**Таблица 1. Результаты эксперимента:**

| **Вид сортировки** | **Время 1000** | **Перестановки 1000** | **Время**  **10000** | **Перестановки 10000** | **Время**  **100000** | **Перестановки 100000** |
| --- | --- | --- | --- | --- | --- | --- |
| 1. Bubble sort | 28.9 ms  (0.029 сек) | ~250000 | 3162.9 ms(3.163 сек) | ~25 млн | 344428.9 ms(5.7 минут) | ~2.5 млрд |
| 2.Selection Sort | 11.9 ms(0.012 сек) | ~1000 | 1130.9 ms(1.131 сек) | ~10000 | 121462.2 ms(2 минуты) | ~100000 |
| 3.Insertion Sort | 14.4 ms(0.014 сек) | ~125000 | 1530.0 ms(1.520 сек) | 12.5 млн | 166888.1 ms(2.8 минуты) | ~1.25 млрд |
| 4.Quick Sort | 0.9 ms(0.0009 сек) | ~7000 | 12.3 ms(0.012 сек) | ~120000 | 158.1 ms(0.158 сек) | ~1.6 млн |
| 5.Merge Sort | 1.5 ms(0.015 сек) | ~8700 | 19.2 ms(0.019 сек) | ~138000 | 240.4 ms(0.240 сек) | ~1.7 млн |
| 6.Shaker Sort | 31.1 ms(0.31 сек) | ~250000 | 3169.1 ms(3.169 сек) | ~25 млн | 342384.5 ms(5.6 минут) | ~2.5 млрд |

**Таблица 2. Сравнение временной сложности:**

| **Алгоритм** | **Лучший случай** | **Средний случай** | **Худший случай** | **Память** |
| --- | --- | --- | --- | --- |
| 1.Bubble Sort | O(n) | O(n²) | O(n²) | O(1) |
| 2.Selection Sort | O(n²) | O(n²) | O(n²) | O(1) |
| 3.Insertion Sort | O(n) | O(n²) | O(n²) | O(1) |
| 4.Quick Sort | O(n log n) | O(n log n) | O(n²) | O( log n) |
| 5.Merge Sort | O(n log n) | O(n log n) | O(n log n) | O(n) |
| 6.Shaker Sort | O(n) | O(n²) | O(n²) | O(1) |

# **АНАЛИЗ РЕЗУЛЬТАТОВ И ВЫВОДЫ**

## **ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ**

На основе проведенного эксперимента можно оценить практическую сложность каждого алгоритма:

**BUBBLE SORT** - показывает худшие результаты на всех размерах данных. Время выполнения растет катастрофически быстро: с 29 мс на 1000 элементах до 344 секунд на 100000 элементах. Это классический пример O(n²) сложности.

**SELECTION SORT** - несмотря на теоретическую O(n²) сложность, на малых данных показывает себя лучше других квадратичных алгоритмов (12 мс на 1000 элементах), но на больших данных также становится непригодным (121 секунда на 100000 элементах).

**INSERTION SORT** - занимает промежуточное положение среди O(n²) алгоритмов, но все равно непрактичен для больших объемов данных.

**QUICK SORT** - демонстрирует выдающуюся производительность на всех размерах данных. С ростом размера массива в 100 раз (с 1000 до 100000 элементов) время выполнения увеличивается всего в 175 раз, что подтверждает его O(n log n) сложность.

**MERGE SORT** - показывает стабильно хорошие результаты, немного уступая Quick Sort, но сохраняя предсказуемость поведения. Также демонстрирует O(n log n) сложность.

**SHAKER SORT** - как и Bubble Sort, имеет O(n²) сложность и показывает схожие с ним результаты.

## **ВЫВОДЫ ПО РАЗМЕРНОСТЯМ МАССИВА**

### **ДЛЯ 1000 ЭЛЕМЕНТОВ:**

Лучший алгоритм: QUICK SORT (0.9 мс)

Все алгоритмы работают достаточно быстро, но уже здесь видно преимущество эффективных алгоритмов. Quick Sort в 32 раза быстрее Bubble Sort и в 13 раз быстрее Selection Sort. На этом размере данных еще можно использовать любые алгоритмы, но Quick Sort и Merge Sort явно предпочтительнее.

### **ДЛЯ 10000 ЭЛЕМЕНТОВ:**

Лучший алгоритм: QUICK SORT (12.3 мс)

Здесь разница становится dramatic. Quick Sort работает в 257 раз быстрее Bubble Sort и в 92 раза быстрее Selection Sort. O(n²) алгоритмы уже занимают секунды, в то время как O(n log n) алгоритмы - десятки миллисекунд. На этом размере использование квадратичных алгоритмов становится нецелесообразным.

### **ДЛЯ 100000 ЭЛЕМЕНТОВ:**

Лучший алгоритм: QUICK SORT (158.1 мс)

Разница в производительности достигает тысяч раз. Quick Sort завершает сортировку за доли секунды, в то время как Bubble Sort требует почти 6 минут. Это наглядно демонстрирует, почему O(n²) алгоритмы непригодны для больших данных.

## ОБЩИЙ ВЫВОД

## На основе эксперимента можно сделать следующие выводы:

1. Quick Sort является оптимальным выбором для сортировки данных любого размера, показывая наилучшую производительность на всех тестах.
2. Теоретическая оценка сложности полностью подтверждается на практике: разница между O(n²) и O(n log n) алгоритмами достигает тысяч раз на больших данных.
3. Для массивов до 1000 элементов можно использовать любые алгоритмы, но эффективные алгоритмы уже показывают преимущество.
4. Для массивов от 10000 элементов следует использовать только O(n log n) алгоритмы.
5. Merge Sort является надежной альтернативой Quick Sort, особенно когда важна стабильность алгоритма.