МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**РАСПАРАЛЛЕЛИВАНИЕ СОРТИРОВОК МАССИВОВ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.Ю. Романенко

(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) «Технология программирования»

Научный руководитель

канд. физ.-мат. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н. Лапина

(подпись)

Нормоконтролер

доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Е. Полупанова

(подпись)

Краснодар

2018

**РЕФЕРАТ**

Курсовая работа содержит 24 страниц, 2 рисунков, 10 источников.

Ключевые слова:

СОРТИРОВКА, МАССИВ, МНОГОПОТОЧНОСТЬ, РАСПАРАЛЛЕЛИВАНИЕ, АЛГОРИТМ.

Цель курсовой работы – исследовать основные способы распараллеливания сортировок массивов и реализовать соответствующую программу.

В курсовой работе изучены принципы технологии распараллеливания сортировок массивов, понятие многопоточности и способы применения распараллеливания программ.

В курсовой работе разработана программа, позволяющая оценить производительность различных сортировок, и на основе полученных данных в различных ситуациях выбрать более приоритетную.

Реализация выполнена на языке программирования Kotlin с использованием набора библиотек JMH и пакета Coroutines.

**СОДЕРЖАНИЕ**

Введение ..................................................................................................................4

1 Принципы технологии распараллеливания программ.......................................5

1.1 Сортировки массивов в программировании ...............................................6

1.1.1 Быстрая сортировка (Quicksort) ………………………......................8

1.1.2 Сортировка слиянием (Merge sort) ......................................................9

1.1.3 Сортировка Шелла (Shellsort) ...........................................................10

1.1.4 Пирамидальная сортировка (Heapsort) .............................................11

1.2 Понятие многопоточности .........................................................................13

1.3 Различия между распараллеливанием и многопоточностью ...................15

2 Язык программирования Kotlin ........................................................................17

2.1 Набор библиотек JMH.................................................................................18

2.2 Возможности Kotlin Couroutines.................................................................18

3 Программная реализация приложения..............................................................20

Заключение.............................................................................................................23

Список использованных источников...................................................................24

**ВВЕДЕНИЕ**

В настоящее время новые информационные технологии занимают важнейшее место не только в специализированных, но и в повседневных сферах жизни. Компьютеры применяются в бизнесе, менеджменте, торговле, юриспруденции и многих других сферах деятельности человека.

Компьютерные технологии очень удобны для выполнения разнообразных операций, но в разных сферах применения эти операции разные. Потому, каждая отдельная отрасль, которая использует специфические технические средства, нуждается в своих собственных программах, которые обеспечивают работу компьютеров.

Разработкой программного обеспечения занимается такая отрасль науки, как программирование. Она приобретает все большее и большее значение в последнее время, ведь с каждым днем компьютер становится все более необходимым, все более повседневным явлением нашей жизни. Ведь вычислительная техника прошлых лет уже почти полностью исчерпала себя и не удовлетворяет тем потребностям, которые появляются перед человечеством.

Таким образом, новые информационные технологии очень актуальны в наше время и нуждаются в большем внимании для последующей разработки и совершенствования. Рядом с этим, большое значение имеет также и программирование. Такая технология содержит целый ряд важных внутренних задач. Одной из наиболее важных задач для программирования является задача сортировки. Алгоритм сортировки – это [алгоритм](https://ru.wikipedia.org/wiki/Алгоритм) для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма. В этой же курсовой будет рассмотрена технология распараллеливания сортировок массивов.

**1 Принципы технологии распараллеливания программ**

Распараллеливание программ – процесс адаптации [алгоритмов](https://ru.wikipedia.org/wiki/Алгоритм), записанных в виде программ, для их эффективного исполнения на вычислительной системе [параллельной архитектуры](https://ru.wikipedia.org/wiki/Параллельные_вычислительные_системы) (в последнее время, как правило, на [многопроцессорной](https://ru.wikipedia.org/wiki/Многопроцессорность) [вычислительной системе](https://ru.wikipedia.org/wiki/ЭВМ)). Заключается либо в переписывании программ на специальный язык, описывающий параллелизм и понятный [трансляторам](https://ru.wikipedia.org/wiki/Транслятор) целевой вычислительной системы, либо к вставке специальной разметки (например, инструкций [MPI](https://ru.wikipedia.org/wiki/Message_Passing_Interface) или [OpenMP](https://ru.wikipedia.org/wiki/OpenMP)) [1].

Распараллеливание может быть ручным, автоматизированным и полуавтоматизированным. Для оценки эффективности его качества применяются следующие критерии:

– ускорение - в идеальном случае (отсутствие накладных расходов на организацию параллелизма) равно количеству процессоров;

– загруженность - показывает долю использования процессоров. В идеальном случае равна 1, или 100%.

При распараллеливании важно учитывать не только формальный параллелизм структуры алгоритма, но и то, что обменные операции в параллельных [ЭВМ](https://ru.wikipedia.org/wiki/ЭВМ) происходят, как правило, значительно медленнее арифметических. С этим связано существование львиной доли накладных расходов на организацию параллелизма.

Эффективность параллелизации в первую очередь зависит от организации вычислительного алгоритма и соотношения доли последовательных и параллельных программных блоков [2].

Два различных типа архитектуры компьютеров предоставляют по крайней мере два разных способа организации параллельных вычислений:

– векторизация;

– распараллеливание.

Векторизация представляет собой тип распараллеливания по данным, т.е. одна параллельная инструкция воздействует на разные потоки данных. Распараллеливание в общем случае предоставляет более широкие возможности: наряду с распараллеливанием по данным, возможно распараллеливание по процессам - различные потоки данных участвуют в вычислительном процессе под управлением различных потоков команд.

Необходимые требования к циклам, подлежащим векторизации:

– цикл должен быть самым внутренним;

– не допустимы ветвления внутри тела цикла;

– не допустимы вызовы внешних функций и процедур из тела цикла;

– не должно быть рекурсии элементов векторов или массивов в теле цикла.

Распараллеливание возможно, как по данным, так и по процессам. Причем в обоих случаях может применяться один и тот же программный инструментарий для распараллеливания.

Методы распараллеливания можно условно разделить на следующие:

– директивы оптимизирующего компилятора;

– специальные директивы, расширяющие возможности языка к параллелизации;

– параллельные языки программирования;

– коммуникационные средства, средства процессорного интерфейса.

**1.1 Сортировки массивов в программировании**

Сортировка – один из наиболее распространенных процессов современной обработки данных. Задачи на сортировку данных встречаются на компьютере очень часто. Главным образом, это связано с тем, что разбираться в отсортированных данных намного проще, чем в неотсортированных.

Алгоритм сортировки – это порядок действий для упорядочения элементов в списке. Обычно говорят о сортировке записей (содержащих любые данные) по ключам – фрагментам этих записей, допускающих отношение упорядочения [3].

Для того чтобы обоснованно сделать выбор метода сортировки, рассмотрим параметры, по которым будет производиться оценка алгоритмов.

Время сортировки - сновной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью.

Память - ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы.

Устойчивость - устойчивая сортировка не меняет взаимного расположения равных элементов. Такое свойство может быть очень полезным, если они состоят из нескольких полей, а сортировка происходит по одному из них.

Естественность поведения – эффективность метода при обработке уже отсортированных, или частично отсортированных данных. Алгоритм ведёт себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

Ещё одним важным свойством алгоритма является его сфера применения. Здесь основных типов сортировки две:

1) Внутренняя сортировка оперирует с массивами, целиком помещающимися в оперативной памяти с произвольным доступом к любой ячейке. Данные обычно сортируются на том же месте, без дополнительных затрат.

2) Внешняя сортировка оперирует с запоминающими устройствами большого объёма, но с доступом не произвольным, а последовательным (сортировка файлов), то есть в данный момент мы «видим» только один элемент, а затраты на перемотку по сравнению с памятью неоправданно велики. Это накладывает некоторые дополнительные ограничения на алгоритм и приводит к специальным методам сортировки, обычно использующим дополнительное дисковое пространство. Кроме того, доступ к данным на носителе производится намного медленнее, чем операции с оперативной памятью.

Существует множество методов сортировки, каждый из которых имеет свои достоинства и недостатки.

**1.1.1 Быстрая сортировка (Quicksort)**

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как п[узырьковая сортировка](https://ru.wikipedia.org/wiki/Пузырьковая_сортировка) и ш[ейкерная сортировка](https://ru.wikipedia.org/wiki/Шейкерная_сортировка)), известного, в том числе, своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. Любопытный факт: улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов[4].

Общая идея алгоритма состоит в следующем:

– выбрать из массива элемент, называемый опорным- это может быть любой из элементов массива, от выбора которого не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

– сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные и большие».

– для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие». Такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Хоар разработал этот метод применительно к [машинному переводу](https://ru.wikipedia.org/wiki/Машинный_перевод). Словарь хранился на [магнитной ленте](https://ru.wikipedia.org/wiki/Магнитная_лента), и сортировка слов обрабатываемого текста позволяла получить их переводы за один прогон ленты, без перемотки её назад.

Быстрая сортировка относится к алгоритмам «разделяй и властвуй».

Алгоритм состоит из трёх шагов:

1) Выбрать элемент из массива. Назовём его опорным.

2) Разбиение: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.

3) Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

Асимптотика: O(n\*logn) в среднем и лучшем случае, O(n2). Наихудшая оценка достигается при неудачном выборе опорного элемента.

**1.1.2 Сортировка слиянием (Merge sort)**

Сортировка слиянием – [алгоритм сортировки](https://ru.wikipedia.org/wiki/Алгоритм_сортировки), который упорядочивает [списки](https://ru.wikipedia.org/wiki/Список_(информатика)) (или другие [структуры данных](https://ru.wikipedia.org/wiki/Структура_данных), доступ к [элементам](https://ru.wikipedia.org/wiki/Множество" \l "Элемент_множества) которых можно получать только [последовательно](https://ru.wikipedia.org/wiki/Последовательный_доступ), например – [потоки](https://ru.wikipedia.org/wiki/Поток_данных)) в определённом порядке. Эта сортировка – хороший пример использования принципа «[разделяй и властвуй](https://ru.wikipedia.org/wiki/Разделяй_и_властвуй_(программирование))». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью [рекурсивного вызова](https://ru.wikipedia.org/wiki/Рекурсия) или непосредственно, если их размер достаточно мал. Наконец, их [решения](https://ru.wikipedia.org/wiki/Решение_задач) комбинируются, и получается решение исходной задачи [5].

Рассмотрим алгоритм сортировки слиянием. Разделим массив пополам, рекурсивно отсортируем части, после чего выполним процедуру слияния: поддерживаем два указателя, один на текущий элемент первой части, второй – на текущий элемент второй части. Из этих двух элементов выбираем минимальный, вставляем в ответ и сдвигаем указатель, соответствующий минимуму. Слияние работает за O(n), уровней всего logn, поэтому асимптотика O(n\*logn). Эффективно заранее создать временный массив и передать его в качестве аргумента функции. Эта сортировка рекурсивна, как и быстрая, а потому возможен переход на квадратичную при небольшом числе элементов.

**1.1.3 Сортировка Шелла (Shellsort)**

Сортировка Шелла – [алгоритм сортировки](https://ru.wikipedia.org/wiki/Алгоритм_сортировки), являющийся усовершенствованным вариантом [сортировки вставками](https://ru.wikipedia.org/wiki/Сортировка_вставками). Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами – это сортировка вставками с предварительными «грубыми» проходами. Аналогичный метод усовершенствования [пузырьковой сортировки](https://ru.wikipedia.org/wiki/Сортировка_простыми_обменами) называется [сортировка расчёской](https://ru.wikipedia.org/wiki/Сортировка_расчёской) [6].

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d=1 (то есть обычной [сортировкой вставками](https://ru.wikipedia.org/wiki/Сортировка_вставкой)). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, [пузырьковой](https://ru.wikipedia.org/wiki/Сортировка_методом_пузырька), каждая перестановка двух элементов уменьшает количество [инверсий](https://ru.wikipedia.org/wiki/Инверсия_(перестановка)) в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем [быстрая сортировка](https://ru.wikipedia.org/wiki/Быстрая_сортировка), она имеет ряд преимуществ:

– отсутствие потребности в памяти под стек;

– отсутствие деградации при неудачных наборах данных – быстрая сортировка легко деградирует до O(n²), что хуже, чем худшее гарантированное время для сортировки Шелла.

Существует довольно много последовательностей с разными оценками. Последовательность Шелла – первый элемент равен длине массива, каждый следующий вдвое меньше предыдущего. Асимптотика в худшем случае – O(n2). Последовательность Хиббарда – 2n - 1, асимптотика в худшем случае – O(n1,5), последовательность Седжвика – O(n4/3), Пратта (все произведения степеней двойки и тройки) – O(n\*log2n).

**1.1.4 Пирамидальная сортировка (Heapsort)**

Пирамидальная сортировка – [алгоритм сортировки](https://ru.wikipedia.org/wiki/Алгоритм_сортировки), работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за O (n\*logn) операций при сортировке n элементов. Количество применяемой служебной памяти не зависит от размера массива (то есть, O(1)) [7].

Может рассматриваться как усовершенствованная [сортировка пузырьком](https://ru.wikipedia.org/wiki/Сортировка_пузырьком), в которой элемент всплывает ([min-heap](https://ru.wikipedia.org/wiki/Min-heap)) / тонет ([max-heap](https://ru.wikipedia.org/wiki/Max-heap)) по многим путям.

Сортировка пирамидой использует [бинарное сортирующее дерево](https://ru.wikipedia.org/wiki/Двоичная_куча). Сортирующее дерево – это такое дерево, у которого выполнены условия:

– каждый лист имеет глубину либо d, либо d-1, d – максимальная глубина дерева.

– значение в любой вершине не меньше (другой вариант – не больше) значения её потомков.

Удобнаяструктура данных для сортирующего дерева – такой массив Array, что Array[0] – элемент в корне, а потомки элемента Array[i] являются Array[2i+1] и Array[2i+2].

Алгоритм сортировки будет состоять из двух основных шагов:

– выстраиваем элементы массива в виде сортирующего дерева. Этот шаг требует O(n) операций.

– будем удалять элементы из корня по одному за раз и перестраивать дерево. То есть на первом шаге обмениваем Array[0] и Array[n-1], преобразовываем Array[0], Array[1],…, Array[n-2] в сортирующее дерево. Затем переставляем Array[0] и Array[n-2], преобразовываем Array[0], Array[1], …, Array[n-3] в сортирующее дерево. Процесс продолжается до тех пор, пока в сортирующем дереве не останется один элемент. Тогда Array[0], Array[1], …, Array[n-1] – упорядоченная последовательность.

Этот шаг требует O(n\logn) операций.

**1.2 Понятие многопоточности**

Многопото́чность – свойство платформы (например, [операционной системы](https://ru.wikipedia.org/wiki/Операционная_система), [виртуальной машины](https://ru.wikipedia.org/wiki/Виртуальная_машина) и т.д.) или [приложения](https://ru.wikipedia.org/wiki/Прикладное_программное_обеспечение), состоящее в том, что [процесс](https://ru.wikipedia.org/wiki/Процесс_(информатика)), порождённый в операционной системе, может состоять из нескольких [потоков](https://ru.wikipedia.org/wiki/Поток_выполнения), выполняющихся «[параллельно](https://ru.wikipedia.org/wiki/Параллельные_вычисления)», то есть без предписанного порядка во [времени](https://ru.wikipedia.org/wiki/Время). При выполнении некоторых задач такое разделение может достичь более эффективного использования [ресурсов вычислительной машины](https://ru.wikipedia.org/w/index.php?title=Вычислительные_ресурсы&action=edit&redlink=1) [8].

Такие потоки называют также потоками выполнения (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) [thread of execution](https://en.wikipedia.org/wiki/thread_(computer_science))); иногда называют «нитями» (буквальный перевод [англ.](https://ru.wikipedia.org/wiki/Английский_язык) thread) или неформально «тредами».

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в [адресном пространстве](https://ru.wikipedia.org/wiki/Адресное_пространство) процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие [дескрипторы файлов](https://ru.wikipedia.org/wiki/Файловый_дескриптор). Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность (как доктрину [программирования](https://ru.wikipedia.org/wiki/Программирование)) не следует путать ни с [многозадачностью](https://ru.wikipedia.org/wiki/Многозадачность), ни с [многопроцессорностью](https://ru.wikipedia.org/wiki/Многопроцессорность), несмотря на то, что [операционные системы](https://ru.wikipedia.org/wiki/Операционная_система), реализующие [многозадачность](https://ru.wikipedia.org/wiki/Многозадачность), как правило, реализуют и многопоточность.

К достоинствам многопоточной реализации той или иной системы перед многозадачной можно отнести следующее:

– упрощение программы в некоторых случаях за счёт использования общего адресного пространства;

– меньшие относительно процесса временные затраты на создание потока.

К достоинствам многопоточной реализации той или иной системы перед однопоточной можно отнести следующее:

– упрощение программы в некоторых случаях, за счёт вынесения механизмов чередования выполнения различных слабо взаимосвязанных подзадач, требующих одновременного выполнения, в отдельную подсистему многопоточности.

– повышение производительности процесса за счёт распараллеливания процессорных вычислений и операций ввода-вывода.

В случае, если потоки выполнения требуют относительно сложного взаимодействия друг с другом, возможно проявление проблем многозадачности, таких как взаимные блокировки.

На обычном процессоре управление потоками осуществляется операционной системой. Поток исполняется до тех пор, пока не произойдёт аппаратное прерывание, системный вызов или пока не истечёт отведённое для него операционной системой время. После этого процессор переключается на код операционной системы, который сохраняет состояние потока (его контекст) или переключается на состояние другого потока, которому тоже выделяется время на исполнение. При такой многопоточности достаточно большое количество тактов процессора тратится на код операционной системы, переключающий контексты. Если поддержку потоков реализовать аппаратно, то процессор сам сможет переключаться между потоками, а в идеальном случае – выполнять несколько потоков одновременно за каждый такт. Для операционной системы и пользователя один такой физический процессор будет виден как несколько логических процессоров.

Различают две формы многопоточности, которые могут быть реализованы в процессорах аппаратно:

– в[ременная многопоточность](https://ru.wikipedia.org/wiki/Temporal_multithreading) (англ. Temporal multithreading);

– о[дновременная многопоточность](https://ru.wikipedia.org/wiki/Simultaneous_multithreading) (англ. Simultaneous multithreading).

**1.3 Различия между распараллеливанием и многопоточностью**

Некоторые люди путают многопоточное и асинхронное программирование, а некоторые говорят, что это одно и тоже. Итак, разъясним эти термины и добавим еще одно понятие, такое как параллелизм. Здесь есть две концепции и обе они совершенно разные, первая синхронное и асинхронное программирование и вторая – однопоточные и многопоточные приложения[9].

Каждая программная модель (синхронная или асинхронная) может работать в однопоточной и многопоточной среде. Давайте обсудим их подробно.

Синхронная программная модель – это программная модель, когда потоку назначается одна задача и начинается выполнение. Когда завершено выполнение задачи тогда появляется возможность заняться другой задачей. В этой модели невозможно останавливать выполнение задачи чтобы в промежутке выполнить другую задачу. Давайте рассмотрим, как эта модель работает в одном и многопоточном сценарии.

Однопоточность – если мы имеем несколько задач, которые надлежит выполнить, и текущая система предоставляет один поток, который может работать со всеми задачами, то он берет поочередно одну за другой и выполняет их все. (Порядок, в котором задачи выполняются не влияет на общее выполнение, у нас может быть другой алгоритм, который может определять приоритеты задач.

Многопоточность – в этом сценарии, мы использовали много потоков, которые могут брать задачи и приступать к работе с ними. У нас есть пулы потоков (новые потоки также создаются, основываясь на потребности и доступности ресурсов) и множество задач.

Это идеальный сценарий, но в обычных условиях мы используем большее количество задач, чем количество доступных потоков, таким образом, освободившийся поток получает другое задание. Как уже говорилось создание нового потока не происходит каждый раз, потому что для этого требуются системные ресурсы, такие как процессор, память и начальное количество потоков должно быть определенным. Теперь давайте поговорим о Асинхронной модели и как она ведет себя в одно и многопоточной среде.

Асинхронная модель программирования отличается от синхронной программной модели тем, что поток однажды начав выполнение задачи может приостановить выполнение, сохранив текущее состояние, и между тем начать выполнение другой задачи. Если наша система способно иметь много потоков тогда все потоки могут работать в асинхронной модели.

Давайте обсудим еще один термин – параллелизм.

Проще говоря параллелизм – способ обработки множественных запросом одновременно. Так как мы обсуждали два сценария, когда обрабатывались множественные запросы, многопоточное программирование и асинхронная модель (одно и многопоточная). В случае асинхронной модели будь она однопоточной или многопоточной, в то время, когда выполняются множество задач, некоторые из них приостанавливаются, а некоторые выполняются. Существует много особенностей, но это выходит за рамки этой публикации.

**2 Язык программирования Kotlin**

Kotlin – это статически типизированный язык программирования, работающий поверх [JVM](https://ru.wikipedia.org/wiki/JVM) и разрабатываемый компанией [JetBrains](https://ru.wikipedia.org/wiki/JetBrains). Компилируется в [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), а также на другие платформы через инфраструктуру [LLVM](https://ru.wikipedia.org/wiki/LLVM) [10].

Kotlin – это небольшой остров в Финском заливе недалеко от Санкт-Петербурга. Видимо, тем самым создатели дают отсылку к тому, что новый язык, как остров Kotlin – младший русский брат далекого острова [Ява](https://ru.wikipedia.org/wiki/Ява).

Авторы ставили целью создать язык более лаконичный и типобезопасный, чем [Java](https://ru.wikipedia.org/wiki/Java), и более простой, чем [Scala](https://ru.wikipedia.org/wiki/Scala_(язык_программирования)). Следствием упрощения по сравнению со Scala стали также более быстрая [компиляция](https://ru.wikipedia.org/wiki/Компиляция_(программирование)) и лучшая поддержка языка в [IDE](https://ru.wikipedia.org/wiki/Интегрированная_среда_разработки).

Действительно, новый язык компилируется в JVM байт-код. А это значит, что он может заинтересовать всех, кто имеет дело с Java-машиной и в целом с языками со сборщиком мусора.

Две главных особенности Kotlin – это его простота и полная совместимость с Java. Kotlin создавался компанией, которая делает очень много продуктов на Java и которая хорошо разбирается в современных инструментах разработки. Создатели нового языка очень хорошо почувствовали потребности бизнеса и разработчиков: бизнесу дали возможность увеличить эффективность разработчиков, а разработчикам дать современный инструмент для разработки. Простота позволяет использовать язык почти любому Java-разработчику, который готов потратить полчаса на то, чтобы посмотреть документацию или спецификацию языка, обратная совместимость же позволяет использовать язык в уже существующем проекте.

Так как Kotlin нацелен в первую очередь на работу вместе с Java, то и целиком своей стандартной библиотеки у него нет. Большая часть стандартной библиотеки Kotlin нацелена на улучшение и исправление библиотеки старшего брата – Java.

**2.1 Набор библиотек JMH**

Java Microbenchmark Harness – набор библиотек для тестирования производительности небольших функций разных языков программирования, таких как, Kotlin, Java, Scala.

Перед запуском теста JMH следует перекомпилировать код для того, чтобы уменьшить погрешности вычисления времени работы функции при запуске её N раз, подсчитать общее время работы, а потом поделить на N. Для этого требуется обернуть запуск в виде цикла и вызова необходимого метода. Однако в этом случае на время работы функции повлияет сам цикл, а также сам вызов замеряемой функции, поэтому вместо цикла будет вставлен непосредственно код вызова функции, без reflection или генерации методов в runtime.

После переделки байткода тестирование можно запустить специальной командой, так как все необходимые компоненты уже будут запакованы в один jar файл.

**2.2 Возможности Kotlin Couroutines**

Kotlin Coroutines – это компоненты [компьютерных программ,](https://en.wikipedia.org/wiki/Computer_program) которые обобщают [подпрограммы](https://en.wikipedia.org/wiki/Subroutine) для [многозадачной работы без](https://en.wikipedia.org/wiki/Non-preemptive_multitasking) вытеснения, позволяя нескольким [точкам входа](https://en.wikipedia.org/wiki/Entry_point) приостанавливать и возобновлять выполнение в определенных местах.

Coroutines хорошо подходят для реализации знакомых программных компонентов, таких как [совместные задачи](https://en.wikipedia.org/wiki/Cooperative_multitasking), [исключения](https://en.wikipedia.org/wiki/Exception_handling), [циклы событий](https://en.wikipedia.org/wiki/Event_loop), [итераторы](https://en.wikipedia.org/wiki/Iterator), [бесконечные списки](https://en.wikipedia.org/wiki/Lazy_evaluation) и [каналы](https://en.wikipedia.org/wiki/Pipeline_(software)).

По словам [Дональда Кнута](https://en.wikipedia.org/wiki/Donald_Knuth), [Мелвин Конуэй](https://en.wikipedia.org/wiki/Melvin_Conway) ввел термин Coroutine в 1958 году, когда применил его к [программе сборки](https://en.wikipedia.org/wiki/Assembly_language). Первое опубликованное появилось позже, в 1963 году.

Подпрограммы являются частными случаями Coroutines. Когда вызываются подпрограммы, выполнение начинается с начала, а после выхода из подпрограммы оно заканчивается. Экземпляр подпрограммы возвращается только один раз и не содержит состояния между вызовами. Напротив, Coroutines могут выйти, вызвав другие Coroutines, которые могут позже вернуться к точке, в которой они были вызваны в исходной Coroutine.

Таким образом, экземпляр Coroutine содержит состояние и изменяется между вызовами.

Coroutines полезны для реализации различных задач.

[Конечные автоматы](https://en.wikipedia.org/wiki/State_machine) в рамках одной подпрограммы, где состояние определяется текущей точкой входа / выхода процедуры. Это может привести к более удобочитаемому коду по сравнению с использованием [goto](https://en.wikipedia.org/wiki/Goto), а также может быть реализовано посредством [взаимной рекурсии](https://en.wikipedia.org/wiki/Mutual_recursion) с [хвостовыми вызовами](https://en.wikipedia.org/wiki/Tail_call).

[Актерская модель](https://en.wikipedia.org/wiki/Actor_model) параллелизма, например, в [видеоиграх](https://en.wikipedia.org/wiki/Video_game). Каждый субъект имеет свои собственные процедуры (это опять-таки логически разделяет код), но они добровольно отдают управление центральному планировщику, который выполняет их последовательно (это форма [совместной многозадачности](https://en.wikipedia.org/wiki/Cooperative_multitasking)).

[Генераторы](https://en.wikipedia.org/wiki/Generator_(computer_programming)), и они полезны для [потоков](https://en.wikipedia.org/wiki/Stream_(computing)) – особенно для ввода / вывода и для общего обхода структур данных.

[Передача последовательных процессов,](https://en.wikipedia.org/wiki/Communicating_sequential_processes) где каждый подпроцесс является Coroutine. Канальные входы / выходы и операции блокировки дают Coroutines, и планировщик разблокирует их при завершении событий. Альтернативно, каждый подпроцесс может быть родительским для того, который следует за ним в конвейере данных (или предшествует ему, и в этом случае шаблон может быть выражен как вложенные генераторы).

**3 Программная реализация приложения**

Для реализации проекта анализа времени сортировок массивов используется язык программирования Kotlin вместе с библиотеками JMH и Kotlin Coroutines.

Выбор набора библиотек Java Microbenchmark Harness обоснован его максимально точными результатами при тестировании производительности небольших функций (то есть тех, где пауза GC увеличивает время работы в разы).

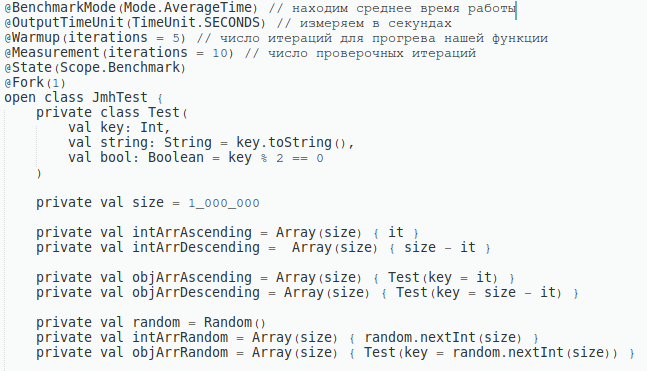
Далее приведена часть кода с полями и аннотациями тестирующего класса (рисунок 1).

Рисунок 1 — Описание массивов для сортировки и аннотации класса

Внутренний приватный класс Test – класс для тестирования сортировок массивов объектов. Поле size – размер тестируемых массивов. Поля intArrAscending, intArrDescending, intArrRandom – массивы чисел (Integer), первый из которых упорядочен по возрастанию, второй по убыванию, а третий заполнен случайными числами. Поля objArrAscending, objArrDescending, objArrRandom – массивы объектов (class Test), ключи первого упорядочены по возрастанию, ключи второго по убыванию, а ключи третьего являются случайными числами.

Для верности получаемых результатов рекомендуется «прогревать» виртуальную машину Java – делать несколько запусков функции перед тестированием с замеркой времени. В моём случае я выбрал 5 разогревающих запусков. Затем выполняется 10 запусков функции, время работы замеряется и выводится среднее значение.

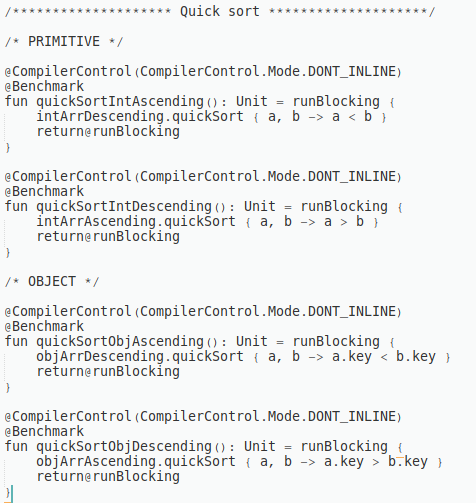
Ниже частично приведена реализация запуска сортировок (рисунок 2).

Рисунок 2 — Частичная реализация запуска сортировок

В функцию qiuckSort, взятую для примера, передаётся лямбда-функция, в которой указан принцип по которому будет проводиться сортировка. Т.е. если массив состоит из объектов, то можно указать один или несколько критериев сортировки.

JMH самостоятельно сделает прогрев, проведёт запуск всех функций и выдаст результаты с уточнением возможной погрешности. На основе полученных вычислений можно проанализировать эффективность той или иной сортировки в различных ситуациях и составить наглядный график.

**ЗАКЛЮЧЕНИЕ**

Цель курсовой работы – исследовать основные способы распараллеливания сортировок массивов и реализовать соответствующую программу – достигнута.

При подготовке к курсовой работе исследовались вопросы, связанные с изучением и применением технологии распараллеливания сортировок массивов, рассмотрены виды сортировок и способы работы многопоточности.

В курсовой работе разработано приложение, позволяющее оценить производительность различных сортировок, и на основе полученных данных в различных ситуациях выбрать более приоритетную.

Разработанную программу можно расширить распараллеливанием других сортировок, таких как, шейкерная, сортировка расчёской и др. Также можно написать интерфейс для программы и результаты приложения использовать в других масштабных проектах, которые работают с большими объёмами данных.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Технология распараллеливания программ [Электронный ресурс]. – URL: https://www.intuit.ru/studies/course/lecture/43256/ (дата обращения : 30 ноября 2018).

2 Технологии распараллеливания [Электронный ресурс]. – URL: http://masters.donntu.org/2009/fvti/turashov/library/book5.html/ (дата обращения : 30 ноября 2018).

3 Сортировки в программировании [Электронный ресурс]. – URL: https://works.doklad.ru/view/Y1jHQRr8B9s.html (дата обращения : 1 декабря 2018).

4 Быстрая сортировка и способы реализации [Электронный ресурс]. – URL: https://habr.com/ru/post/335920/ (дата обращения : 3 декабря 2018).

5 Сортировка слиянием и виды её оптимизации [Электронный ресурс]. – URL: https://works.doklad.ru/view/KhQlCnoNzZc.html (дата обращения : 8 декабря 2018).

6 Сортировка Шелла и способы реализации [Электронный ресурс]. – URL: https://works.doklad.ru/view/Y1jHQRr8B9s.html (дата обращения : 11 декабря 2018).

7 Пирамидальная сортировка и способы её оптимизации [Электронный ресурс]. – URL: https://habr.com/post/337528/ (дата обращения : 13 декабря 2018).

8 Понятие многопоточности и распараллеливания программ [Электронный ресурс]. – URL: https://habr.com/post/277479/ (дата обращения : 20 декабря 2018).

9 Различия между распараллеливанием и многопоточностью [Электронный ресурс]. – URL: https://works.doklad.ru/view/p4pEdEPsXno/2.html (дата обращения : 22 декабря 2018).

10 Язык программирования Kotlin и его преимущества [Электронный ресурс]. – URL:

https://habr.com/post/349914/ (дата обращения : 25 декабря 2018).