A2. Анализ MERGE+INSERTION SORT

Владимиров Алексей, БПИ235

24 ноября 2024 г.

1 Реализация алгоритма

Для реализации алгоритма была использована комбинация двух стандартных алгоритмов: `merge sort` и `insertion sort`. Код `insertion sort` помещен в тело условия, которое останавливает рекурсивные вызовы в процедуре `merge sort`-а. А само условие было ослаблено, так что теперь алгоритм в него заходит при $l - r \leqslant k$.

Стоит отметить, что при $k = 1$ алгоритм вырождается в простой `merge sort`.

2 Тестирующая и генерирующая системы

Для того что бы протестировать эффективность полученного алгоритма, были разработаны специальные классы которые автоматизировали работу с сортировкой.

Первый из них - `ArrayGenerator`. Заранее инициализирован генератор(`seed` в данном случае фиксирован, что бы обеспечить повторяемость результатов при повторных запусках), а так же установлены распределения для генерации чисел.

Далее интерфейс класса представляет из себя следущие методы:

- `getChaotic(n)` - возвращает массив длины `n` заполненный случайными элементами, в случайном порядке
- `getReverseSorted(n)` - возвращает массив длины `n` заполненный случайными элементами, в порядке обратном порядку сортировки
- `getShuffled(n)` - возвращает массив длины `n` заполненный случайными элементами, отсортированный, но слегка перемешанные между собой(всего совершается около 1000 перемешиваний, при размере массива в 10000 элементов)

Следующий класс - `SortTester`. В нем происходят запуски алгоритма на различных данных. Так как класс универсальный, то он хранит в себе умный указатель на функцию сортировки.

В нем есть 2 метода:

`test(depth_of_switch, min_size, max_size, step)` - эти параметры задают, соответственно, размер массива на котором будет выполено переключение на сортировку вставками, и параметры для генерации массивов(используются префиксы заранее сгенерированного массива длинной max_size = 10000).

3 Гиперпараметры тестирования

Тестировались следующие границы перехода к квадратичной сортировке: {1, 5, 10, 20, 30, 50}

Для усреднения результатов было проведено 10 итераций на различных массивах.

Измерения проводились в микросекундах.

4 Графики

Ниже приведены три графика, отражающие сравнительную эффективность алгоритма при разных k, на разных размерах массива. Каждый из графиков отвечает за свой тип массивов.

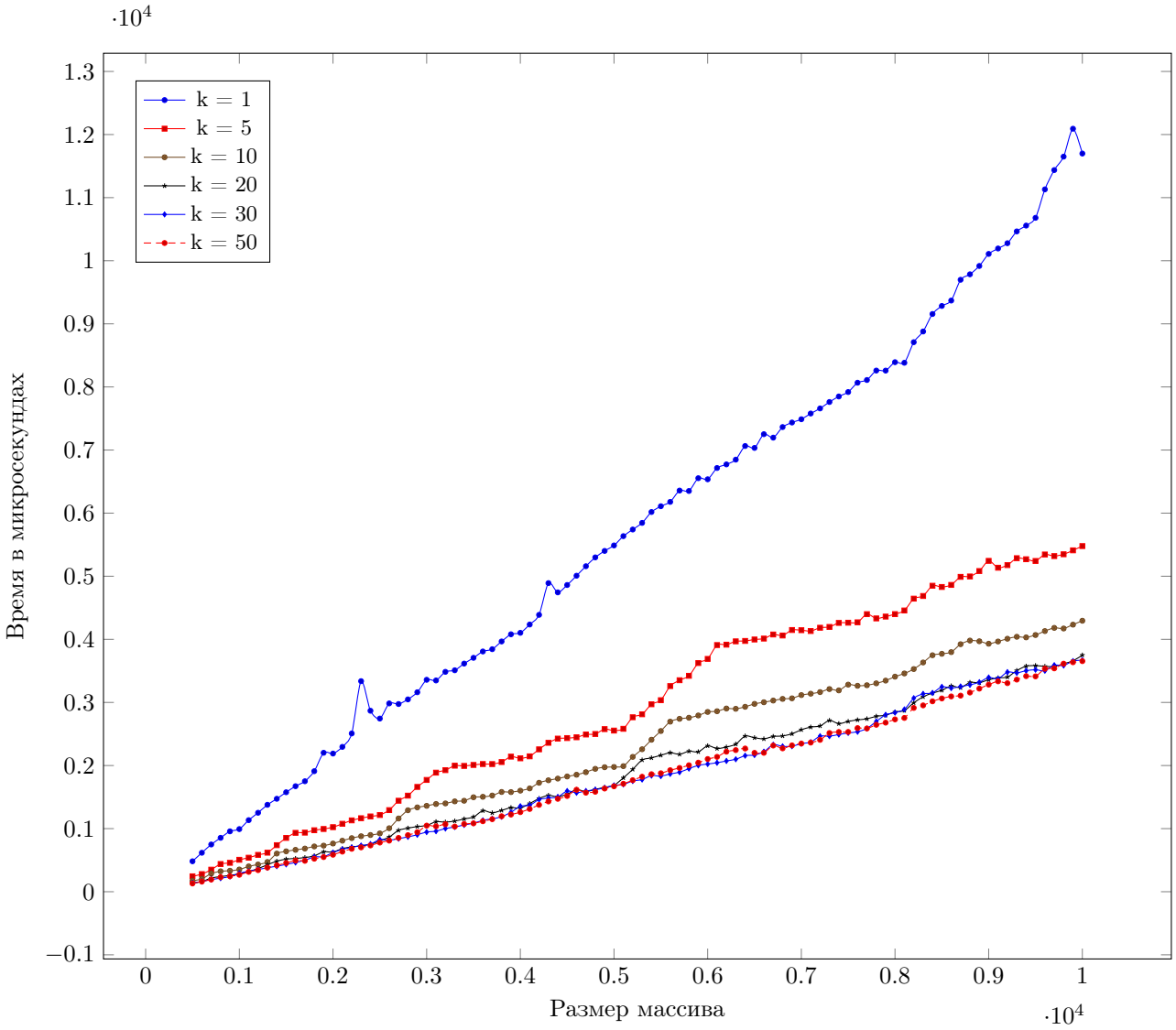

Рис. 1: Случайные, не отсортированные массивы чисел

На графике видно, что хотя все алгоритмы показывают кривые времени, которые неплохо регрессируются прямой, очевидно что хуже всех работает обычный `merge sort`. Коэффициент угла наклона у нее был бы самым высоким, однако даже здесь можно видеть «отметки» после каждой из которых алгоритм начинает расти еще сильнее, те он активно демонстрирует природу своей оценки - $O(nlogn)$.

Уже при отсечении в 5 элементов достигается значительное ускорение. Алгоритм с отсечением по длине равной 5 показывает себя в 3 раза более эффективно на входных данных.

С увеличением размера отсечки, среднее затраченное время на сортировку снижается, однако после примерно $k = 20$ прирост в скорости перестает быть заметным, хотя что интересно, характерные скачки и плато, заметные например при $k = 5$, сглаживаются.

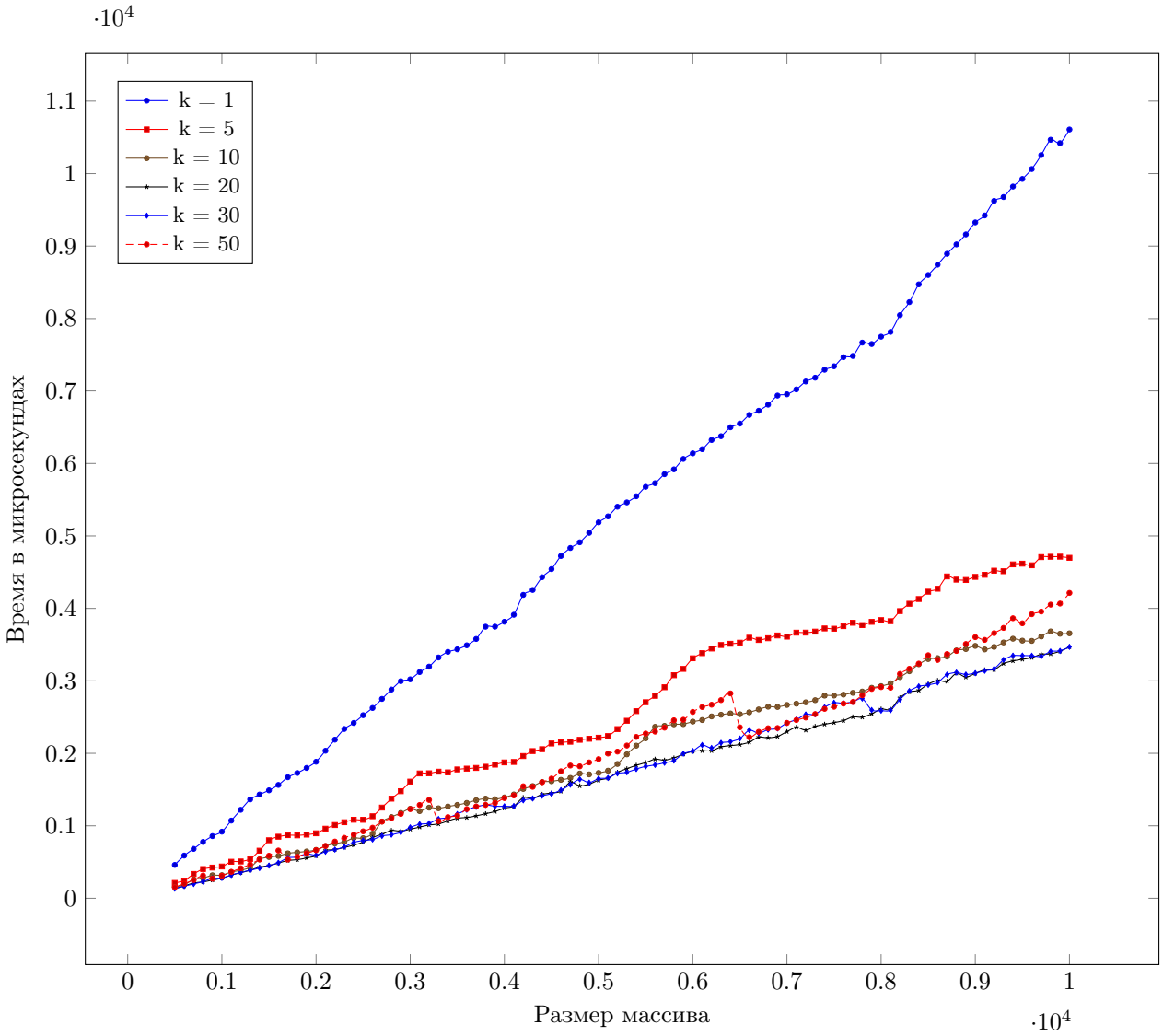

Рис. 2: Случайные, отсортированные в обратном порядке, массивы чисел

На этом графике, общие тенденции сохраняются, хотя в абсолютных величинах алгоритмы тратят меньше времени на выполнение задачи.

По-прежнему лидером по временным затратам остается не модифицированный `merge sort`. График роста сложности стал плавнее, и теперь заканчивается на отметке в 11 тысяч мс а не 13 как было раньше, но в нем все так же легко угадываются 3 «надлома» укзазывающие на ассимптотическую сложность алгоритма.

Остальные алгоритмы не сильно изменили позиции. Разрыв составляет не менее 2.75 раз по сравнению с наивной реализацией. Однако, алгоритм с параметром $k = 50$ стал давать более шумный(зубчатый) график, и под конец, обогнал алгоритмы с $k \in \{30, 10, 20\}$. Это отвечает нашей интуиции, что оптимальный k не слишком большой и лежит где-то между 10 и 20. Характерные пики последнего алгоритма обусловлены скорее всего тем, что квадратичная сортировка начинает работать слишком заметно плохо, а в каждой точке пика, массив «удваивается»(те длина массива находится рядом с некоторой степенью двойки) и срабатывает стратегия разделяй и властвуй, деля массив пополам, и возвращая время опять вниз.

В то время как для `merge sort`-а случай обратно отсортированного массива никак не отличается по сложности от любой другой перестановки элементов в массиве, для квадратичной сортировки это оказывается существенным, и она начинает преобладать сильно раньше чем в других случаях.

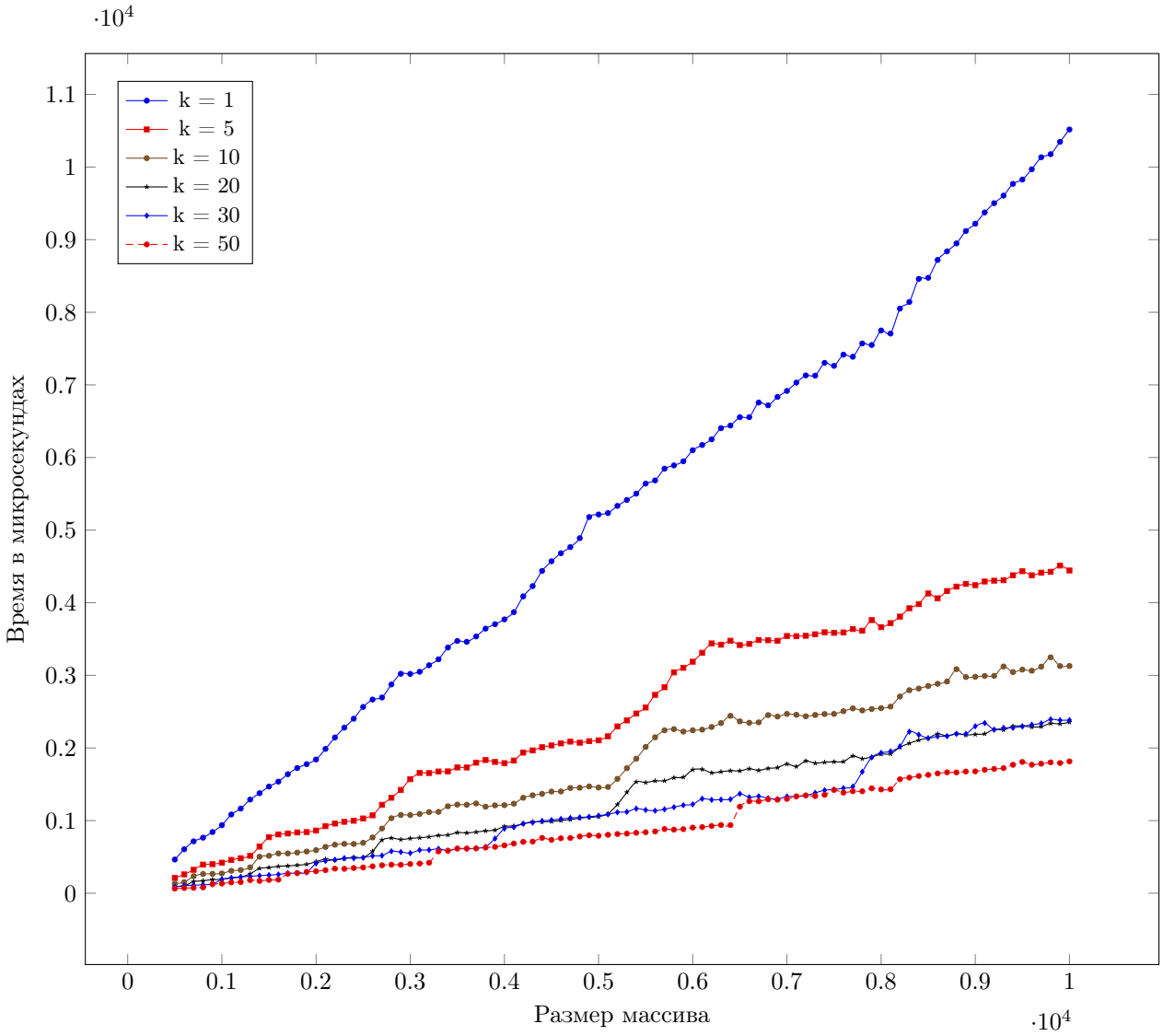

Рис. 3: Случайные, «почти» отсортированные, массивы чисел

В последнем типе массивов сохраняются закономерности отраженные на первом графике. Разрыв так же составляет более 2.75 раз.

Но что можно отметить, так это то что графики ведут себя куда более плавно, гораздо меньше резких пиков и локальных скачков, что естественно обуславливается тем, что данные в большей степени отсортированы чем в первом случае.

Так же, это сильно влияет на алгоритм сортировки вставками, так как на этом графике можно увидеть гораздо более четкую дифференциацию между всеми алгоритмами, в том числе и когда k уже велико.

Алгоритм с $k = 50$ показывает здесь наилучшие результаты(понятное дело, ведь сортировка вставками просто пробегает большие куски массива, практически ничего не делая) около 100-150 микросекунд.

5 Выводы

Так как сортировка вставками зависит от порядка элементов в массиве, а сортировка слиянием демонстрирует устойчивость к нему, то для случайных массивов(которых большинство на практике) рекомендуется брать константу k, такую что:

$$\frac{k}{n} \leqslant 0.002 \text{ - где } n \text{ длина массива}$$

Этот вывод, подтвердился на практике(см графики выше), где средняя k не превышает 15-20 для оптимальной работы алгоритма во всех случаях.

Такой алгоритм будет давать в среднем выигрыш в 3 раза, по скорости работы, по сравнению с обычной сортировкой слияния.

6 Ресурсы

Ссылка на гитхаб репозиторий: `https://github.com/pepsiplusmilk/DSA_HSE_SE_SET3/tree/main/A2` Ссылка на посылку в контесте: `https://dsahse.contest.codeforces.com/group/NOflOR1Qt0/contest/565612/submission/293131149`