**Министерство науки и высшего образования Российской Федерации**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский университет ИТМО**

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

# ЛАБОРАТОРНАЯ РАБОТА №1

## По дисциплине “Технологии программирования”

Выполнил Аронов Данила Алексеевич
(Фамилия Имя Отчество)

Проверил Ивницкий Алексей
(Фамилия Имя Отчество)

Санкт-Петербург, 2022г

# 1. Изучение механизма интероп для Java и C#

## Java

Java Native Interface – инструмент для взаимодействия программ написанных на Java с динамическими библиотеками.

Сперва необходимо описать заголовок метода, пометив его ключевым словом native:

```
public static native int sum(int[] array);
```

Далее используя штатные средства JDK javac нужно построить заголовочные файлы для проекта. В нашем случае для единственного файла.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class InteropMain */

#ifndef _Included_InteropMain
#define _Included_InteropMain
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     InteropMain
 * Method:    sum
 * Signature: ([I)I
 */
JNIEXPORT jint JNICALL Java_InteropMain_sum
  (JNIEnv *, jclass, jintArray);

#ifdef __cplusplus
}
#endif
#endif
```

После создания заголовочного файла можно приступать к написанию кода на низкоуровневом языке.

```cpp
#include <InteropMain.h>

JNIEXPORT jint JNICALL Java_InteropMain_sum(JNIEnv *env, jclass, jintArray jArray) {
    int sum = 0;

    jsize len = env->GetArrayLength(jArray);
    jint *arr = env->GetIntArrayElements(jArray, nullptr);

    for (int i = 0; i < len; i++) {
        sum += arr[i];
    }

    env->ReleaseIntArrayElements(jArray, arr, 0);

    return sum;
}
```

Помимо аргументов в метод так же передаётся информация о среде исполнения. Типы Java имеют свои аналоги в C++ (например, int -> jint). Так же не стоит забывать о переходе к языку без автоматического управления памятью, так что используемые ресурсы необходимо освобождать по окончании решения задачи.

Когда код написан, необходимо собраться его под выбранную платформу как динамическую библиотеку. Полученный файл необходимо подключить в Java, указав название библиотеки (название dll файла без расширения в моём случае):

```java
static {
    System.loadLibrary( libname: "libNative");
}
```

Так при загрузке класса, где размещена эта строка, JRE попытается найти в списке директорий с библиотеками найти соответствующий файл. Этот список директорий необходимо передать как параметр JRE при запуске:

```
-Djava.library.path="Interop/native/cmake-build-debug"
```

В результате выполнения следующего кода:

```java
public static void main(String[] args) {
    int n = 100_000_000;
    int[] numbers = new int[n];

    for(int i = 0; i < n; numbers[i] = i++) {
        numbers[i] = i;
    }

    System.out.println(MeasureUtils.measure(() -> sum(numbers),  count: 5,  label: "JNI sum"));
    System.out.println(MeasureUtils.measure(() -> jSum(numbers),  count: 5,  label: "Java sum"));
}
```

были получены следующие результаты:

```
[DEBUG] JNI sum: took 531 ms
[DEBUG] JNI sum: took 568 ms
[DEBUG] JNI sum: took 489 ms
[DEBUG] JNI sum: took 515 ms
[DEBUG] JNI sum: took 484 ms
[DEBUG] JNI sum: took 496 ms in average
887459712
[DEBUG] Java sum: took 40 ms
[DEBUG] Java sum: took 38 ms
[DEBUG] Java sum: took 36 ms
[DEBUG] Java sum: took 36 ms
[DEBUG] Java sum: took 37 ms
[DEBUG] Java sum: took 36 ms in average
887459712
```

Результат совпал, однако накладные расходы оказались выше, чем «эффективность» C++.

## C#

В случае с C# всё работает несколько проще. Вновь сначала нужно написать код на C++, но при этом используя обычные типы и собрать .dll библиотеку.

```cpp
extern "C" {
int __declspec(dllexport) Sum(int *numbers, int length) {
    int sum = 0;

    for (int i = 0; i < length; i++) {
        sum += numbers[i];
    }

    return sum;
}
}
```

А затем указав с ключевым словом extern и откуда его импортировать, описать метод:

```cpp
public class Program
{
    [DllImport( dllName: "..\\..\\..\\native\\cmake-build-debug\\libNative.dll")]
    public static extern int Sum(int[] array, int length);

    public static void Main(string[] args)
    {
        var n = 100_000_000;
        var numbers = new int[n];

        for (int i = 0; i < n; numbers[i] = i++)
        {
            numbers[i] = i;
        }

        Console.WriteLine(Sum(numbers, n));
    }
}
```

Запуская, получаем ожидаемый результат:

```
887459712

Process finished with exit code 0.
```

## 2. Изучение функциональных языков их специальных возможностей

### Scala

Типичным представителем функционального мира JVM является Scala. Интересной функциональностью является piping – построение цепочек обработки данных. Однако позже это появилось в Java в виде Stream API.

Подсчёт количество использований слов в тексте на Scala и Java:

```
def printTuple(tuple: (String, Int)): Unit = {
  println(f"${tuple._1} -> ${tuple._2}")
}

def printMap(map: Map[String, Int]): Unit = {
  map.foreach(tuple => {
    printTuple(tuple)
  })
}

def main(args: Array[String]): Unit = {
  File(args.apply(0)).lines.toList.flatMap(line => line.split( regex = " ")) : List[String]
    .map(word => word.trim.toLowerCase)                                       : List[String]
    .filter(word => word.matches( regex = "^[a-яё]+$"))                       : List[String]
    .groupBy(word => word)                                                    : Map[String, List[String]]
    .map(word => word._1 -> word._2.length)                                   : Map[String, Int]
    .tapEach(tuple => printTuple(tuple))                                      : Unit
}
```

```
/**
 * Read file args[0] and count number of each word
 *
 * @param args
 * @throws Exception
 */
public static void main(String[] args) throws Exception {
    (new ScalaMain()).main(args);
    System.out.println("================================================================================");
    (new BufferedReader(new FileReader(args[0]))).lines() Stream<String>
            .flatMap(line -> Arrays.stream(line.split( regex: " ")))
            .map(String::trim)
            .filter(word -> word.matches( regex: "^[a-яё]+$"))
            .collect(HashMap::new, (map, str) -> map.put(str, (Integer) map.getOrDefault(str, defaultValue: 0) + 1),
                    (a, b) -> {}) HashMap<Object, Object>
            .forEach((key, value) -> System.out.println(key + " " + value));
}
```

Scala работает на JVM и компилируется в .class файлы, как java. Если декомпилировать, мы увидим императивный код:

```
@ScalaSignature(
   bytes = "\u0006\u0005i2A!\u0002\u0004\u0001\u0013!)\u0001\u0003\u0001C\u0001#!)A\u0003\u0001C\u0001+!
)
public class ScalaMain {
   public void printTuple(final Tuple2 tuple) {
       var10000 = .MODULE$;
      Object arg$macro$1 = tuple._1();
      Object arg$macro$2 = BoxesRunTime.boxToInteger(tuple._2$mcI$sp());
      var10000.println(scala.collection.StringOps..MODULE$.format$extension("%s -> %s", scala.runtime.Sc
   }

   public void printMap(final Map map) {
      map.foreach((tuple) -> {
         $anonfun$printMap$1(this, tuple);
         return BoxedUnit.UNIT;
      });
   }

   public void main(final String[] args) {
      scala.reflect.io.File..MODULE$.apply(scala.reflect.io.Path..MODULE$.string2path(args[0]), scala.io
         return .MODULE$.wrapRefArray((Object[])line.split(" "));
      }).map((word) -> {
         return word.trim().toLowerCase();
      }).filter((word) -> {
         return BoxesRunTime.boxToBoolean($anonfun$main$3(word));
      }).groupBy((word) -> {
         return word;
      }).map((word) -> {
         return scala.Predef.ArrowAssoc..MODULE$.$minus$greater$extension(.MODULE$.ArrowAssoc(word._1()),
      }).tapEach((tuple) -> {
         $anonfun$main$6(this, tuple);
         return BoxedUnit.UNIT;
      });
   }
}
```

F#

F# также имеет в своём арсенале различные функциональные возможности.

С помощью Computation Expression, например, можно функционально инициализировать последовательности чисел.

```
let numbers = seq [
    for i in 1 .. 5 -> i * i - 1
]
val numbers: seq<int> = [0; 3; 8; 15; 24]
```

Discriminated Union позволяет определить набор полиморфных типов:

```
type public OptionalField=
    | OptionalString of value : string
    | OptionalInteger of value : int
```

Затем их можно использовать из C#^

```
public static void Main(string[] args)
{
    Types.OptionalField integerField = Types.OptionalField.NewOptionalInteger(1);
    Types.OptionalField stringField = Types.OptionalField.NewOptionalString("Hello");

    Console.WriteLine(integerField);
    Console.WriteLine(stringField);
}
```

```
OptionalInteger 1
OptionalString "Hello"
```

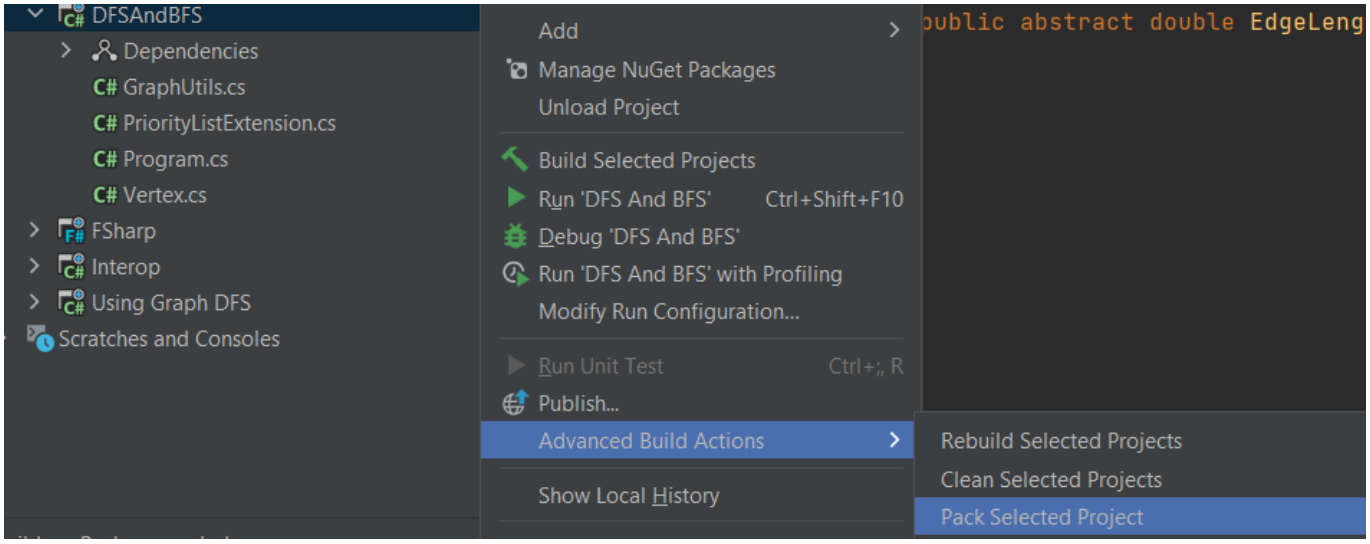
Синтаксис языка позволяет определить Pipe Operator:

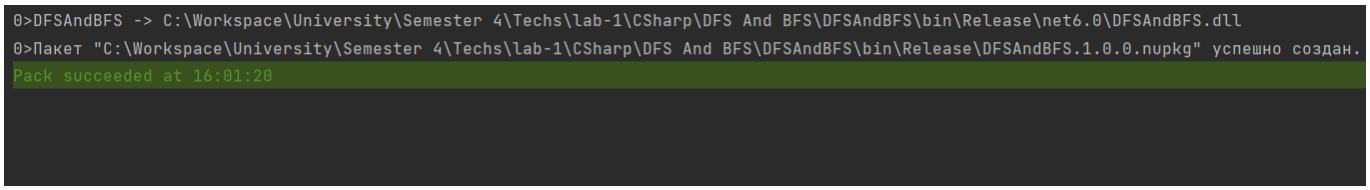
```
let (|>) x f = f x
let increase x = x + 1
let double x = x * 2

5 |> increase |> increase |> double |> double |> double
val (|>) : x: 'a -> f: ('a -> 'b) -> 'b
val increase: x: int -> int
val double: x: int -> int
val it: int = 56
```

## 3. Публикация собранных пакетов библиотек для Java и C#

### C#

Rider имеет встроенные инструменты для выполнения данной операции. Необходимо выбрать проект, из которого будет собран проект и выбрать необходимый пункт в контекстном меню:



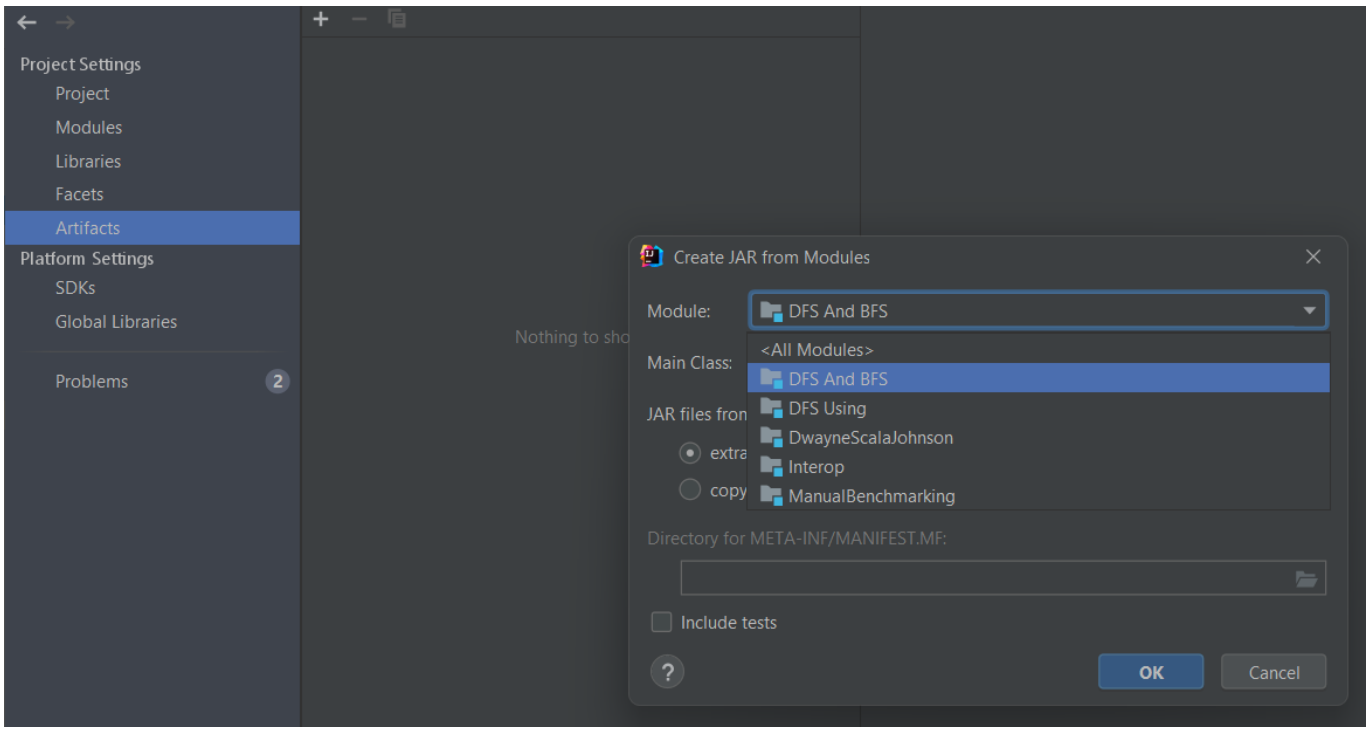




Так сформировался NuGet пакет по указанному пути.

Чтобы его использовать нужно добавить в Rider локальный репозиторий, указав папку, где лежит .nupkg файл или положить в другую папку, которая используется в качестве локального репозитория.

### Java

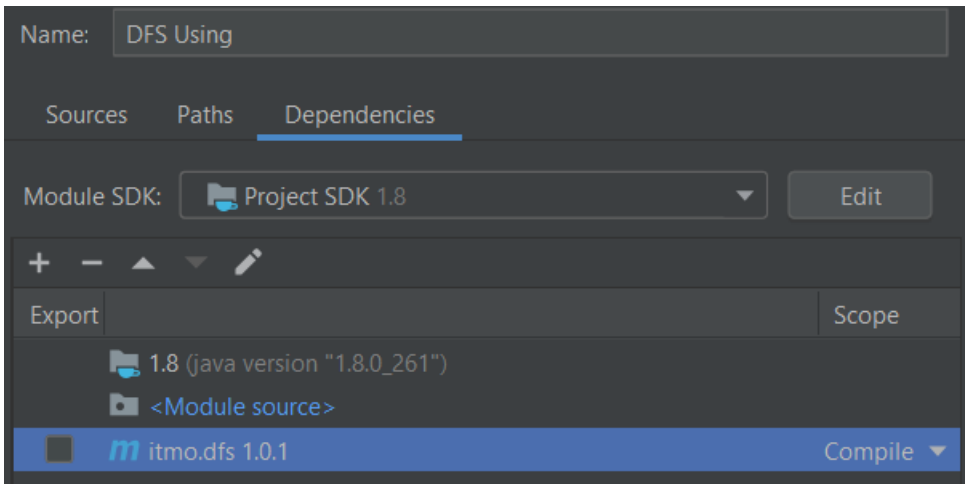
Пользуясь IntelliJ, можно собрать артифакт .jar пакет и дальнейшую работу проводить с ним.




Полученный jar файл можно установить в локальный maven репозиторий:

```
PS C:\Workspace\University\Semester 4\Techs\lab-1\Java\out\artifacts\DFS_And_BFS_jar> mvn
 install:install-file -Dfile='DFS And BFS.jar' -DgroupId='itmo' -DartifactId='dfs' -Dvers
ion='1.0.1' -Dpackaging=jar -DgeneratePom=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing C:\Workspace\University\Semester 4\Techs\lab-1\Java\out\artifacts\DFS_A
nd_BFS_jar\DFS And BFS.jar to C:\Users\danil\.m2\repository\itmo\dfs\1.0.1\dfs-1.0.1.jar
```

И указать в зависимостях в другом модуле



## 4. Анализ инструментов для оценки производительности в C# и Java.

В следующих тестах будет проверяться производительность тривиальной реализации пузырьковой сортировки и сортировки, предоставляемой стандартными библиотеками языка. Предварительно создаётся массив из 10 000 элементов и затем вызывается операция его сортировки.

### C#

Для C# существует NuGet BenchmarkDotNet. Необходимо пометить процедуры, которые будут протестированы атрибутом [Benchmark] и в Main программы запустить проверку производительности.

```
public static void Main(string[] args)
{
    var summary = BenchmarkRunner.Run<Benchmark>();
}
```

```
[Benchmark]
public void BubbleSort()
{
    var changes = 1;
    while (changes > 0)
    {
        changes = 0;
        for (var i = 1; i < _data.Length; i++)
        {
            if (_data[i - 1] > _data[i])
            {
                (_data[i], _data[i - 1]) = (_data[i - 1], _data[i]);
                changes++;
            }
        }
    }
}

[Benchmark]
public void SharpSort()
{
    Array.Sort(_data);
}
```

Результаты представляются в терминале:

```
|     Method |         Mean |       Error |      StdDev |
|----------- |-------------:|------------:|------------:|
| BubbleSort | 192,628.0 us | 3,614.73 us | 4,700.17 us |
|  SharpSort |     630.1 us |    16.45 us |    45.58 us |
```

Так мы видим, что стандартная сортировка быстрее пузырьковой в 305 раз, что ожидаемо теоретически. Асимптотически разница должна быть около 100 раз O(n^2) против O(n log n).

## Java

Для анализа производительности программ в Java существует библиотека Java Microbenchmark Harness. Она подключается в качестве Maven зависимости

```xml
<properties>
    <jmh.version>1.28</jmh.version>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencies>
    <dependency>
        <groupId>org.openjdk.jmh</groupId>
        <artifactId>jmh-core</artifactId>
        <version>${jmh.version}</version>
    </dependency>
    <dependency>
        <groupId>org.openjdk.jmh</groupId>
        <artifactId>jmh-generator-annprocess</artifactId>
        <version>${jmh.version}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Далее аналогично C# здесь нужно пометить процедуры для проверки аннотацией @Benchmark. Так же с помощью аннотаций настраиваются параметры тестирования: количество прогонов, количество потоков, проверка длительности операции или её частота и так далее.

```
@State(Scope.Thread)
@Fork(value = 1)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@Warmup(iterations = 1, time = 1000, timeUnit = TimeUnit.MILLISECONDS)
@Measurement(iterations = 1, time = 1000, timeUnit = TimeUnit.MILLISECONDS)
```

```
@Benchmark
public static void setup() { data = randomArray( size: 10_000); }

public static double[] randomArray(int size) {...}

@Benchmark
public static void bubbleSort() {
    setup();
    int changes = 1;
    while (changes > 0) {
        changes = 0;
        for (int i = 1; i < data.length; i++) {
            if (data[i - 1] > data[i]) {
                double tmp = data[i];
                data[i] = data[i - 1];
                data[i - 1] = tmp;
                changes++;
            }
        }
    }
}

@Benchmark
public void javaSort() {
    setup();
    Arrays.sort(data);
}
```

| Benchmark | Mode | Cnt | Score | Error | Units |
|---|---|---|---|---|---|
| Bench.bubbleSort | avgt | | 177,144 | | ms/op |
| Bench.javaSort | avgt | | 0,755 | | ms/op |
| Bench.setup | avgt | | 0,228 | | ms/op |

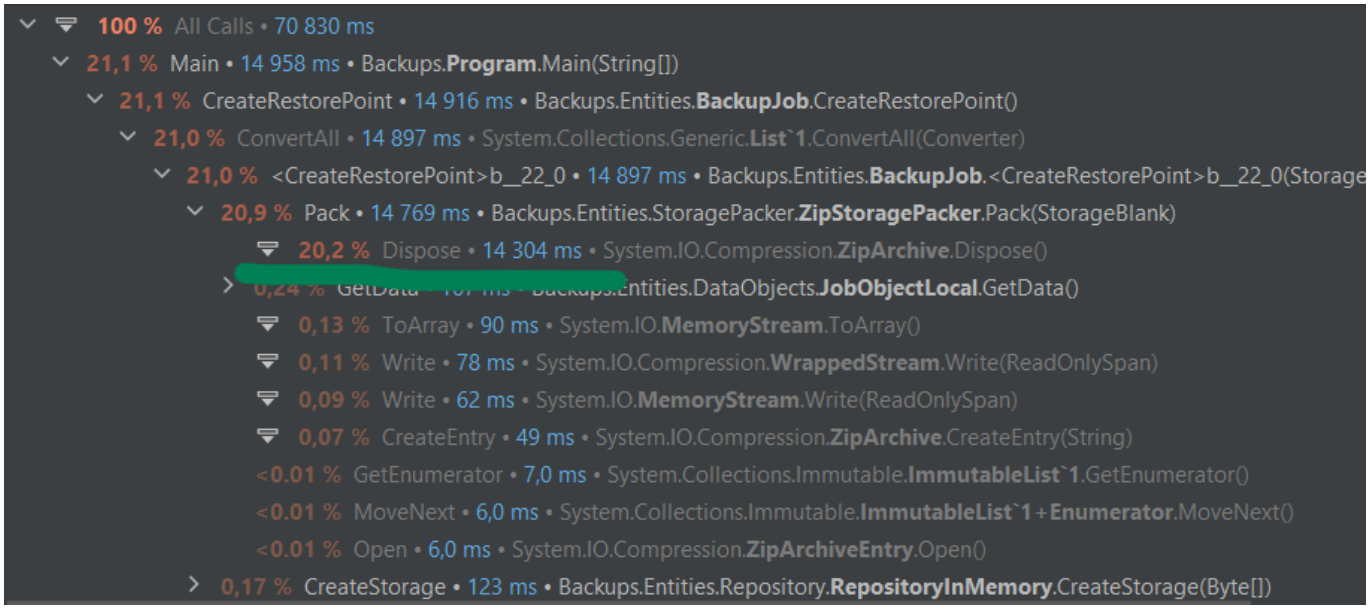
Как и в первом случае стандартная сортировка быстрее пузырьковой в 335 раз.

## 5. Трассировка памяти и производительности проекта Backups

В качестве тестовой нагрузки для создания резервных копий использовались 2 незначительных текстовых файла и 1 pdf документ размером 14 Мбайт. Для теста производительности создавалось 20 резервных точек, для теста памяти – 200.

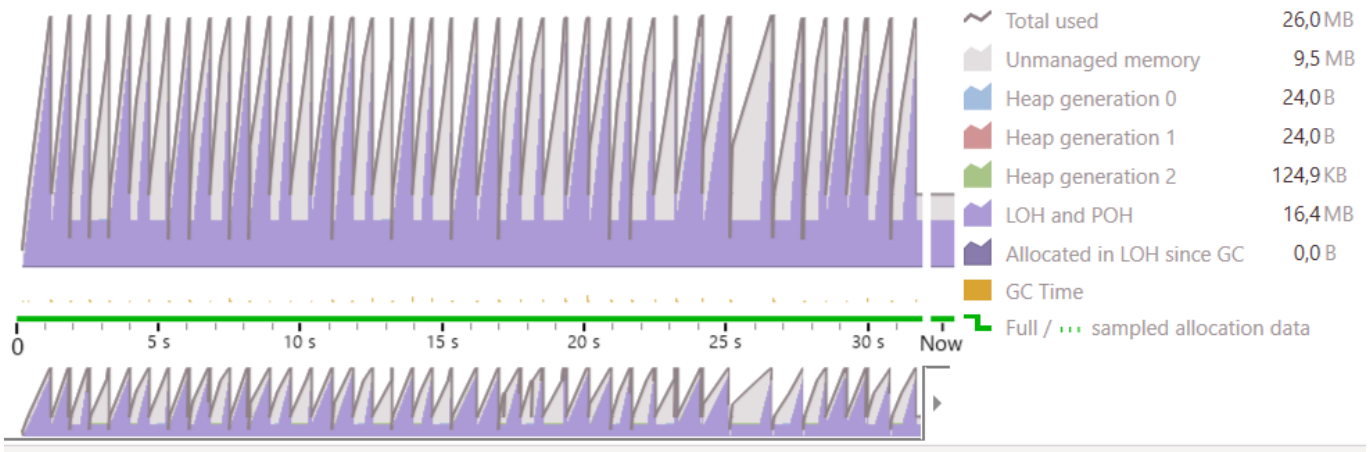
### Данные сохраняются на локальный диск

Данные dotTrace:




Как видно из скриншота, большая часть времени исполнения программы уходит на сжатие данных.

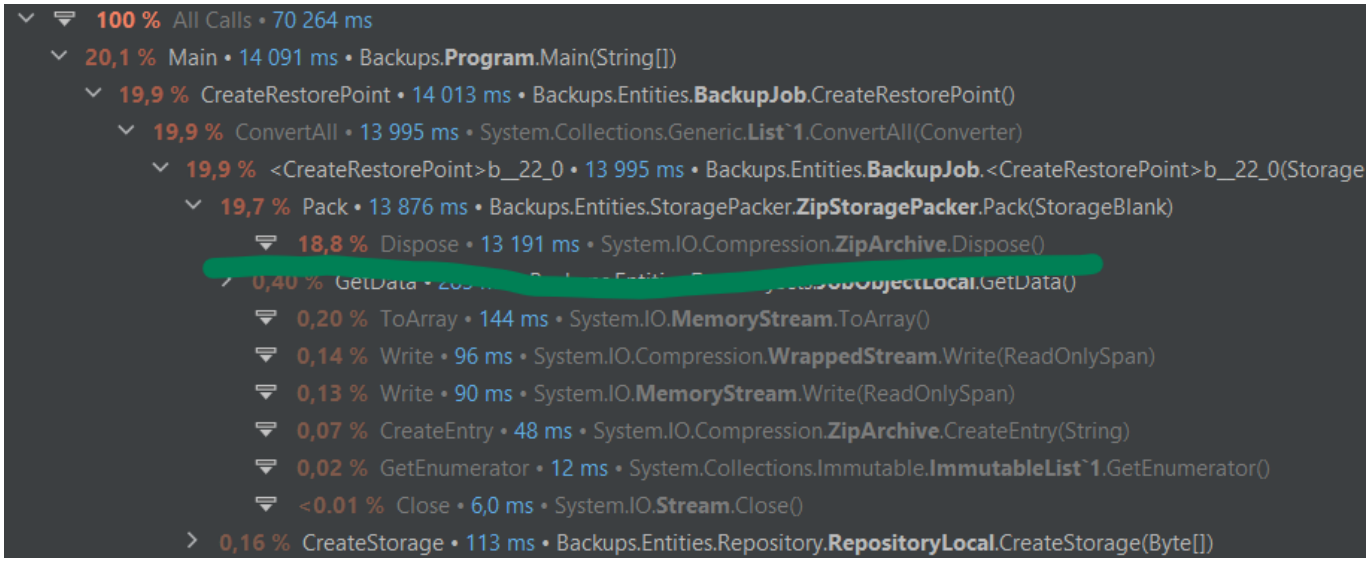
Данные dotMemory:




Утечек памяти не обнаружено.

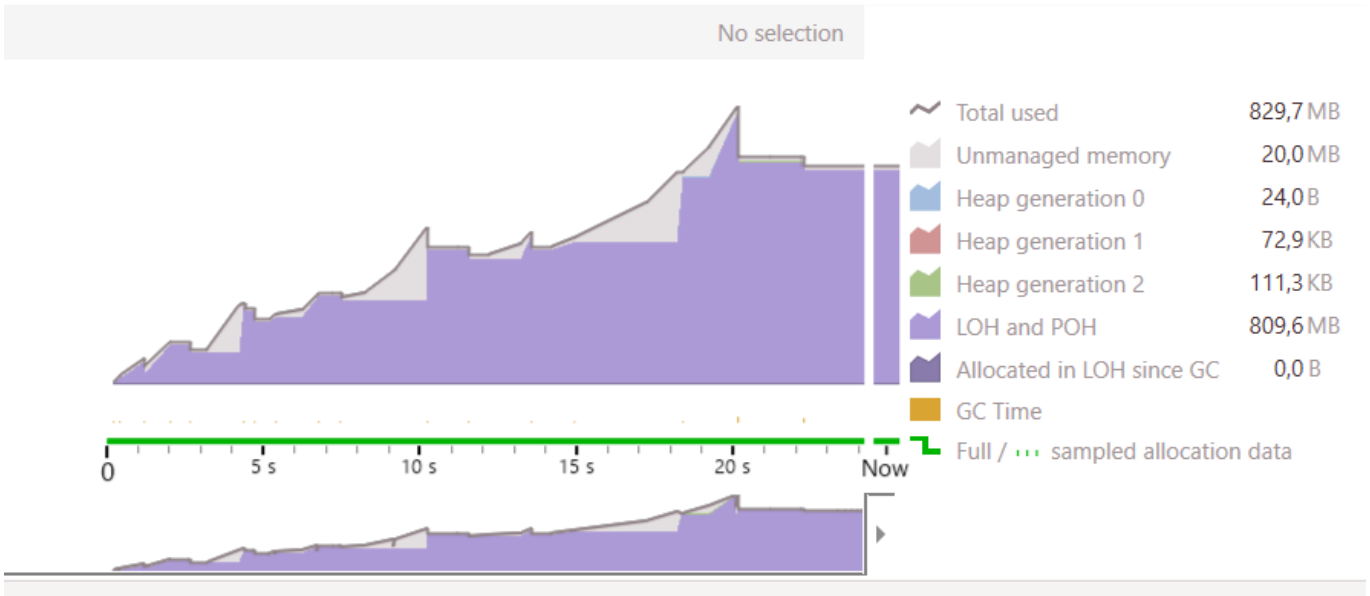
Данные сохраняются в оперативную память

Данные dotTrace:




Вновь большая часть времени уходит на сжатие данных.

Данные dotMemory:




Как и ожидалось, память, используемая приложением, постоянно увеличивается в связи с увеличением количества точек сохранения.