Начнем отчет с параметров системы:
Вот характеристики моего ноутбука:
Процессор AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
Оперативная память 8,00 ГБ (доступно: 7,34 ГБ)
Тип системы 64-разрядная операционная система, процессор x64
Выпуск Windows 11 Домашняя для одного языка
Версия 22H2
Сборка ОС 22621.2861
Взаимодействие Windows Feature Experience Pack 1000.22681.1000.0

Теперь напишем параметры виртуальной машины (тут все по классике):
ОС: Ubuntu (64 - bit)
Версия: 22.04.3
Процессоры: 1 - 2
Предел загрузки ЦПУ: 100%

Тепрь перейдем к эксперементам:
Сами скрипты я буду помещать в отчет для вашего удобства, чтобы Вам было удобнее проверять.

```
#!/bin/bash

count=0

if [[ $# -eq 1 ]]; then
  count=$1
fi

limit=$((count % 350000 + 700000))

result=0

for ((i = 1; i <= limit; i++)); do
  ((result++))
done
```

Теперь мы создаем как требуется в условии 20 файлов от n1 до n20, которые содержат от 1 до 20 случайных чисел, для этого я написал такой скрипт для заданного диапозона:

```
#!/bin/bash


if [[ $# -ne 2 ]]; then
  echo "Использование: $0 <минимальное_значение> <максимальное_значение>"
  exit 1
fi

min=$1
max=$2
```

```
random_number=$((RANDOM % (max - min + 1) + min))

echo "Случайное число в диапазоне от $min до $max: $random_number"
```

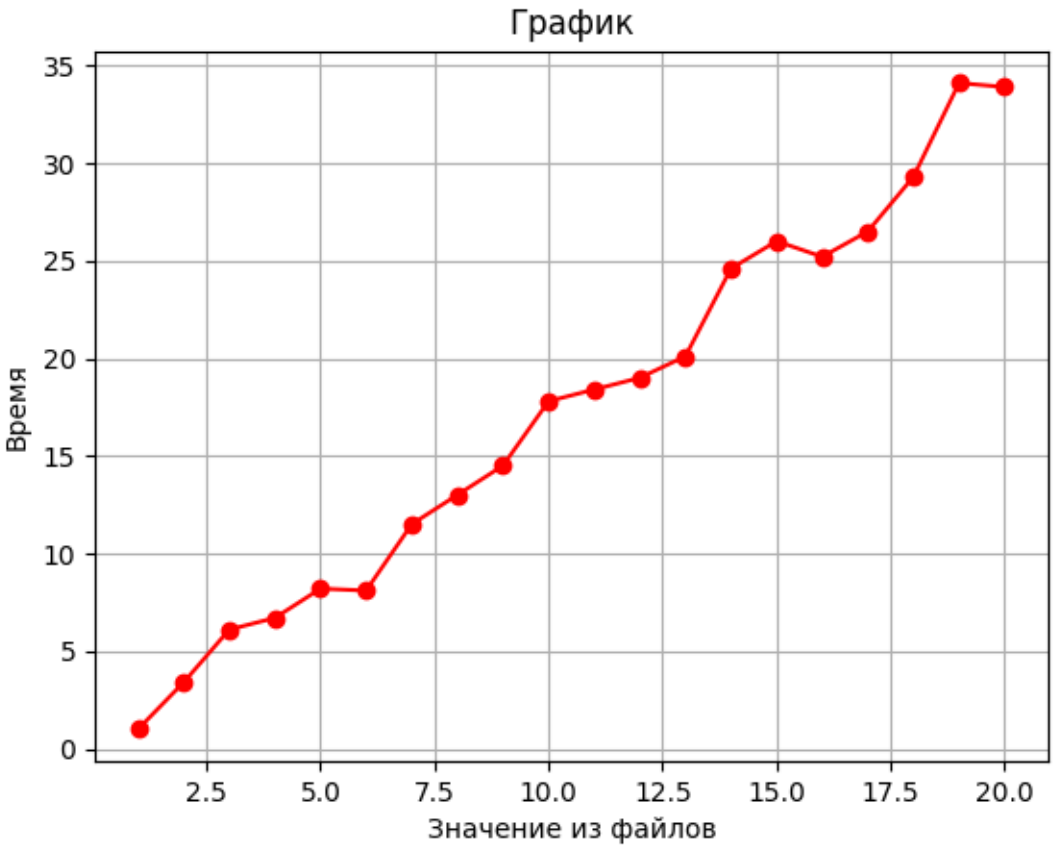
Тепреь надо запускать начальный скрипт со всеми числами из файла для этого используем такой скрипт:

```
#!/bin/bash
        while read line
        do
bash baseScript.sh $line
        done < $1
```

```
#!/bin/bash

MAX_LOG_FILES=20
MAX_ITERATIONS=10

for (( log_number=1; log_number <= $MAX_LOG_FILES; log_number++ ))
do
   log_file="log$log_number"

   if [[ -f "$log_file" ]]
   then
      rm "$log_file"
   fi

   for (( iteration=1; iteration <= $MAX_ITERATIONS; iteration++ ))
   do
      output_file="example"
      input_file="n$log_number"

      sudo time -o "$output_file" -f "%e" bash gettingLogs.sh "$input_file"

      echo -n "$(cat "$output_file")," >> "$log_file"
   done

   sed -i 's/,$//' "$log_file"
done
```

После запуска этого скрипта я получаю время и руками считал среднее значение (можно было и через скрипт, но и так быстро,тк известно количество аргументов и есть данные, которые можно копировать)

Теперь строим график результатов:



Вот данные из графика:

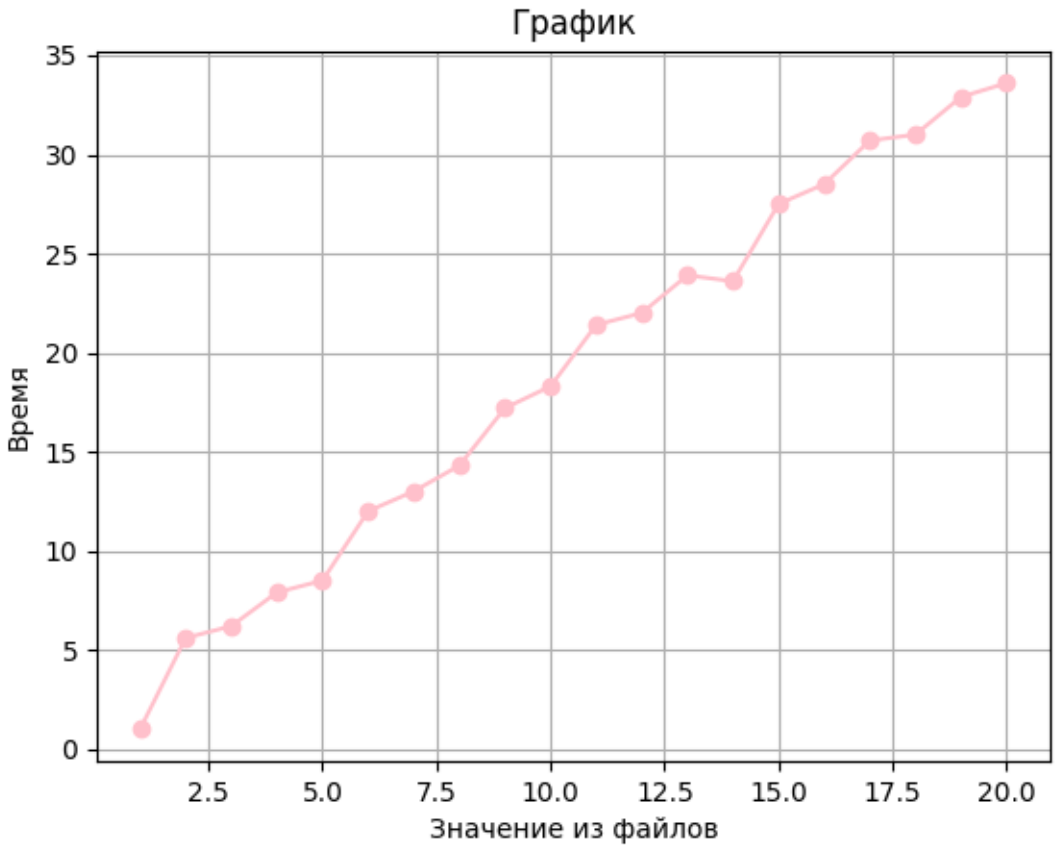
```
time_values = [1, 3.4, 6.1, 6.7, 8.2, 8.1, 11.5, 13, 14.5, 17.8, 18.4, 19, 20.1, 24.6, 26, 25.2, 26.5, 29.3, 34.1, 33.9]
value_counts = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Теперь я использовал этот скрипт для запуска всех сразу, используем запуск черз PID тмпа того как делали в 4 лабе и ко всем применяем 1 скрипт *baseScript.sh*:

```
#!/bin/bash
declare -a process_ids=()

while read input_line
do
  bash baseScript.sh "$input_line" &
  process_ids+=($!)
done < "$1"

for process_id in "${process_ids[@]}"
do
  wait "$process_id"
done
```

Получим похожий график,но как будто более равномерный:

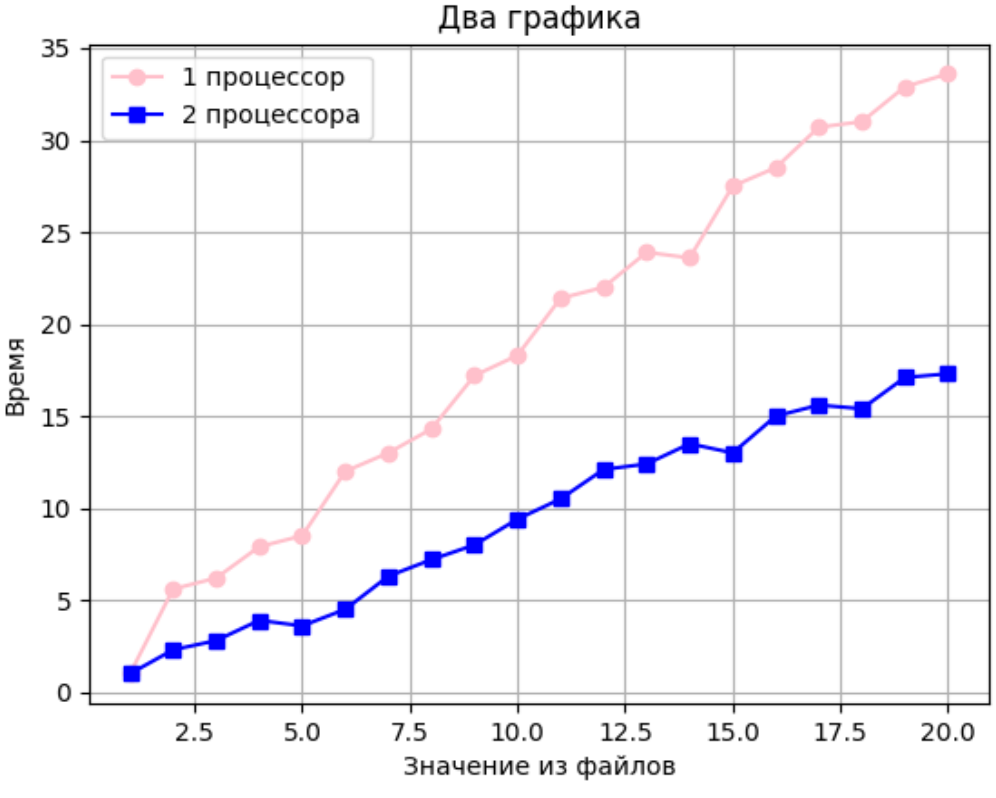
```
time_values = [1,5.6,6.2,7.9,8.5,12,13,14.3,17.2,18.3,21.4,22,23.9,23.6,27.5,28.5,30.7,31,32.9,33.6]
value_counts = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```



Вот данные:

```
time_values = [1,5.6,6.2,7.9,8.5,12,13,14.3,17.2,18.3,21.4,22,23.9,23.6,27.5,28.5,30.7,31,32.9,33.6]
value_counts = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Тепрерь посмотрим как изменился график при запуске с 2 процессорами:



Как можно заметить, это достаточно сильно влияет на скорость исполнения (плюс могло повлиять то, что сторонних процессов на момент запуска с 2 процессорами стало меньше)

вот данные нового графика:

```
time_values_2 = [1, 2.3, 2.8, 3.9, 3.6, 4.5, 6.3, 7.2, 8, 9.4, 10.5, 12.1, 12.4, 13.5, 13, 15, 15.6, 15.4, 17.1, 17.3]
```

С точки зрения теории это можно объяснить тем, что теперь процессоры могли выполнять какие-то скрипты одновременно, что облегчает нагрузку на один процессор и задача выполнится быстрее.

Теперь перейдем ко второй серии эксперементов:

Напивем как надо в условии скрипт,который удваивает каждую строку в файле и записывает результат обратно в файл и проверяет, чтобы не обработать более строк, чем есть в файле, и завершаем выполнение со статусом 0 (успешно), если это условие выполняется.

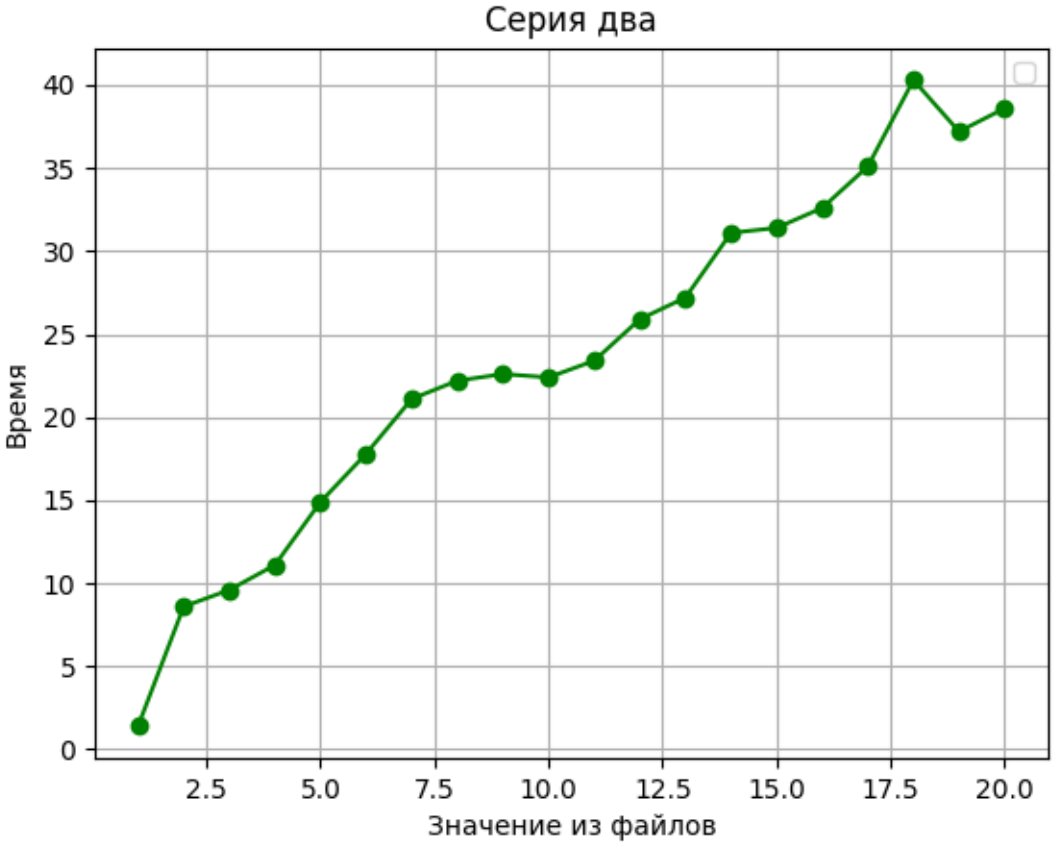
.
Получается такой основной скрипт:

```
#!/bin/bash

file_path=$1
total_lines=$(wc -l < "$file_path")
current_line=0

while read -r current_value
do
  let current_line=$current_line+1

  if [[ $current_line -gt $total_lines ]]
  then
    exit 0
  fi

  doubled_value=$(( $current_value * 2 ))
  echo $doubled_value >> "$file_path"

done < "$file_path"
```

Теперь используем этот скрипт как основной, и аналогично прошлой серии эксперементов, только теперь изменяем скрипт создания файлов

так чтобы в начале было не только создание, но и обновления после работы нашего основного скрипта, скрипты для последующего поочередного и параллельного запуска почти такие же как в прошлый раз,поэтому не буду включать его в отчет,чтобы не загромождать его , а лучше перейдем к самому интересному — графикам:

Вот график с поочерёдным запуском и одним процессором:

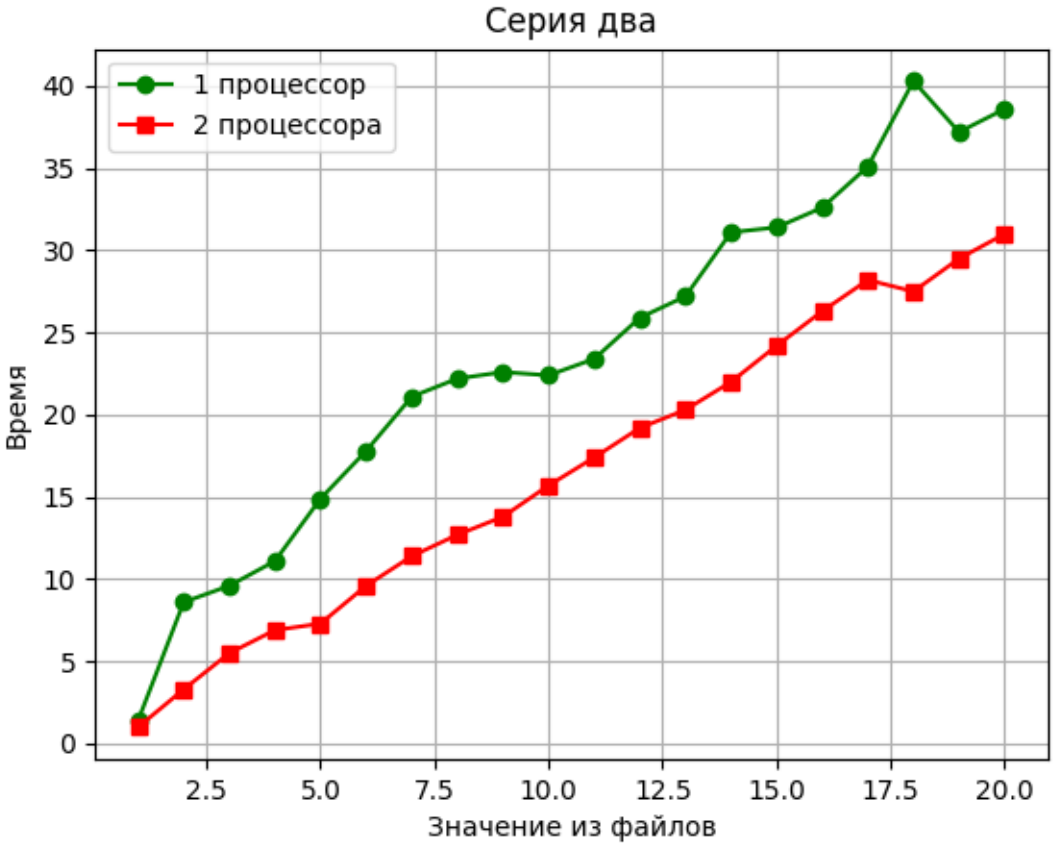


И вот его данные:

```
time_values_1 = [1.4,8.6,9.6,11.1,14.9,17.8,21.1,22.2,22.6,22.4,23.4,25.9,27.2,31.1,31.4,32.6,35.1,40.3,37.2,38.6]
value_counts = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Теперь рассмотрм аналогичным запуском, но двумя процессорами:

Теперь рассмотрим полученный шрафик при параллельном запуске,тоже с одним процессором:

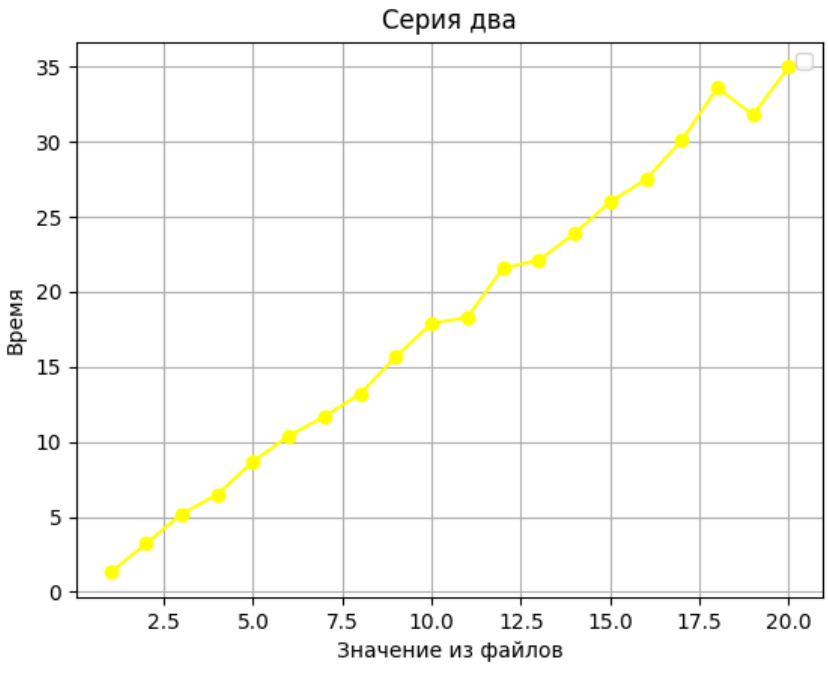


Как можно заметить скорость не сильно меняется и остается почти такой же, (в отличии от случая с паралельным запуском), что вполне ожидаемо с точки зрения теории так как при таком запуске второй процессор не особо поможет и нагрузка меняется совсем немного, хотя это уже помогает.
Данные второго графика:

```
time_values_2 = [1,3.3,5.5,6.9,7.3,9.6,11.4,12.7,13.8,15.7,17.4,19.2,20.3,22,24.2,26.3,28.2,27.5,29.5,31,]
```

Тепрь сравним с тем, что получается при параллельном запуске:

Вот график работы с одним процессором:

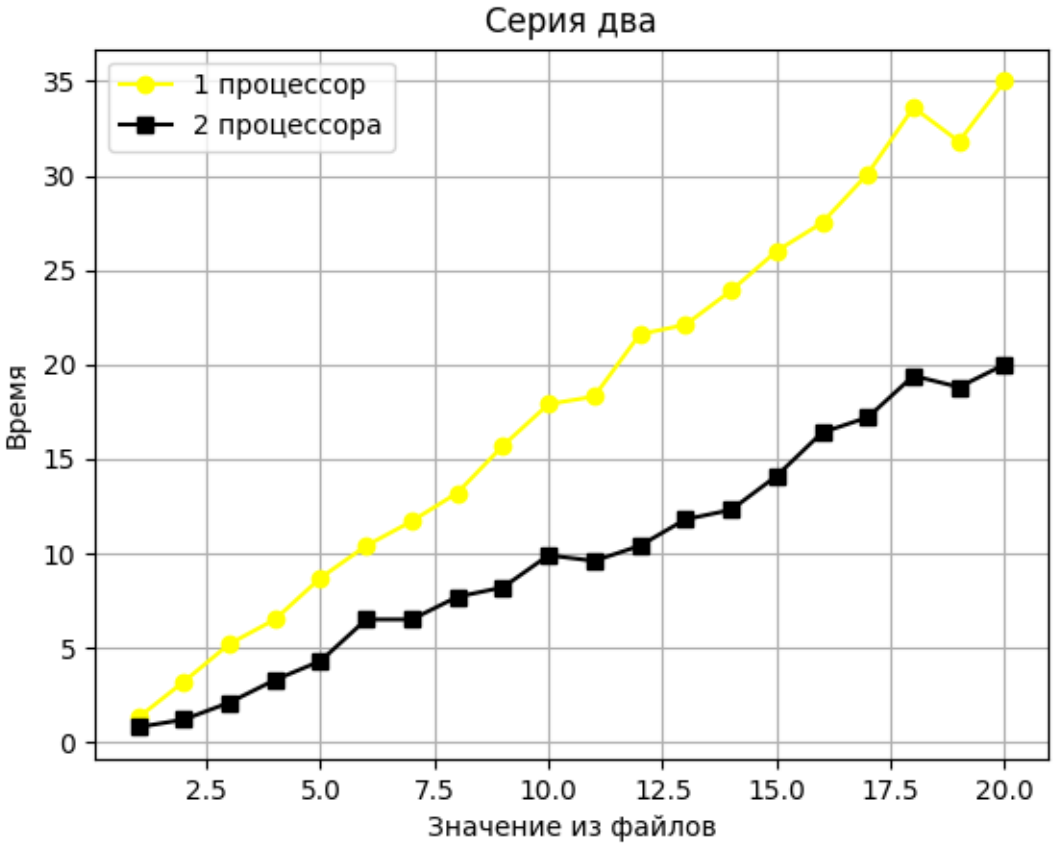


Вот данные:

```
time_values_1 = [1.3,3.2,5.2,6.5,8.7,10.4,11.7,13.2,15.7,17.9,18.3,21.6,22.1,23.9,26,27.5,30.1,33.6,31.8,35]
value_counts = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

А теперь рассмотрим параллельный запуск, но с 2 процессорами:

Получим такой график (с расцветкой Бэтмена):



Вот тут уже значение сильно отличается почти в 2 раза!
Данные нового графика:

```
time_values_2 = [0.8,1.2,2.1,3.3,4.3,6.5,6.5,7.7,8.2,9.9,9.6,10.4,11.8,12.3,14.1,16.4,17.2,19.4,18.8,20]
```

Вывод, который мы можем сделать, это то, что поведение виртуальной машины с заданным числом процессоров близко к ожидаемому в теории, за исключением редких неожиданных скачков. Также можно заметить, что при параллельном запуске с 2 процессами различие очень критично, и уменьшает время почти в 2 раза, и это в особенности эффективно на

большом объеме данных.В отличии от последовательного запуска, где результаты меняются не сильно.

Также тк это последняя лаба, хотел сказать спасибо за оперативную проверку, интересный курс и полезные советы по улучшеню скриптов!