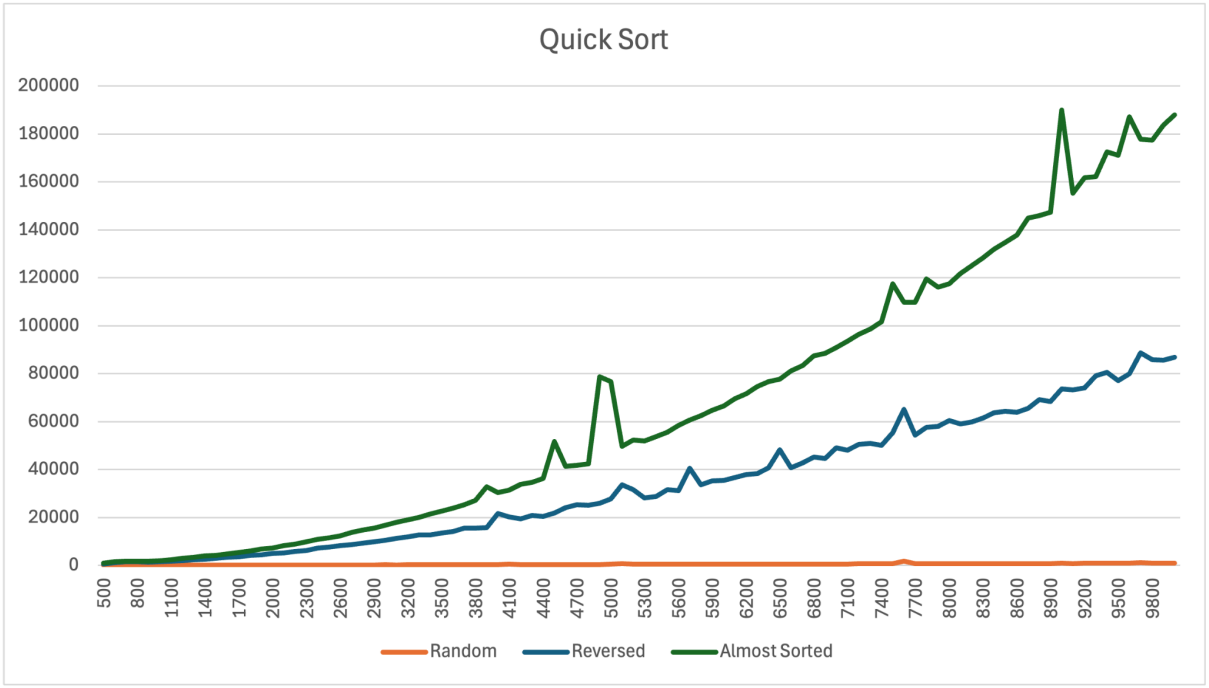
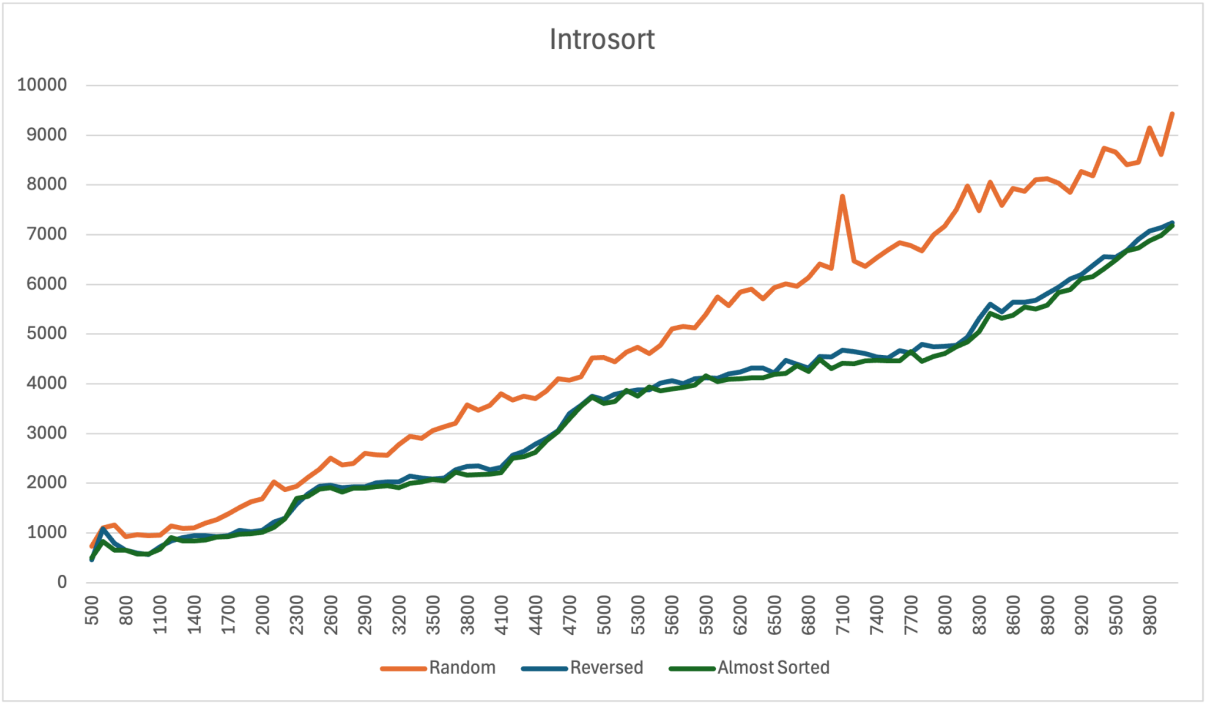
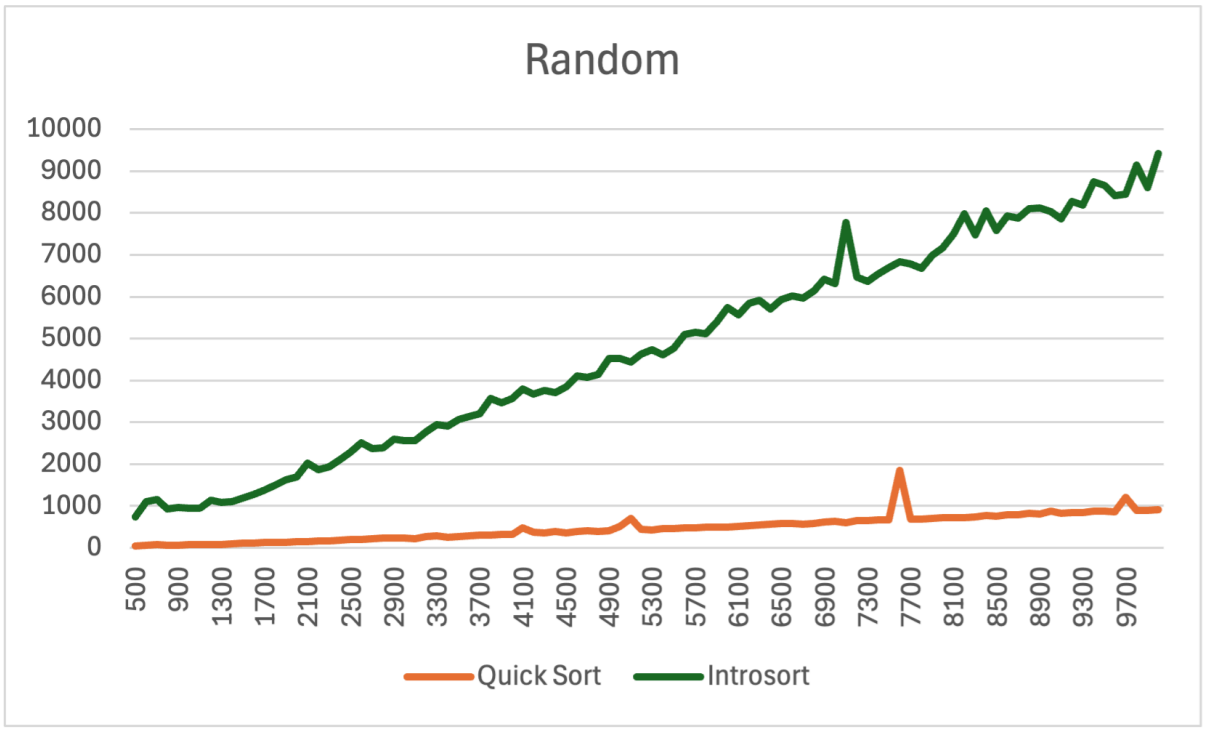
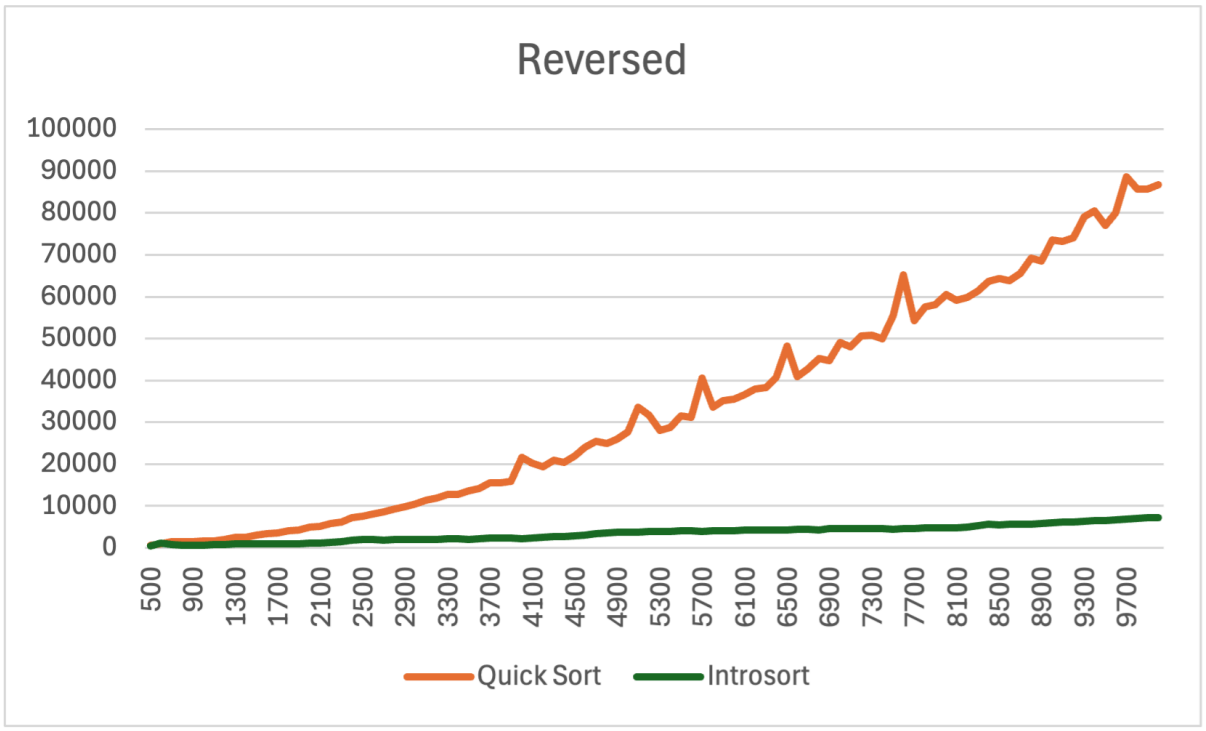
A3.
ID посылки: 293159847
Ссылка на гитхаб с кодом: https://github.com/vilina4kaa/Algorithms.git
Графики (вертикальная ось - время в мс, горизонтальная - size массива):
Отдельные для каждой функции:

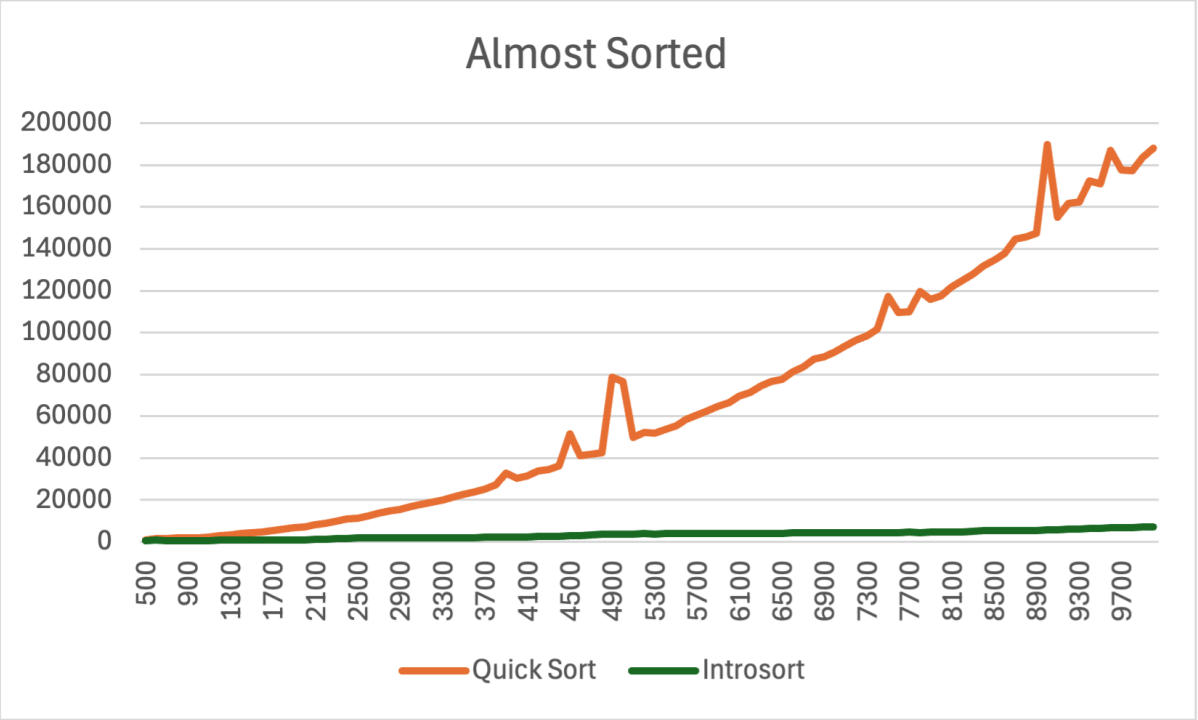



Сводные для каждого набора данных:


Random
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
500
900
1300
1700
2100
2500
2900
3300
3700
4100
4500
4900
5300
5700
6100
6500
6900
7300
7700
8100
8500
8900
9300
9700
Quick Sort
Introsort


Reversed
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
500
900
1300
1700
2100
2500
2900
3300
3700
4100
4500
4900
5300
5700
6100
6500
6900
7300
7700
8100
8500
8900
9300
9700
Quick Sort
Introsort




**Выводы:**
Для анализа скорости работы quick sort и гибридной introsort я написала программу и построила соответствующие графики.
В моем коде я написала функции для сортировки: quickSort(), heapSort() + heapify() и insertionSort() . А для добавления heap sort я добавила дополнительный параметр в аргументы функции - depthLimit. Он показывает, на какой длине массивы нужно перестать делать quicksort и сделать вместо этого сортировку кучей. Если осталось меньше 16 элементов для сортировки, алгоритм переключается на insertionSort. Эти шаги зафиксированы в функции introsort.

Для генерации рандомного массива длиной до 10000 я использовала функцию generateRandomArray(), в которой использую рандомный алгоритм mt19937. Для генерации перевернутого и почти отсортированных массивов я использую функции generateReversedArray() и generateAlmostSortedArray().

По результатам замеров и построенным графикам можно сделать следующие выводы:

1. Quick Sort лучше всего показал себя на рандомных массивах, на порядок хуже на перевернутых и хуже всего (снова на порядок) на почти отсортированных массивах. Это может быть связано с тем, что:
    - На случайных данных QuickSort работает быстро (в среднем за O(nlogn)), так как он делит массив на части близкие по размеру, минимизируя глубину рекурсии
    - При плохом выборе pivot для реверсированных данных QuickSort может вести себя не очень хорошо

    - Почти отсортированный массив содержит много локальных инверсий, которые QuickSort обрабатывает относительно неэффективно, так как может делать больше ненужных перестановок.
2. Introsort почти одинаково хорошо сработал для почти отсортированных и перевернутых массивов, однако для рандомных чуть хуже. Это может быть связано с частым переключением на heapSort.
3. Только для random массивов introsort может уступать Quick Sort(см пункт 1), однако в общем случае чаще будет работать быстрее. Для revrsed и almost sorted очень заметна разница между introsort и quick sort, что указывает на эффективность переключения между функциями в introsort (работает почти за линию).