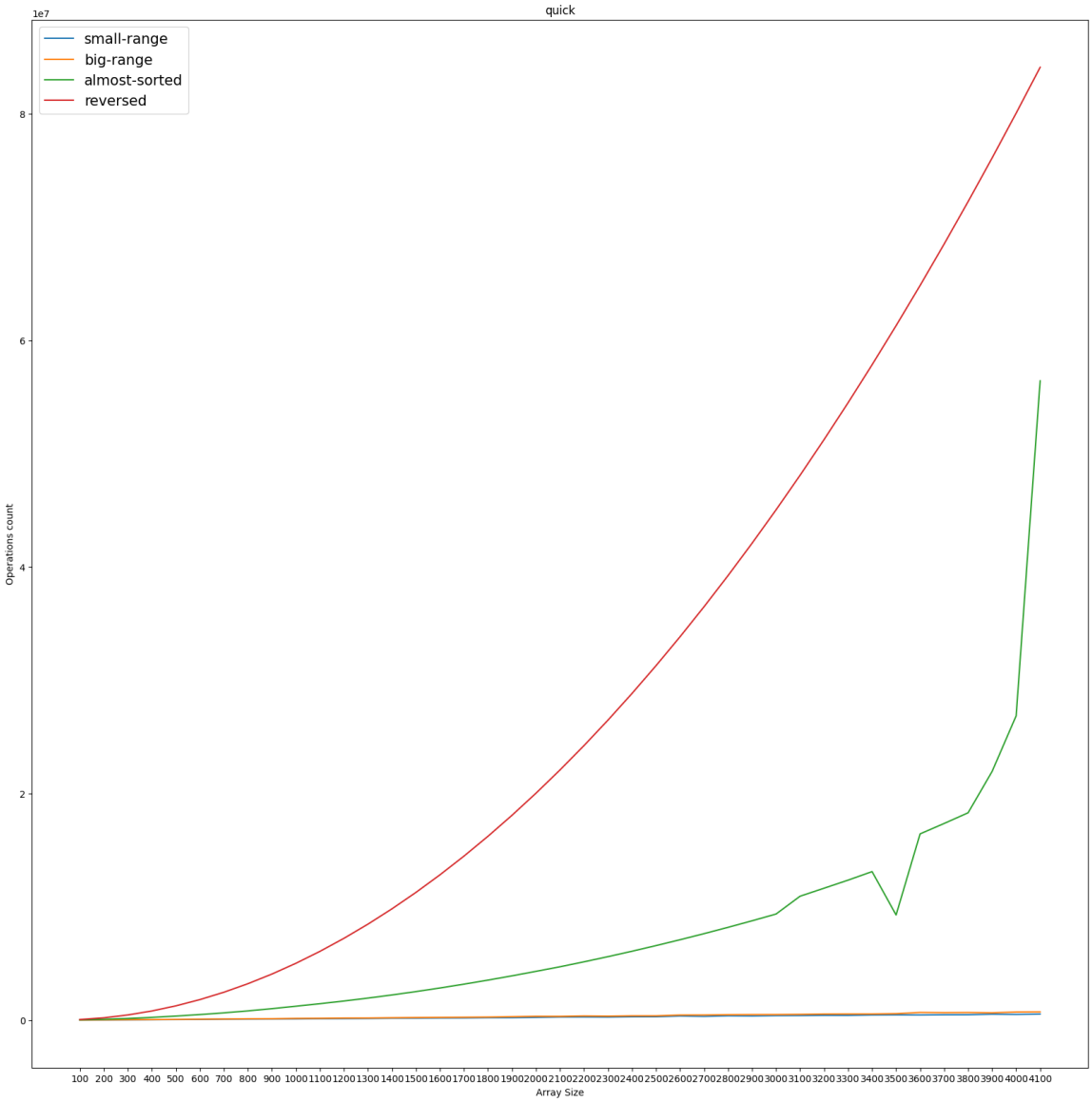
In [ ]:
```
print_sort(data, sorts[8])
```



Вывод: сортировка стабильна относительно входных элементов

10. Быстрая

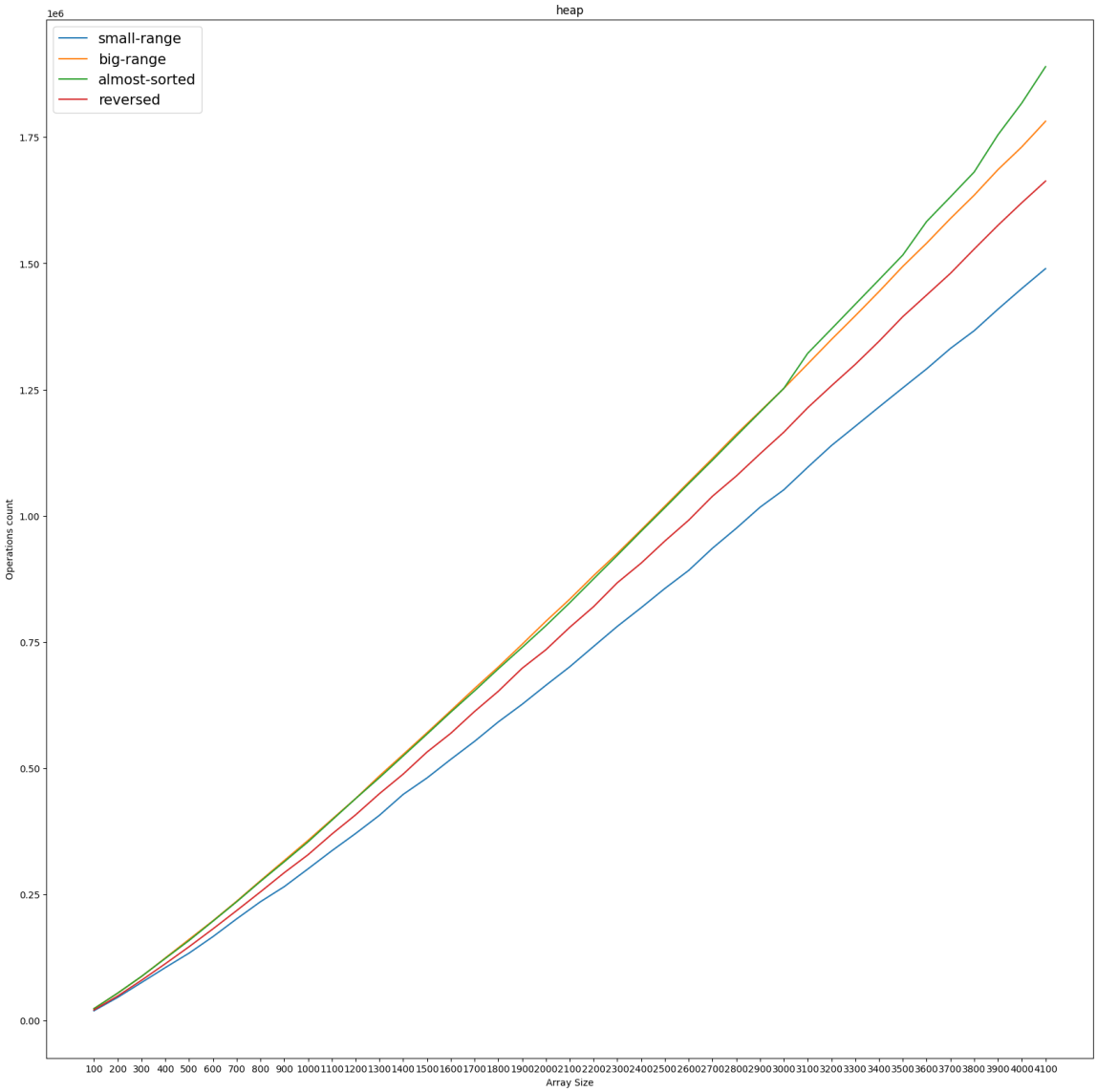
In [ ]:
```
print_sort(data, sorts[9])
```



Вывод: для частично упорядоченных последовательностей растёт количество операций ввиду из-за принципа обработки

11. Пирамидальная

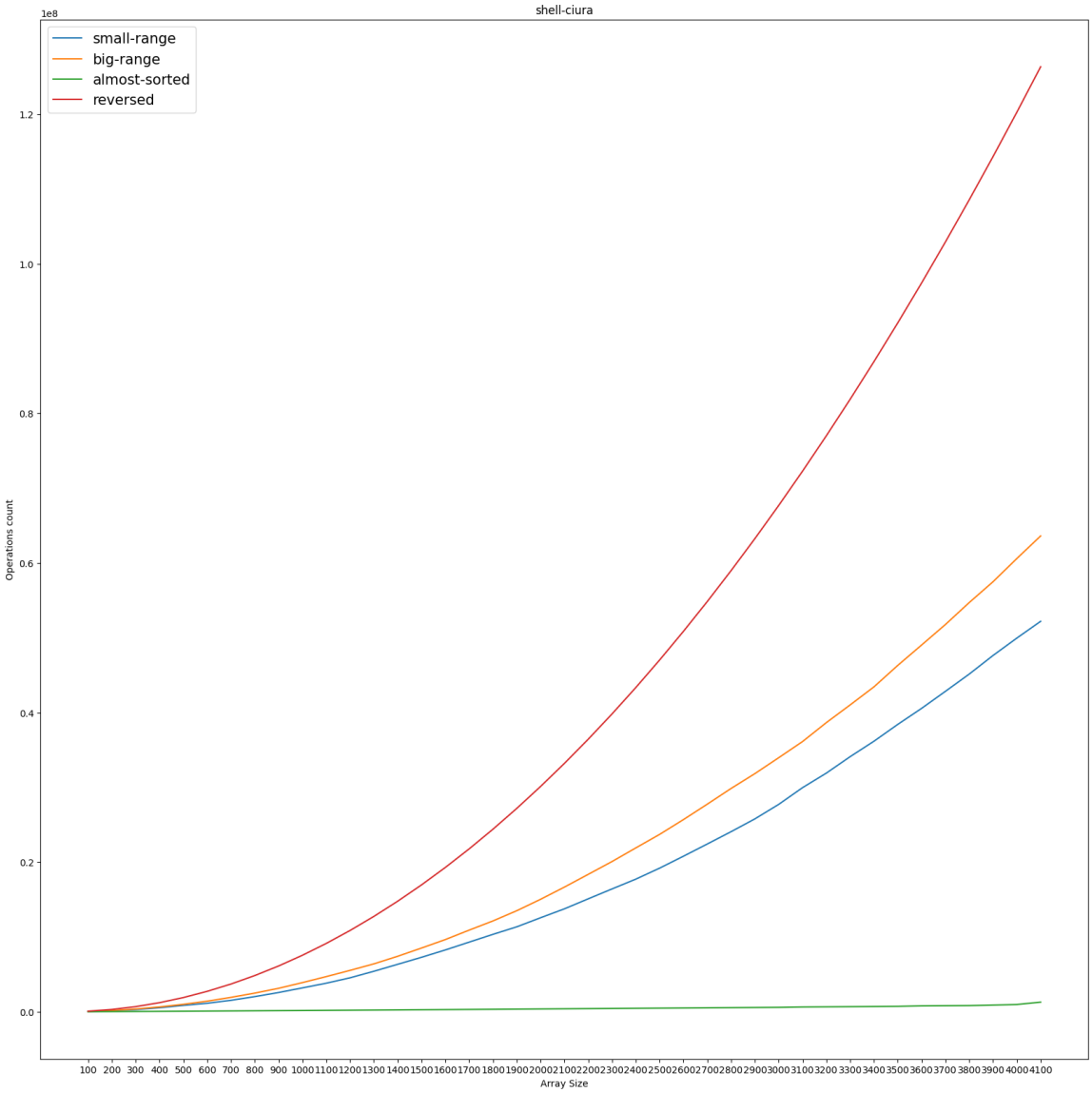
In [ ]:
```
print_sort(data, sorts[10])
```



Вывод: снова можно отметить количество влияение диапазона значений на сортировку

12. Шелла (последовательность Циура)

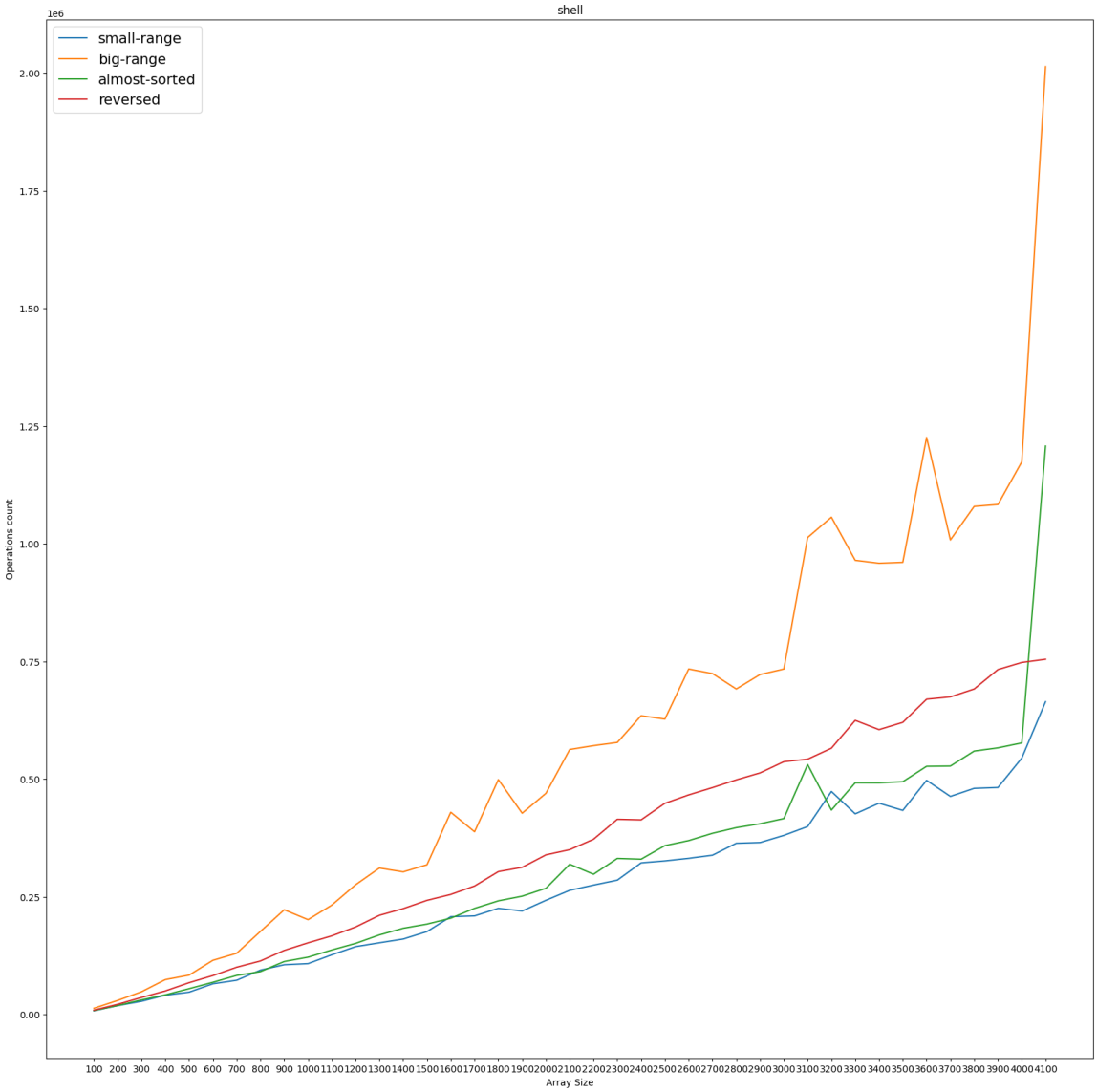
In [ ]:
```
print_sort(data, sorts[11])
```



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

13. Шелла (последовательность Шелла)

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

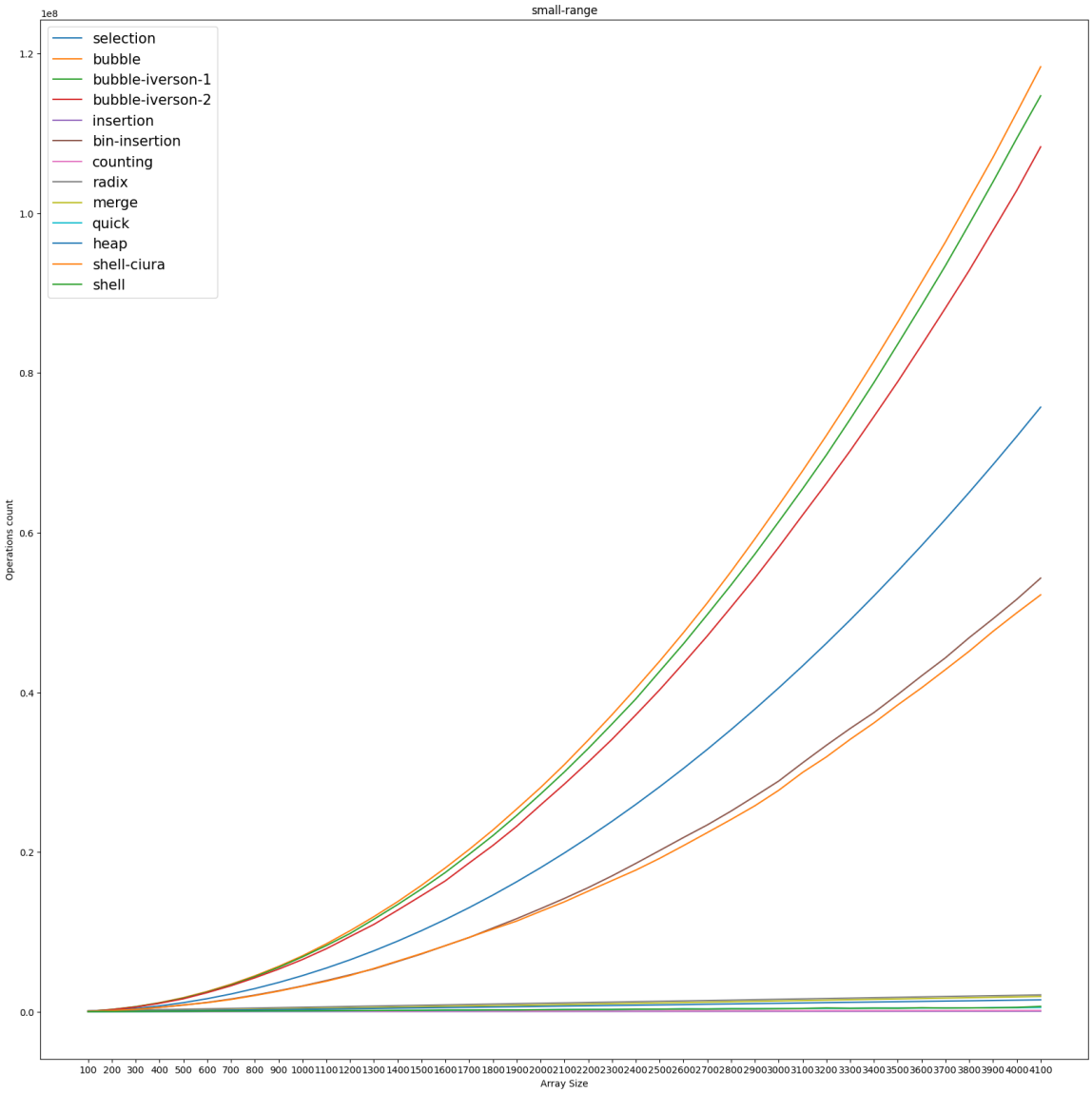


Вывод: выбросы для числа операций объяснить не могу, вероятно, это связано с генерацией входных массивов

По массивам

1. Случайные числа от 0 до 5

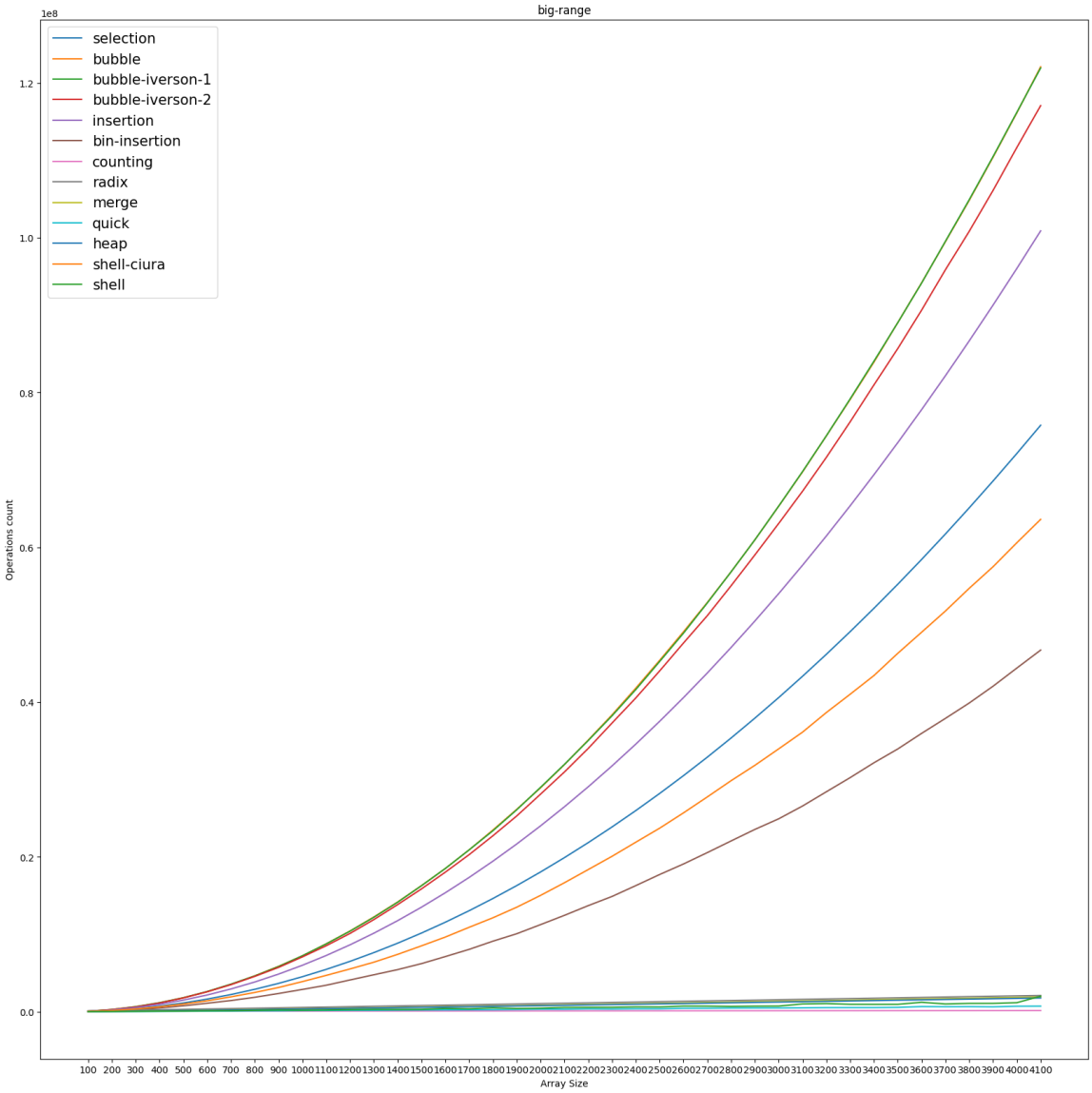
In [ ]:
```
print_array(data, arrays[0])
```



Вывод: красиво соответсвует теоретической асимптотите для сортировок

2. Случайные числа от 0 до 4000

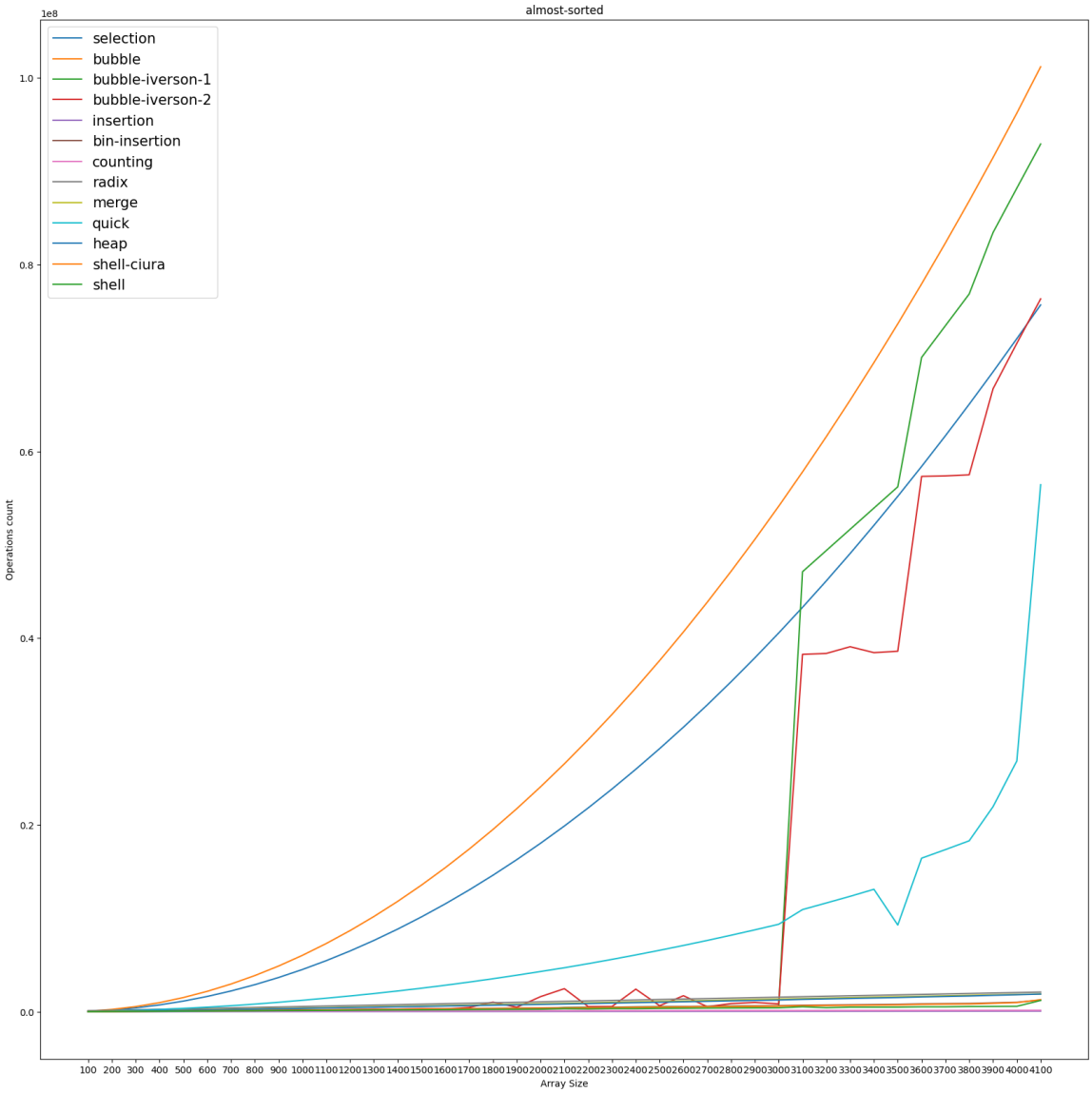
In [ ]:
```
print_array(data, arrays[1])
```



Вывод: из интересного -- снова `insertion` стал хуже по сравнению с предыдущим графиком

3. Почти отсортированный массив

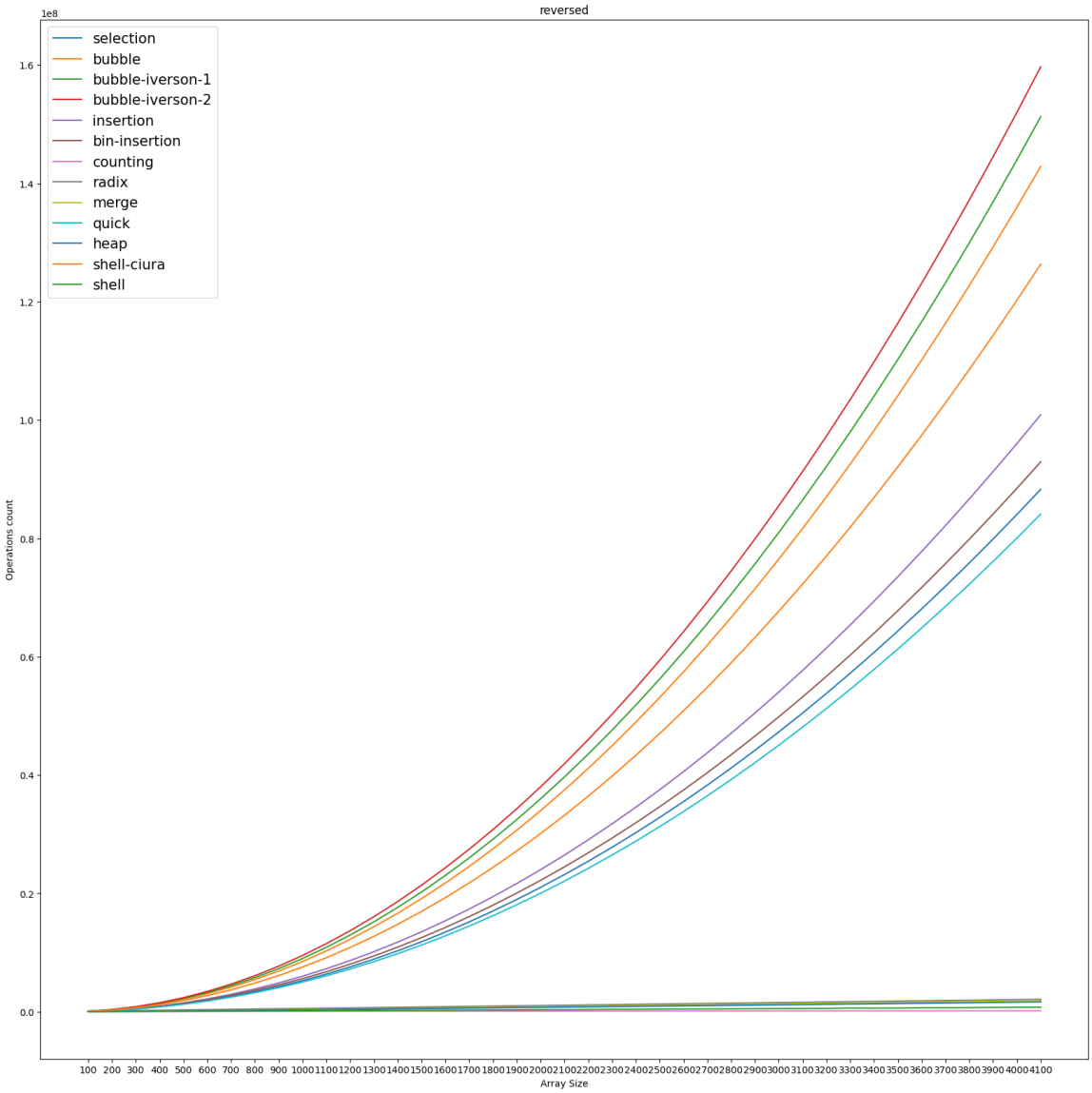
In [ ]:
```
print_array(data, arrays[2])
```



Вывод: снова деградировал `quick`, опять странности с пузырьком

4. Отсортированный в обратном порядке массив

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



Вывод: хорошо заметна разница между линейными + `nlogn` сортировками и квадратичными