Лабораторная работа №1

По курсу «Языки программирования и методы программирования»

Цель: написать алгоритмы сортировки и проанализировать их работу на различных данных.

Были написаны следующие сортировки:
1) BubbleSort (пузырьковая сортировка)

2) ShakerSort (шейкерная сортировка)

3) QuickSort (быстрая сортировка)

4) ShellSort (сортировка Шелла)

5) InsertSort (сортировка вставками)

6) MergeSort (сортировка слиянием)

Также были реализованы структуры данных – динамический массив и связный список. Кроме того, был реализован абстрактный класс ISorter, с помощью наследников которого выполнялись сортировки.

Тестирование

Было проведено тестирование всех алгоритмов. В качестве данных для сортировки использовались динамические массивы и связные списки с различным наполнением:

1) заполненные случайными данными

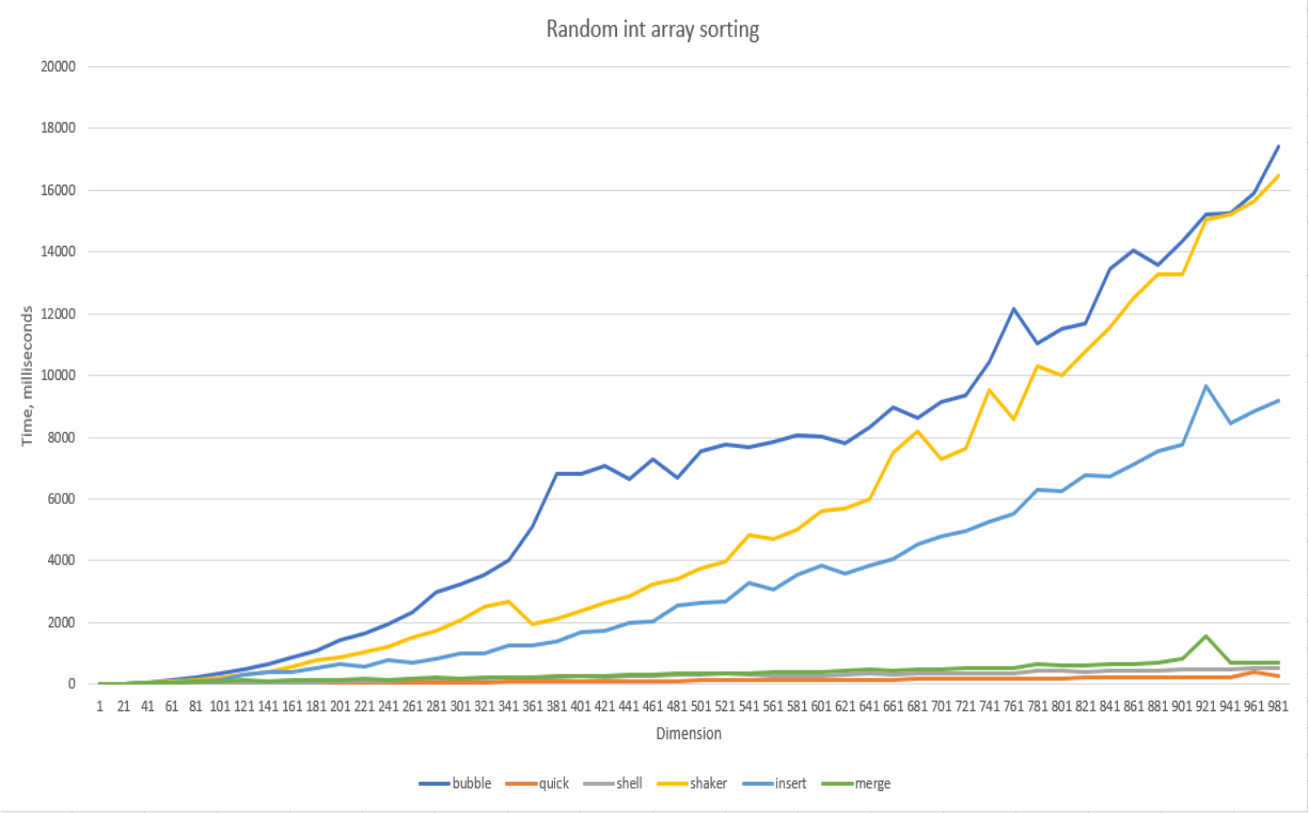
2) отсортированные

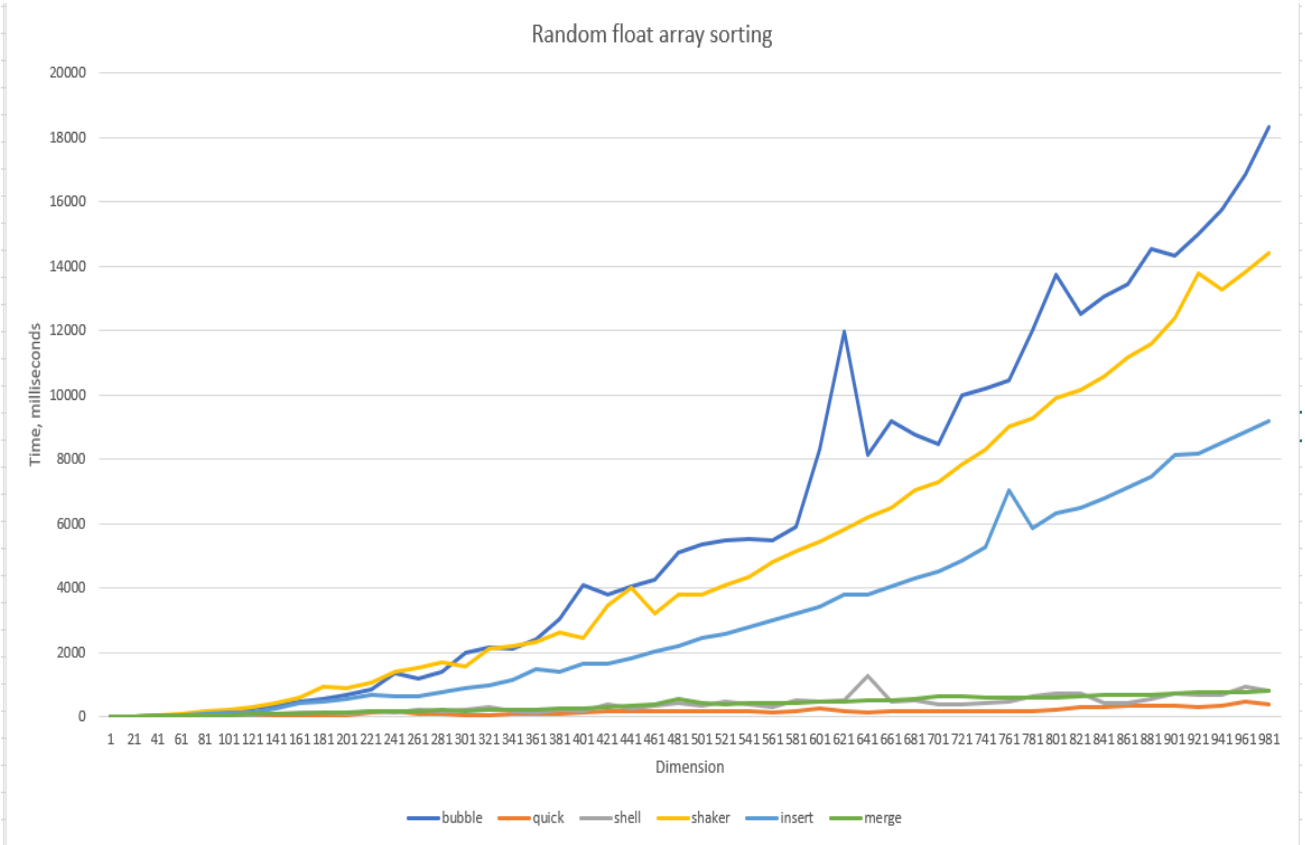
3) отсортированные в обратном порядке

Было проведено тестирование алгоритмов сортировки на массивах и списках с количеством элементов от 1 до 1000, также по полученным данным были построены графики зависимости времени выполнения сортировки от количества элементов. По горизонтальной оси было отсчитывалось количество элементов, по вертикальной – время в микросекундах.

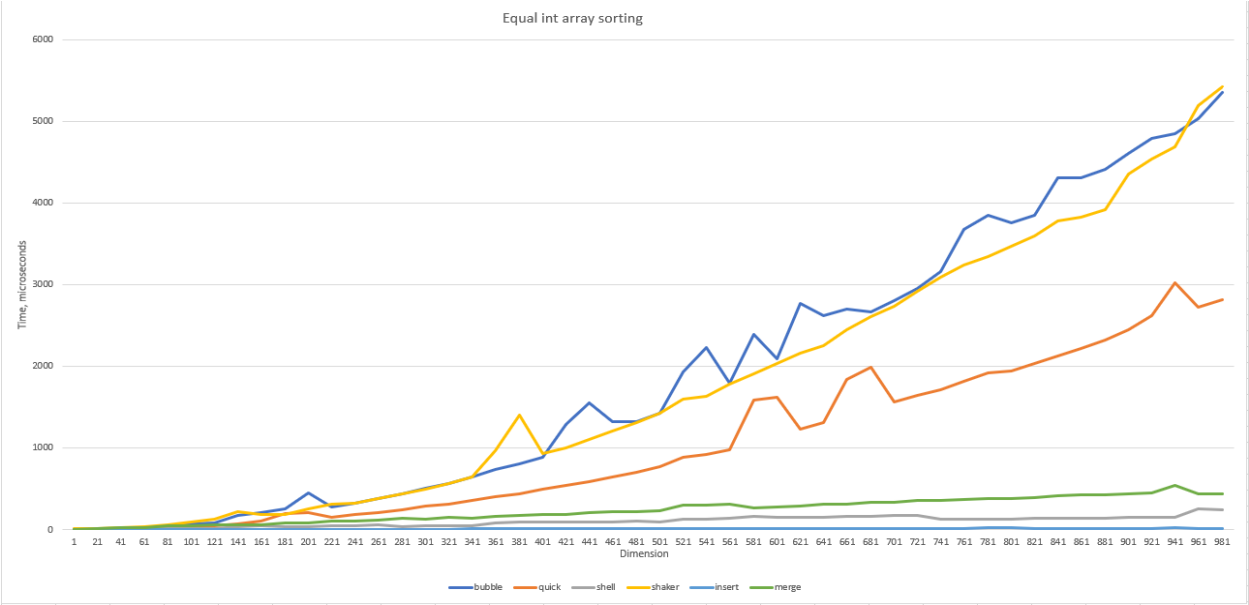
Анализ графиков

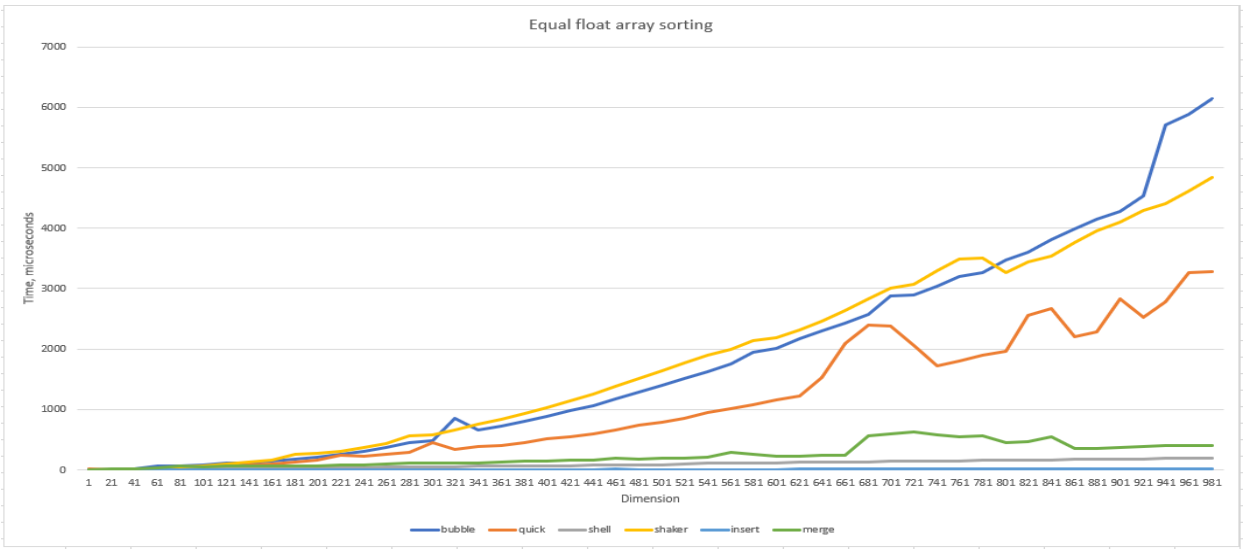
Массивы со случайными данными:



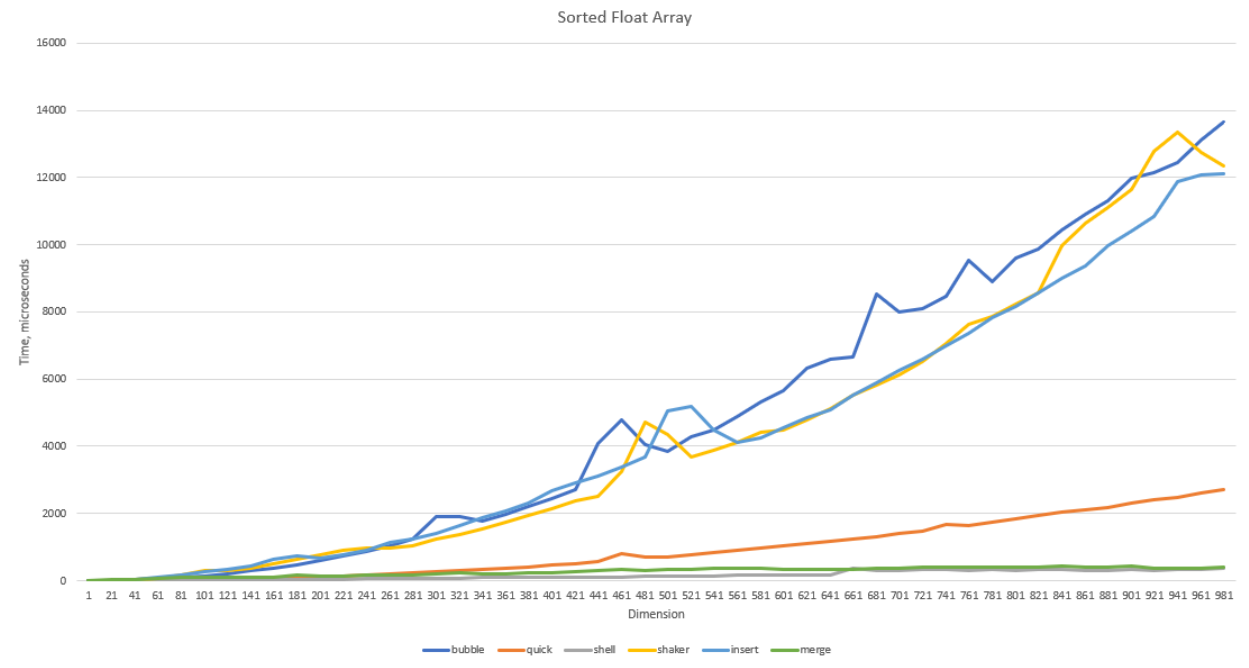


Отсортированные массивы:





Отсортированный в обратном порядке массив:



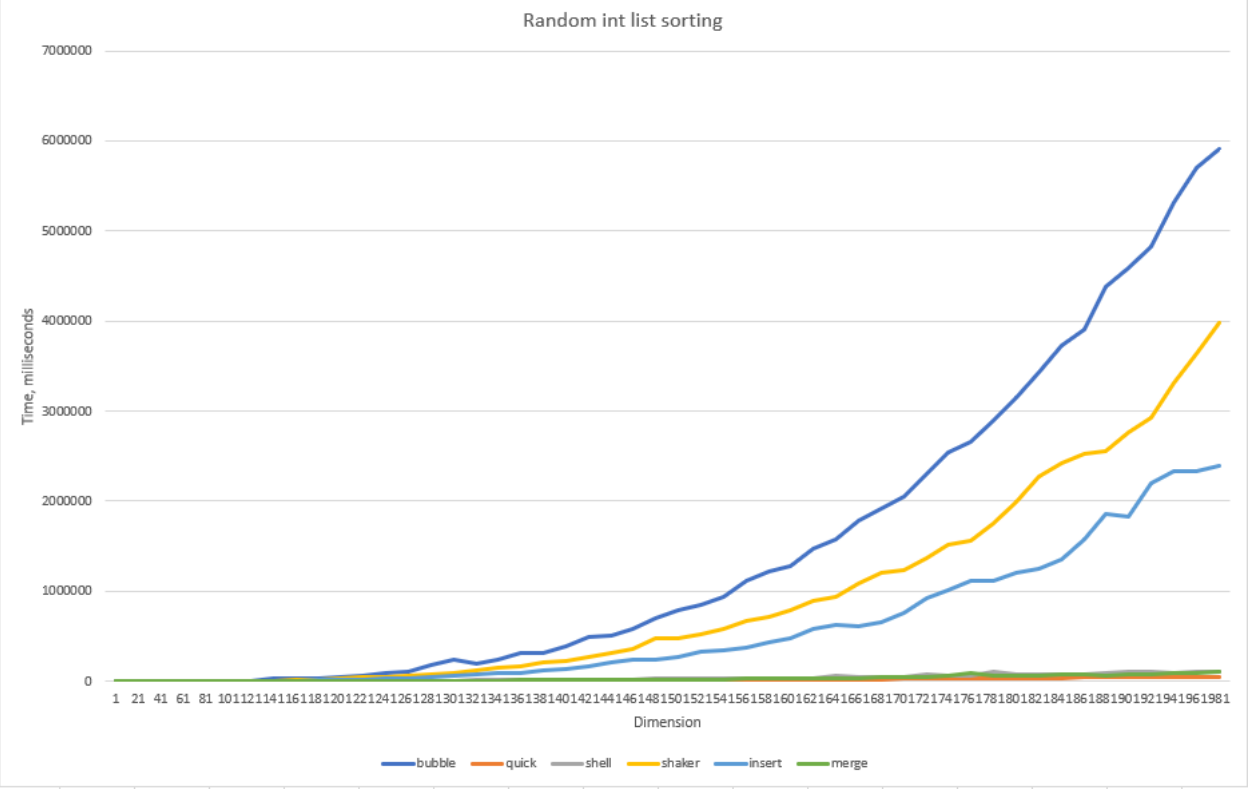
Из данных графиков можно сделать следующие выводы:

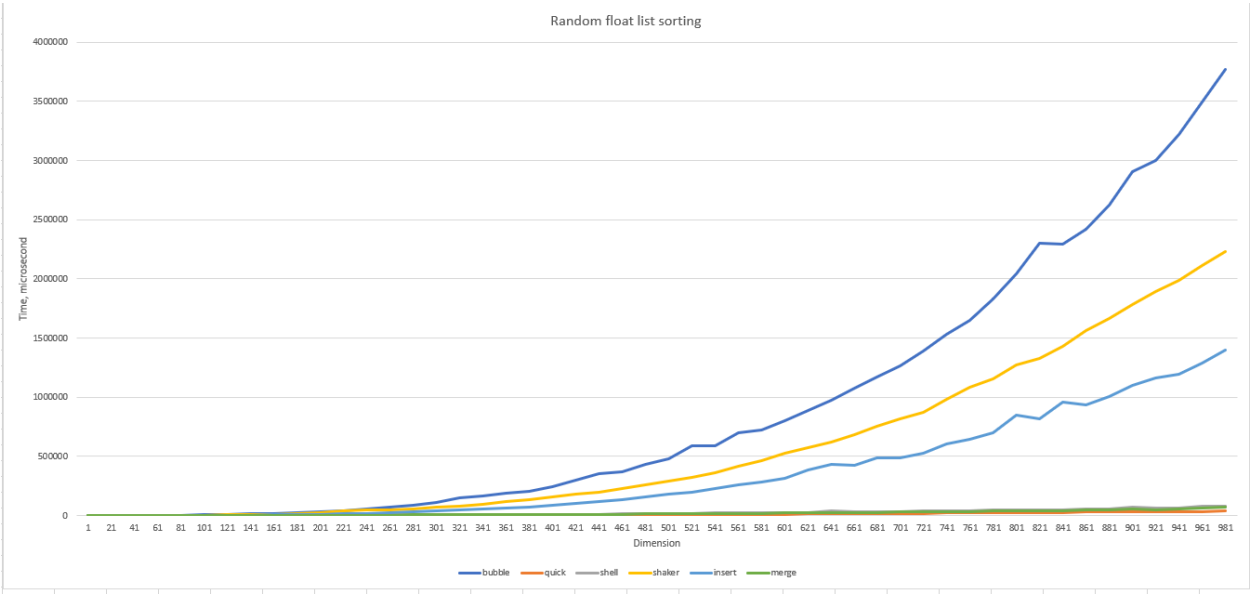
1) На случайных данных сортировки с асимптотикой O(n*log(n)) – Quick Sort, Shell Sort, Merge Sort- работают намного быстрее, чем сортировки с асимптотикой O(n^2) – Bubble Sort, Shaker Sort и Insert Sort.

2) Insert sort на уже отсортированных данных работает очень быстро. При этом здесь проявляется проблема Quick sort – она связана с тем, что в моей реализации алгоритма опорным элементом (pivot) был выбран крайний левый.

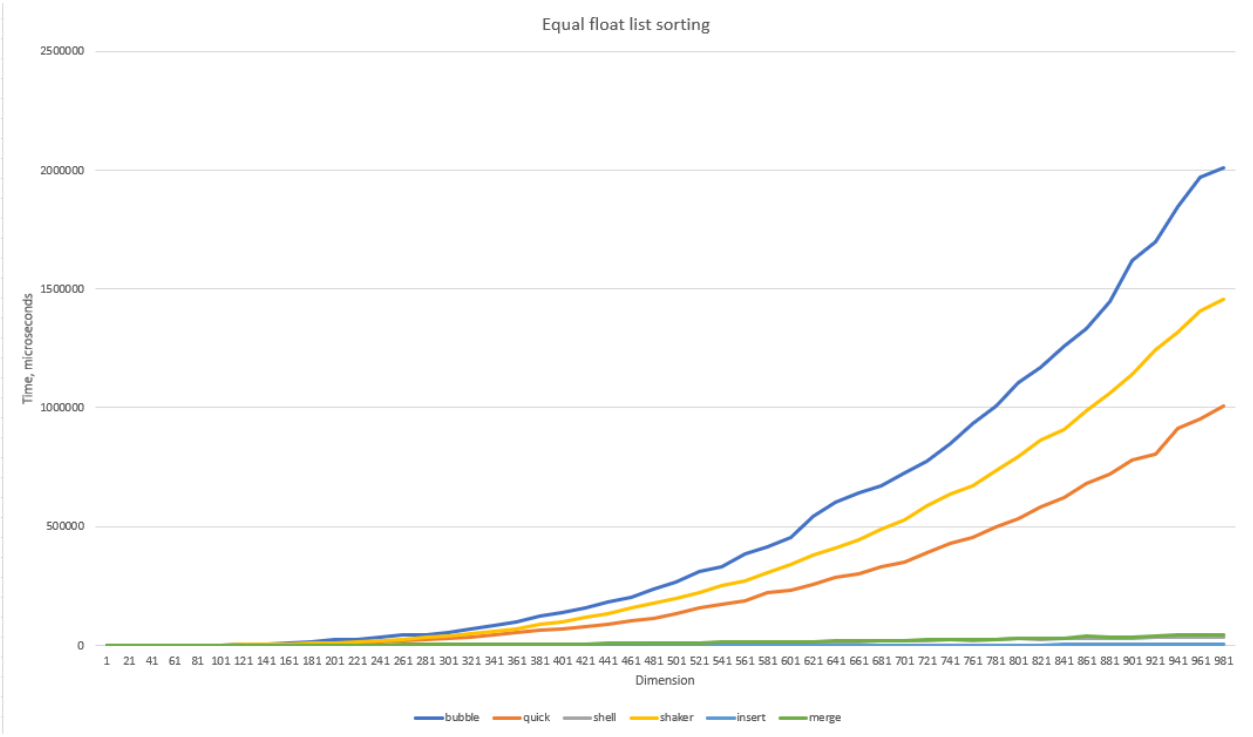
3) На reverse-sorted массиве быстрее всего работают Merge sort и Shell sort, а все остальные сортировки отрабатывают худший случай.

Списки со случайными данными:

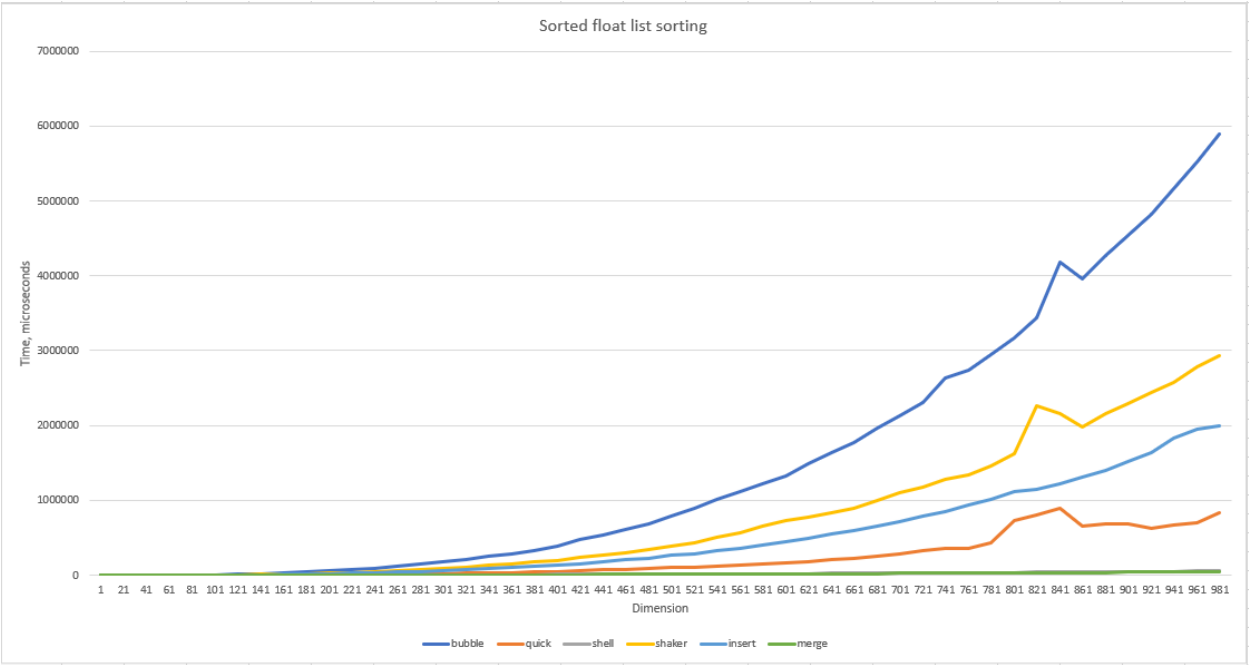




Списки с отсортированными данными:



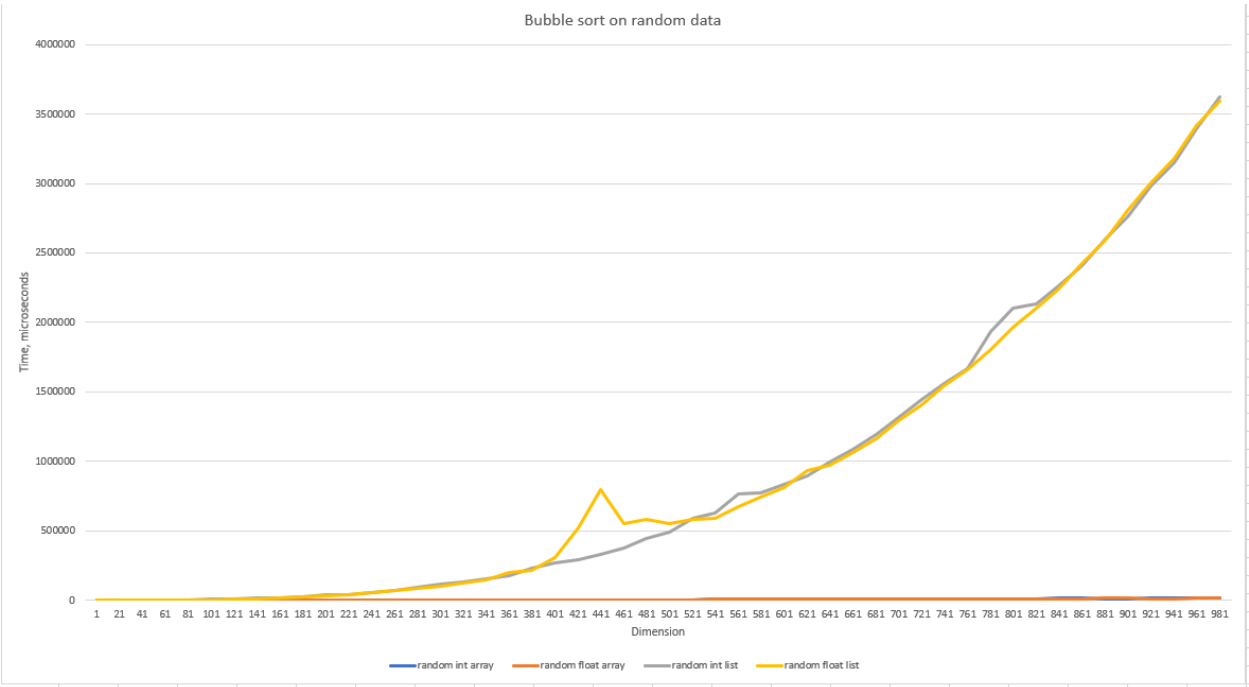
Отсортированный в обратном порядке список:

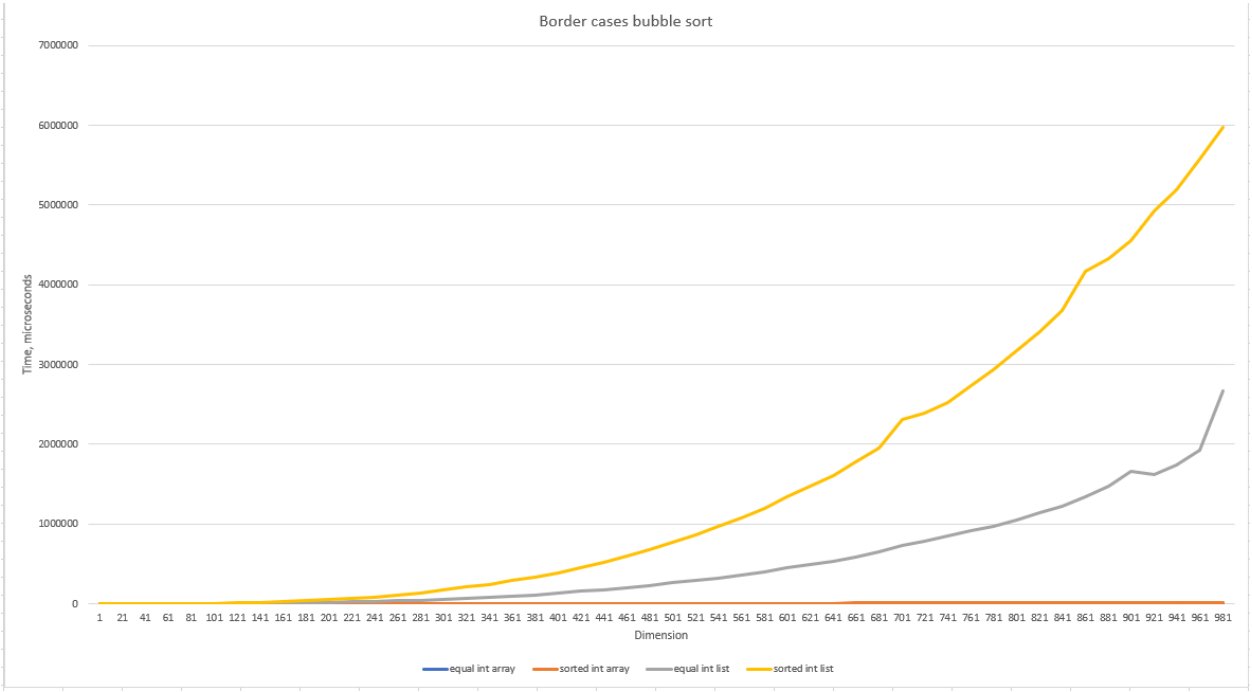


Из графиков видно, что сортировка для списков проходит за гораздо большее время, чем для массивов. Это связано с тем, что доступ к элементу по индексу в массиве осуществляется за O(1), а в списке – за O(0.5*n) – то есть за O(n).

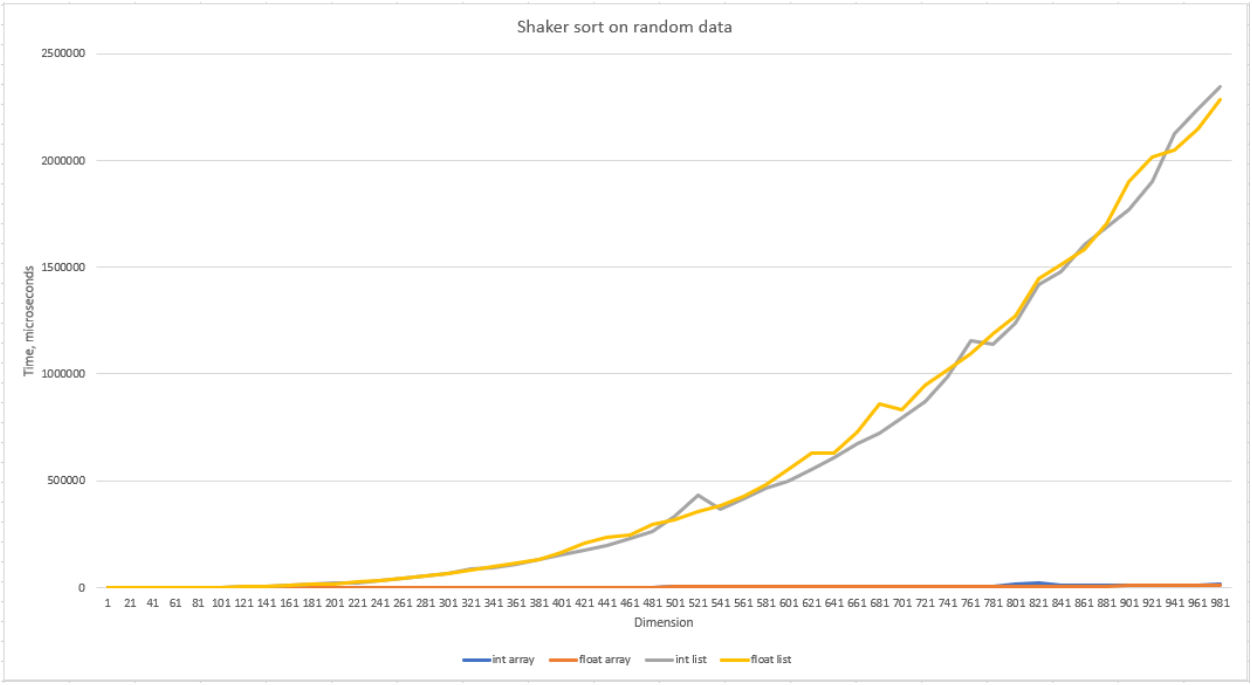
Также были построены графики для демонстрации поведения времени каждой из сортировок на различных входных данных.

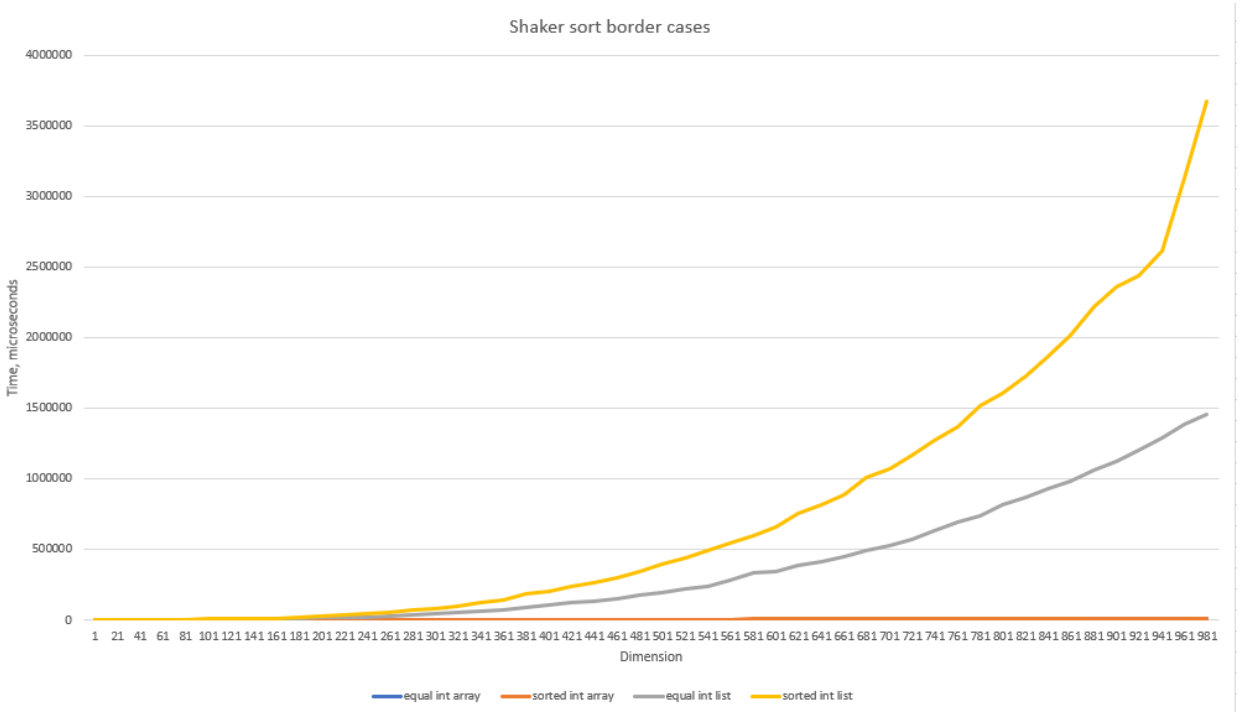
1) Bubble sort



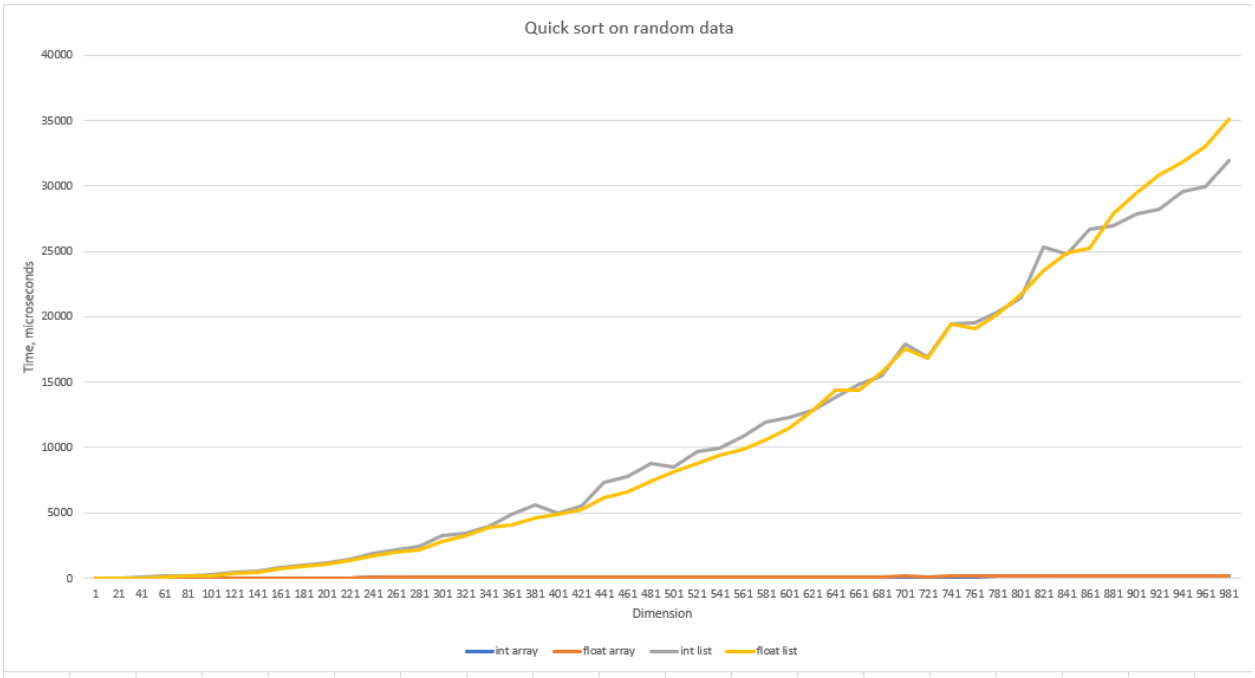


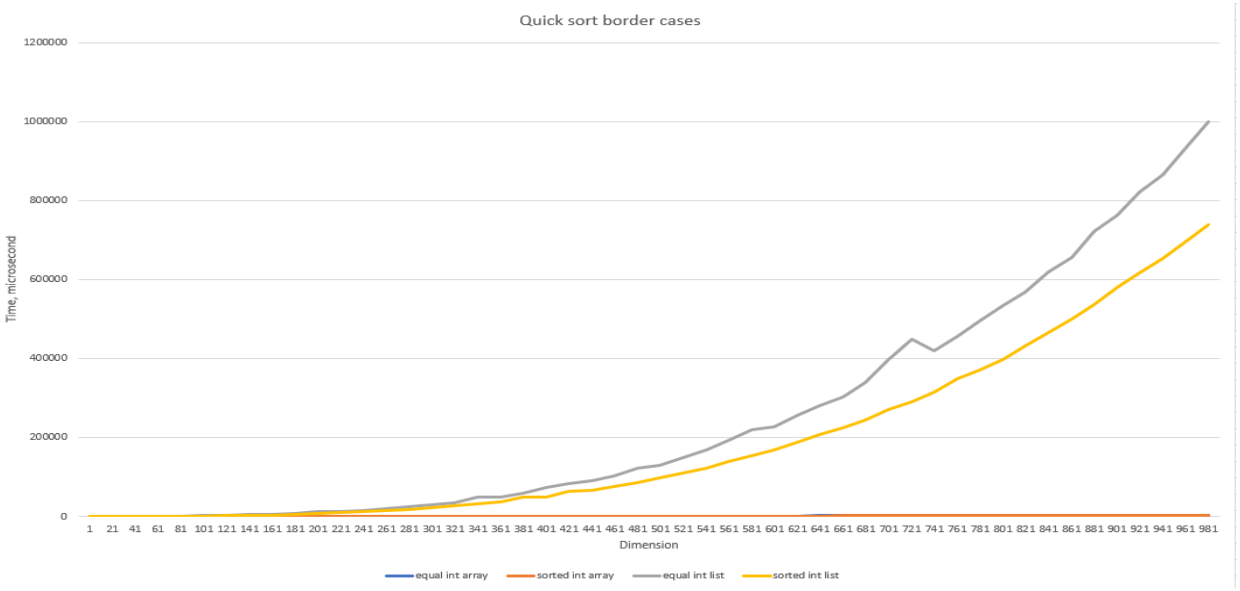
2) Shaker sort



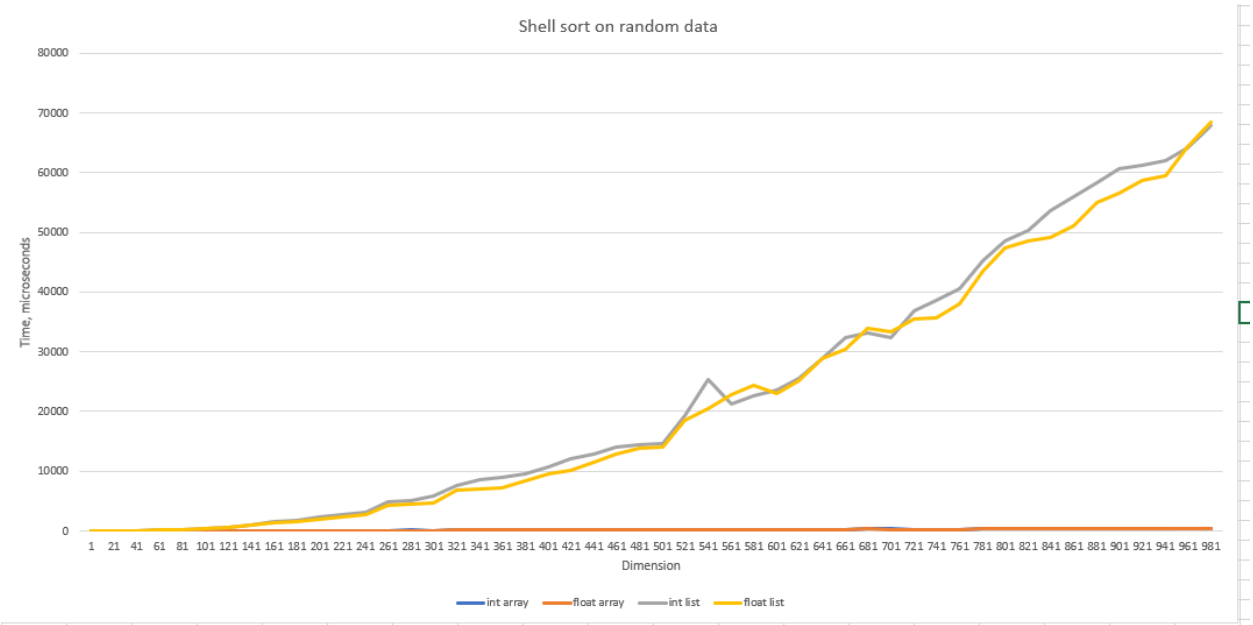

Shaker sort border cases
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
Time, microseconds
1 21 41 61 81 101 121 141 161 181 201 221 241 261 281 301 321 341 361 381 401 421 441 461 481 501 521 541 561 581 601 621 641 661 681 701 721 741 761 781 801 821 841 861 881 901 921 941 961 981
Dimension
equal int array
sorted int array
equal int list
sorted int list

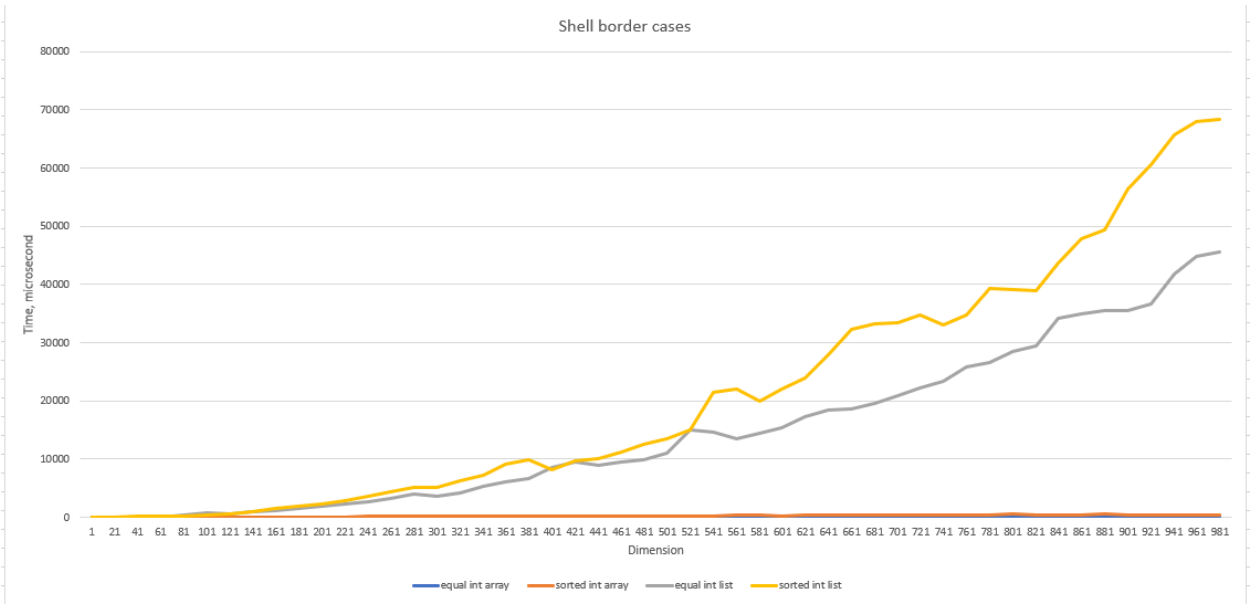
3) Quick sort


Quick sort on random data
40000
35000
30000
25000
20000
15000
10000
5000
0
Time, microseconds
1 21 41 61 81 101 121 141 161 181 201 221 241 261 281 301 321 341 361 381 401 421 441 461 481 501 521 541 561 581 601 621 641 661 681 701 721 741 761 781 801 821 841 861 881 901 921 941 961 981
Dimension
int array
float array
int list
float list

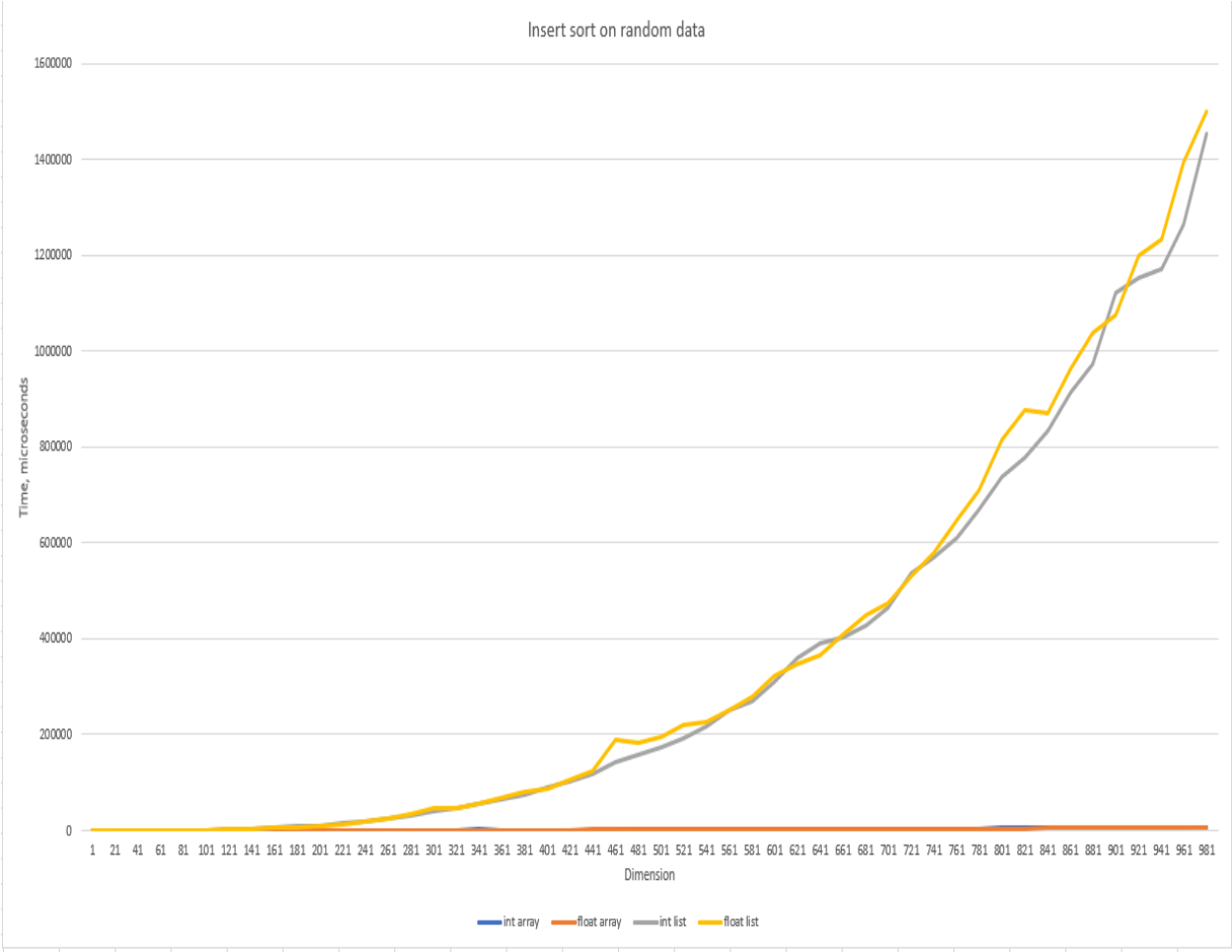

Quick sort border cases
Time, microsecond
Dimension
equal int array
sorted int array
equal int list
sorted int list

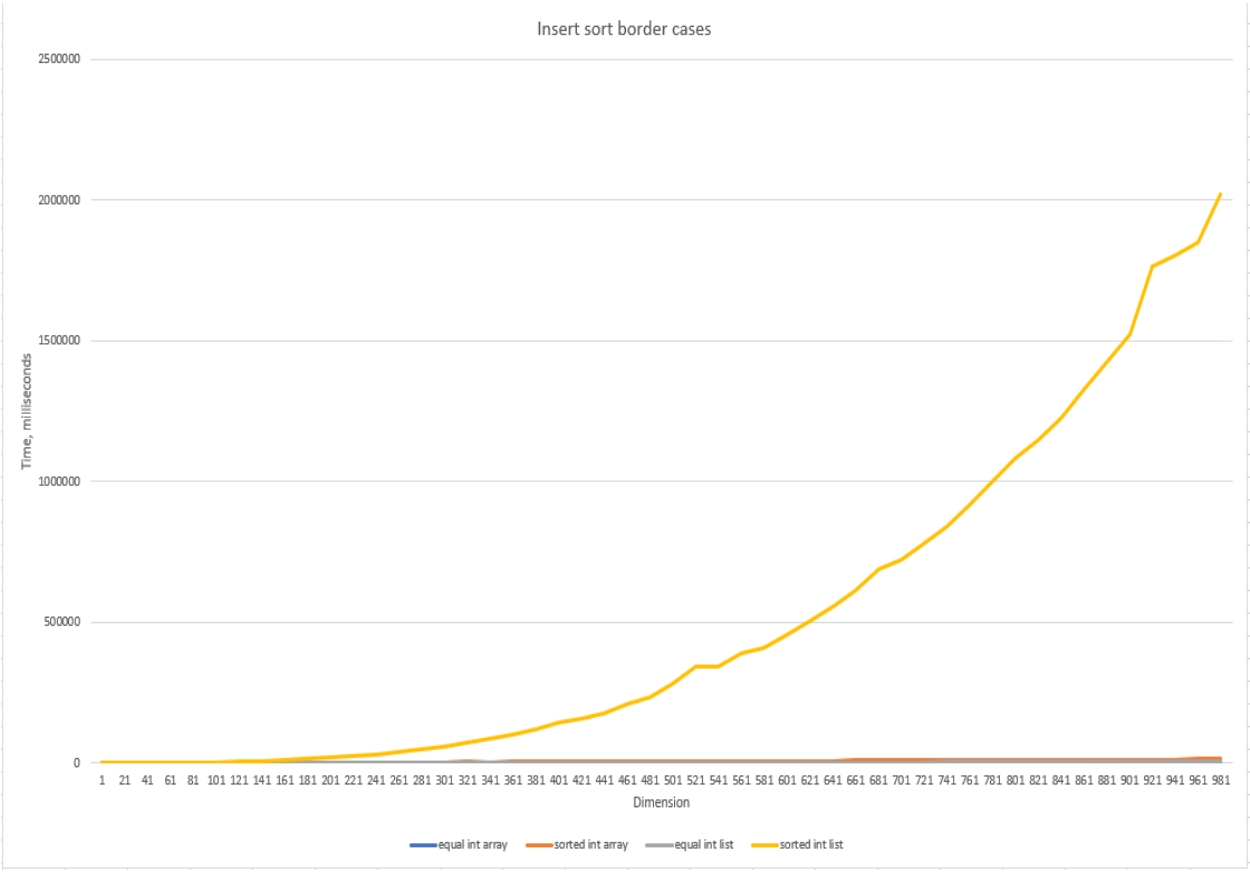
4) Shell sort


Shell sort on random data
Time, microseconds
Dimension
int array
float array
int list
float list

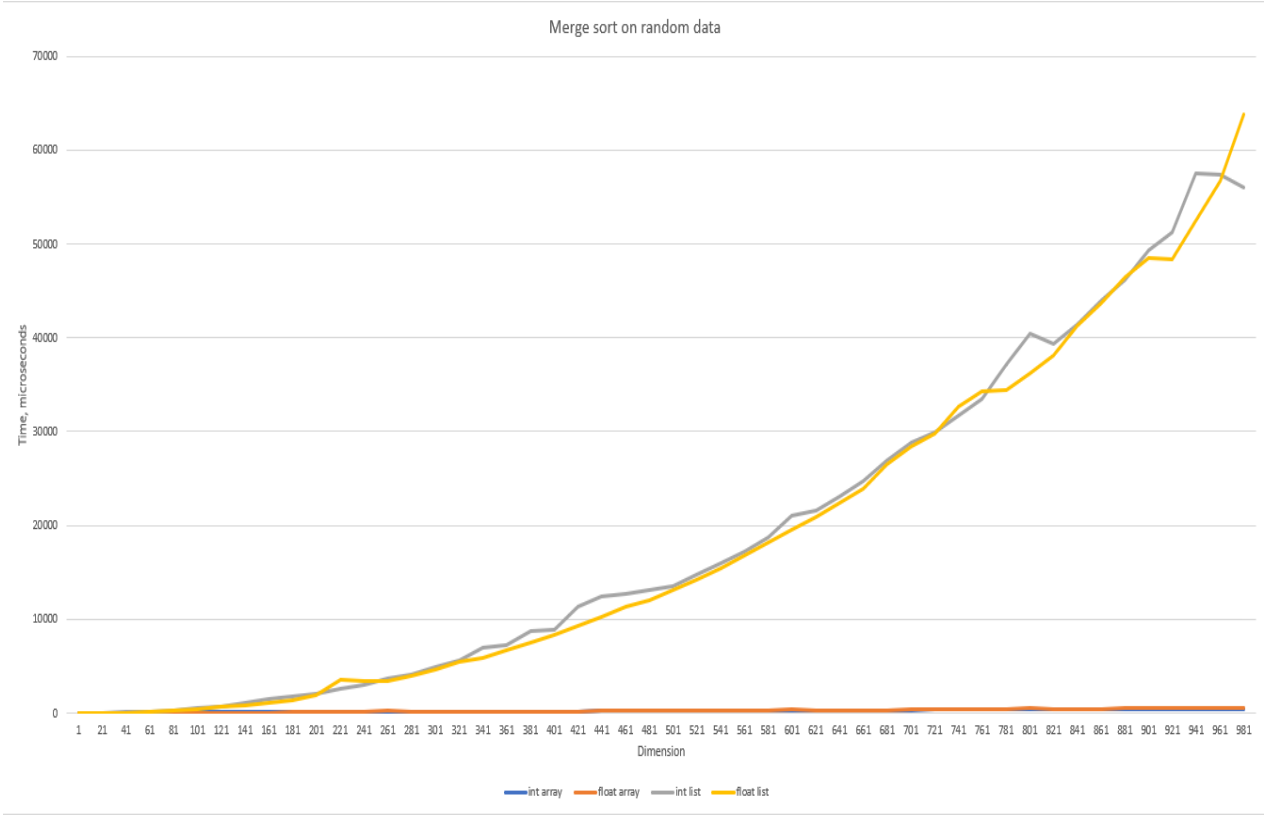

Shell border cases
Time, microsecond
Dimension
equal int array
sorted int array
equal int list
sorted int list

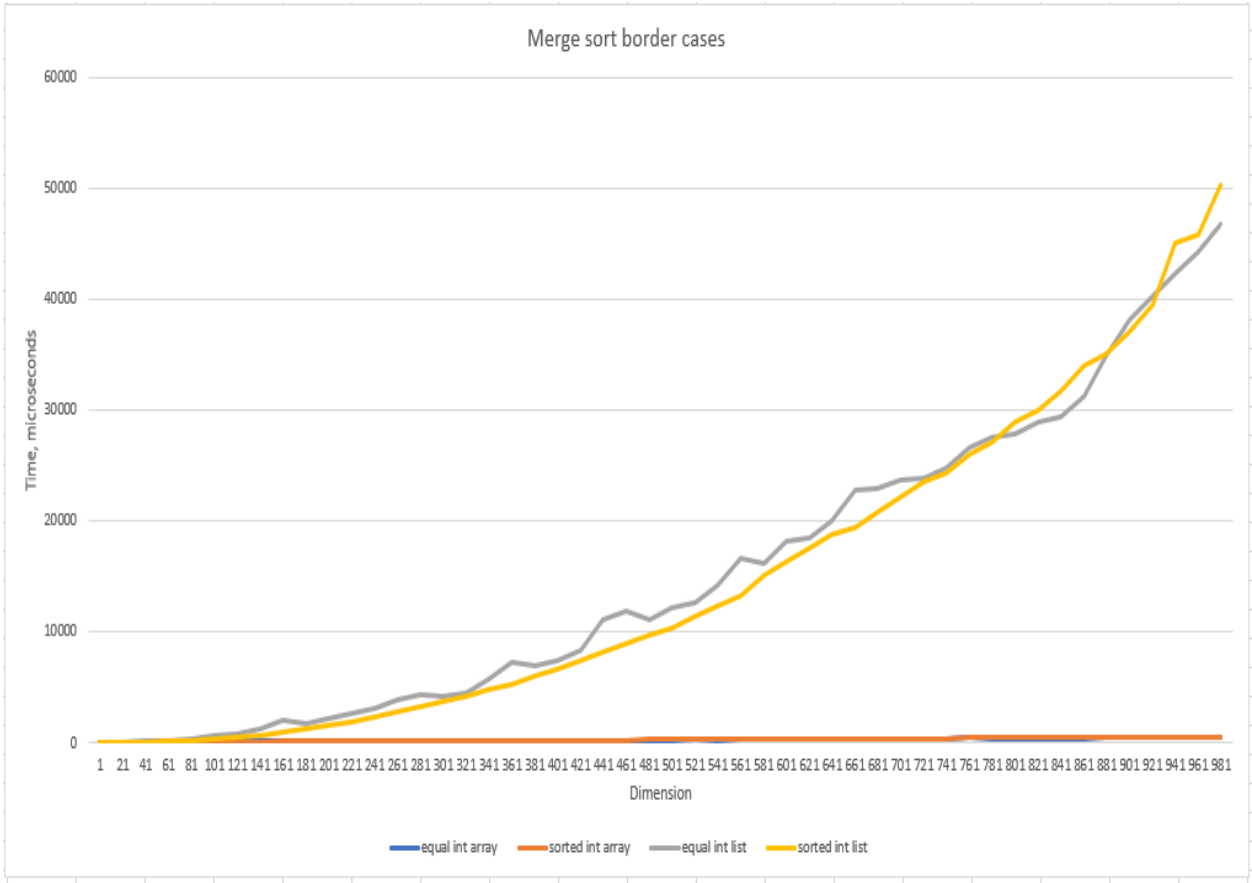
5) Insert sort





6) Merge sort


Merge sort on random data
70000
60000
50000
40000
30000
20000
10000
0
Time, microseconds
1 21 41 61 81 101 121 141 161 181 201 221 241 261 281 301 321 341 361 381 401 421 441 461 481 501 521 541 561 581 601 621 641 661 681 701 721 741 761 781 801 821 841 861 881 901 921 941 961 981
Dimension
int array
float array
int list
float list


Merge sort border cases
60000
50000
40000
30000
20000
10000
0
Time, microseconds
1 21 41 61 81 101 121 141 161 181 201 221 241 261 281 301 321 341 361 381 401 421 441 461 481 501 521 541 561 581 601 621 641 661 681 701 721 741 761 781 801 821 841 861 881 901 921 941 961 981
Dimension
equal int array
sorted int array
equal int list
sorted int list

Сравнение времени работы сортировок с теоретическим и выводы.

Bubble sort на массиве работает за O(n^2) в худшем и среднем случае, и за O(n) в лучшем случае. На связном списке Bubble sort работает за O(n^3), так как доступ к элементу списка по индексу имеет асимптотику O(n). Shaker sort имеет такую же асимптотику, как и Bubble sort (являясь её модификацией). При этом реальное время работы Shaker sort несколько быстрее, так как с каждой итерацией уменьшаются границы рабочей части массива (исключаем из рассмотрения уже отсортированные наибольшие и наименьшие элементы). Quick sort имеет асимптотику O(n*log(n)) в лучшем и среднем случаях, а в худшем её асимптотика будет O(n^2). Этим объясняется её быстрая работа на случайных массивах, но возникновение проблем со скоростью на отсортированных. Сортировка Шелла имеет асимптотику O(n*(log(n)^2)) в лучшем случае и O(n^2) – в худшем. Insert sort в среднем и худшем случаях работает за O(n^2), но в лучшем случае – за O(n). Это объясняет как её большое время работы для random и reverse-sorted структур, так и тот факт, что на отсортированных данных она работает быстрее всех остальных представленных сортировок.Merge sort для всех случаев имеет асимптотику O(n*log(n)). Это объясняет её стабильно быструю работу для данных любой упорядоченности.

Таким образом, полученные результаты времени работы сортировок за исключением некоторых выбросов согласуются с теорией.