**Universidade Federal da Paraíba (UFPB)**
**Centro de Ciências Aplicadas e Educação (CCAE)**
**Curso**: Bacharelado em Sistemas de Informação
**Disciplina**: Avaliação de Desempenho de Sistemas
**Professor**: Marcus Carvalho

**Aluno:**

# Prática 1 – Medição

Nesta atividade, dois problemas de medição são apresentados. Você deve executar os programas fornecidos e avaliar o desempenho dos mesmos, a partir das métricas medidas pelos programas. Para responder às questões, você deve escrever um relatório com as suas análises.

Deve ser enviado através do Google Classroom este documento, contendo as respostas das questões com as análises, além de um arquivo ZIP com os dados coletados na medição dos programas (arquivos .txt gerados pelos programas). É **fortemente recomendado** o uso de gráficos nos relatórios, para a apresentação dos resultados e para ajudar na sua análise. Tabelas também podem ser usadas para exibir os dados quando necessário.

## Problema 1: Procurando números primos

**Visão geral:** o objetivo deste problema é avaliar o desempenho de um programa que busca a quantidade de números primos que existem entre 1 e um número X informado como entrada. Queremos saber como o desempenho do sistema varia à medida que mudamos os seus parâmetros de entrada.

**Descrição do programa:** o programa Java, chamado BenchmarkNumerosPrimos, recebe como parâmetros de entrada o número de *threads* desejada para executá-lo e o número máximo a ser buscado *X*. O programa irá procurar a quantidade de números primos que existem entre 1 e X.

Exemplo de linha de execução do programa no Linux:

```
java -cp bin/:lib/* BenchmarkNumerosPrimos 4 90000
```

Exemplo de linha de execução do programa no Windows:

```
java -cp bin\;lib\* BenchmarkNumerosPrimos 4 90000
```

Neste exemplo, o programa executará usando 4 *threads* e buscará a quantidade de primos entre 1 e 110000. Cada thread irá fazer a busca de primos para um sub-conjunto da faixa de números, de modo que a carga seja dividida por todas elas.

**Saída do programa:** Para cada execução do programa, dois arquivos serão criados contendo os resultados das medições feitas sobre o seu desempenho, os quais serão localizados no diretório "output-bench-primos/" criado no diretório de execução. Os dois arquivos criados são:

- **measurements-mn?-nt?-st?.txt**: Contém informações sobre a execução e as medições com relação à execução. Possui as seguintes colunas:
    - NumThreads: número de threads especificado na execução.
    - NumeroMaximo: número especificado para buscar primos em [1, NumeroMaximo]
    - StartTime: timestamp que registra o início da execução.
    - ElapsedTime: tempo total gasto na execução, em milissegundos.
    - MeanUtilization: utilização média de CPU durante toda a execução.

Nos nomes dos arquivos há identificações do tipo mn?-nt?-st?, onde o valor seguido de "mn" significa o número máximo especificado para a busca, "nt" é o valor do número de threads especificado e "st" registra o timestamp inicial da execução, para diferenciar execuções iguais.

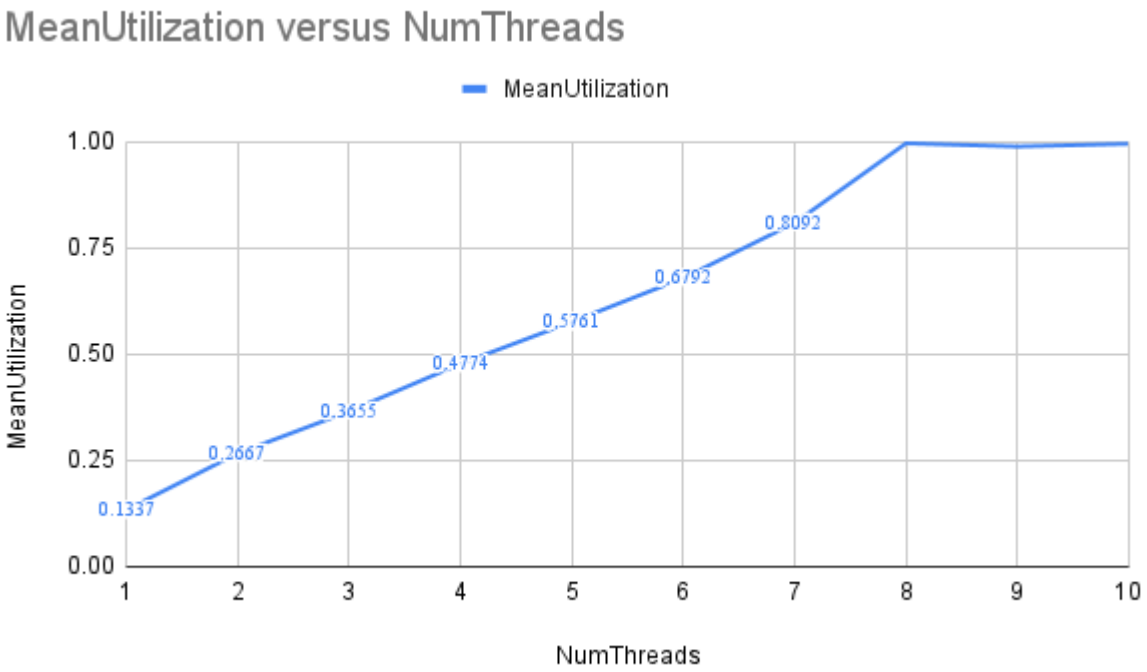
Com base nas informações sobre o programa, responda no relatório as questões abaixo:

## Questões

0. Informe a configuração da máquina na qual você está executando esta avaliação de desempenho, com informações sobre a CPU, Memória e Disco da mesma.

Intel(R) Core(TM) i3-10100T CPU @ 3.00GHz  2.90 GHz
16 GB de Memória
SSD de 240 GB de Disco

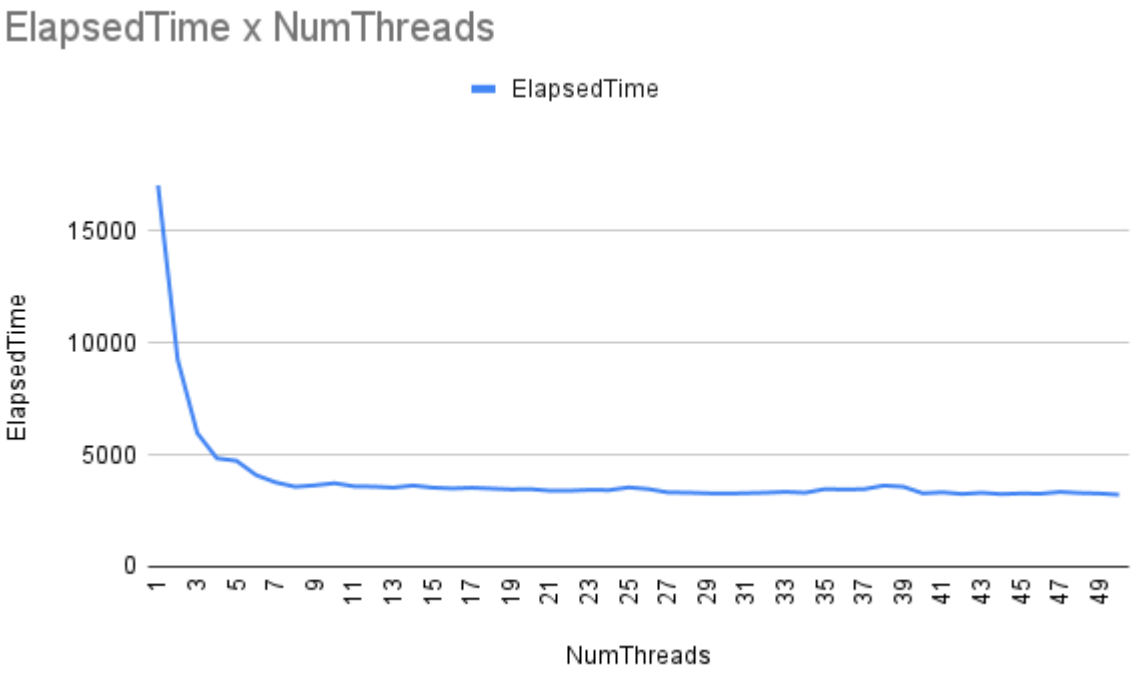
1. Primeiro, queremos saber o impacto do uso de *multithreading* no desempenho. Execute o programa para valores diferentes de número de threads (*NumThreads*). Por exemplo, para número de threads = {1, 2, 3, ..., 10, ...} (não se restrinja a estes valores). Para esta questão, execute o programa para buscar a quantidade de primos com o valor de número máximo = 90000 (valor default). Faça as seguintes análises:
    a. Como o aumento da quantidade de threads afeta a utilização de CPU? Analise os resultados e discuta possíveis causas deste comportamento.




Observando o gráfico, podemos observar que quanto maior o números de Threads usadas maior é o uso da CPU, podemos observar que com 1 thread o uso da CPU é de apenas 13%, está usando pouco a CPU ainda, também observamos que de 8 threads para

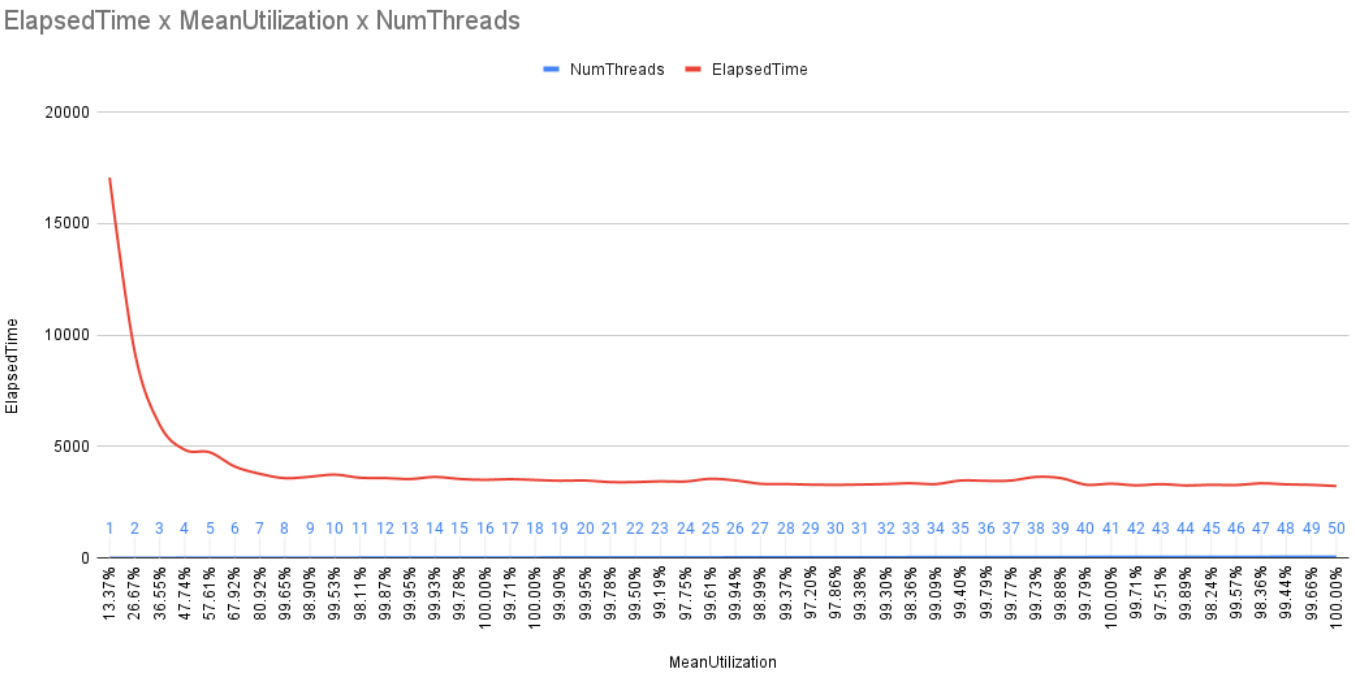
cima está usando quase 100% da CPU deixando ela sobrecarregada, isso acontece por que a CPU tem que ficar alternando entre os números de Threads com mais frequência.

b. Como o aumento da quantidade de threads afeta o tempo total de execução? Analise os resultados e discuta possíveis causas deste comportamento.



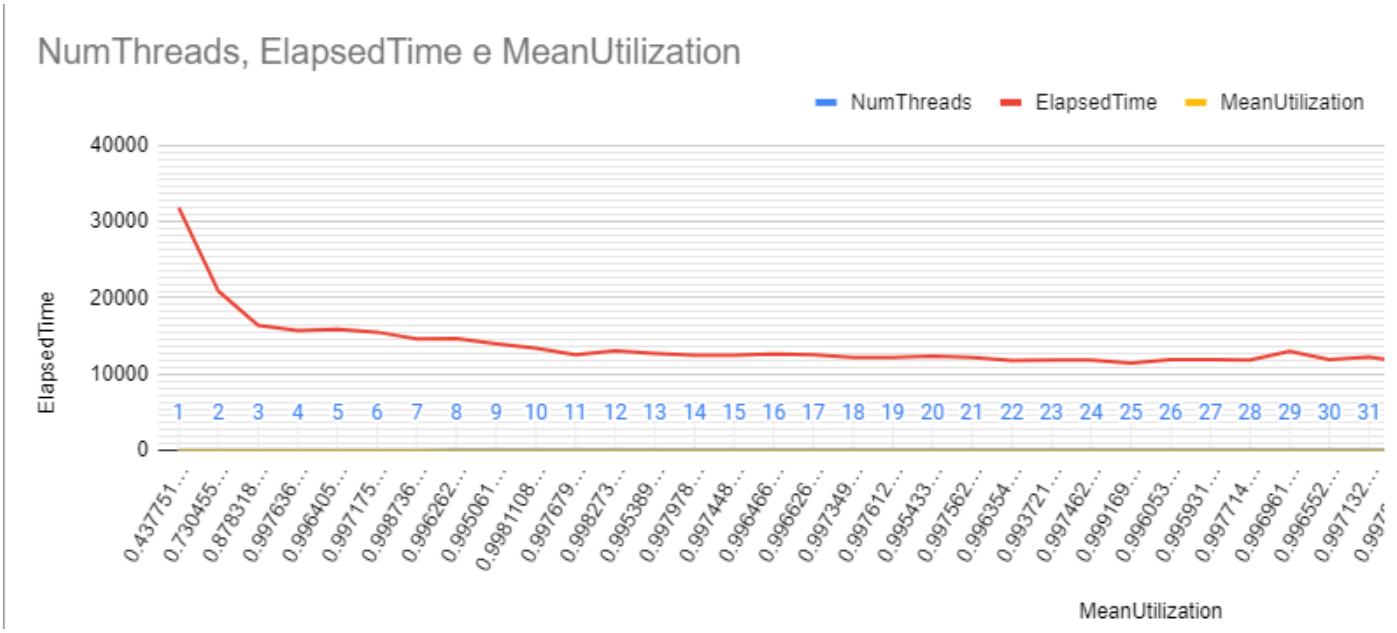

Podemos observar que quanto maior o números de Threads menor é o tempo total de execução, isso acontece pois a execução das tarefas está sendo dividida em números de Threads, assim gastando menos tempo.

c. Qual número de threads você acha mais adequado para executar este programa? Você acha que sua escolha poderia ser diferente caso você executasse o mesmo programa em uma máquina diferente? Justifique.



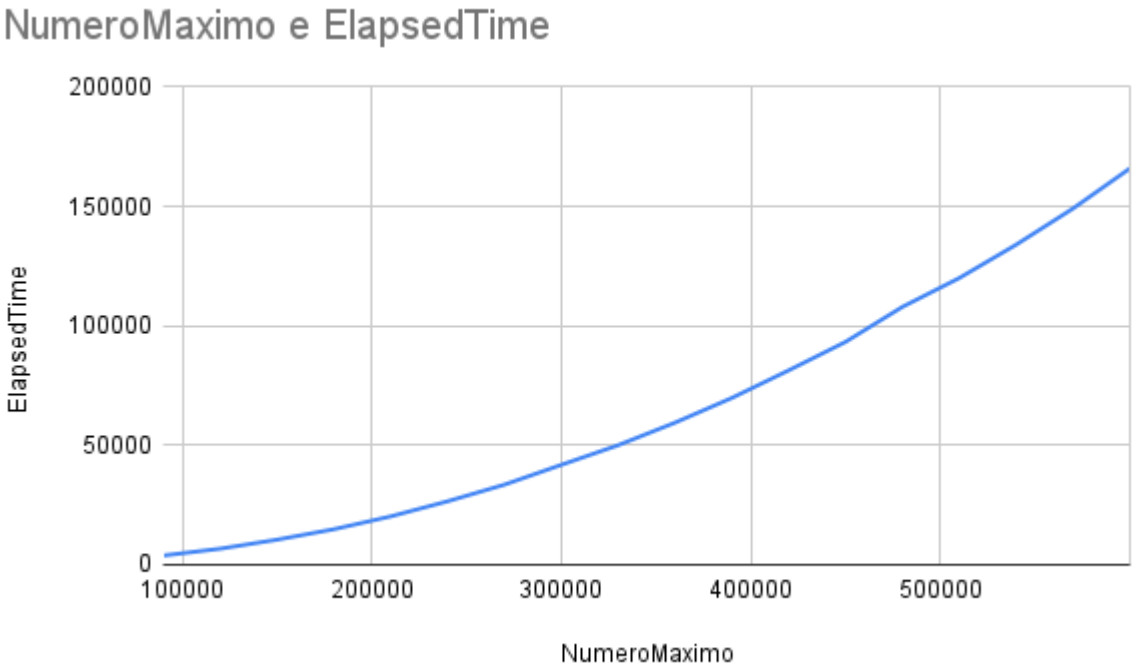

Observando o gráfico ElapsedTime vs MeanUtilization x NumThreads podemos observar que o número de threads mais adequado é 7, pois com 7 threads temos um uso da CPU de 80% então a CPU não está sobrecarregada e tem um tempo de execução de 3768ms. Com mais de 7 threads a CPU já fica sobrecarregada, podemos usar o exemplo com 8 threads que tem utilização da CPU de 99%, e com menos de 7 threads tem um tempo de execução maior, podemos utilizar o exemplo de 6 threads que tem o tempo total de execução de 4096ms e utiliza 67% da CPU.

Sim, em caso de executar o programa em uma máquina diferente o resultado seria diferente, executei o mesmo programa na minha máquina que é Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz  2.30 GHz, com 12 de RAM e HD de 1T,  e encontrei um resultado totalmente diferente, acompanhe o gráfico abaixo.



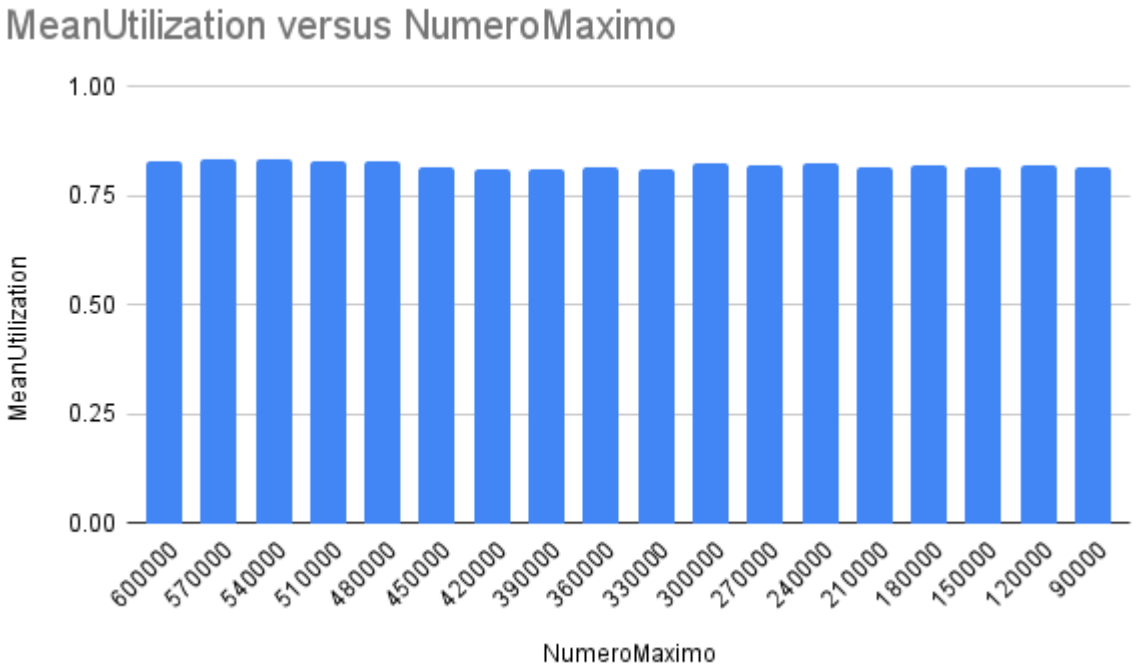

Vendo o gráfico é possível observar que o melhor número de threads utilizando essa máquina é 3, pois usa 87% da CPU ainda não está sobrecarregando e tem uma velocidade total de execução de 16421 ms. Se comparar os valores do melhor número de threads do exemplo anterior que foi 8 threads com uso de 80% da CPU e uma velocidade de 3768ms pode se vê uma diferença olhando para esse exemplo agora, quando se utiliza 8 threads temos o uso de 99% da CPU e uma velocidade total de execução de 14681 ms, assim dá para ver a diferença utilizando máquinas diferentes.

2. Queremos agora analisar a escalabilidade deste programa. Então, vamos analisar como o desempenho é impactado quando aumentamos a carga de trabalho do sistema. Neste problema, a carga aumenta quando aumentamos o número máximo indicado para a busca de números primos (*NumeroMaximo*). Execute o programa para valores diferentes do número máximo para a busca dos primos. Exemplo: 100000, 200000, 300000, 400000 ... (não se restrinja a esses valores). Para esta análise, mantenha fixo o número de threads, usando o número escolhido na letra (c) da questão 1. Analise:
    a. Como o aumento da carga do sistema (o número máximo) afeta o tempo total de execução? Analise os resultados e discuta possíveis causas deste comportamento.




Com base no gráfico acima, podemos observar que, quanto maior o número máximo, maior será o tempo total de execução, pois o número de threads está contínuo, fixo em 7, então quanto maior a carga maior o tempo será para executar.

b. Com base nos resultados de desempenho, você considera este programa escalável? Justifique sua resposta e dê sua opinião sobre possíveis motivos dele ser ou não ser.




Como podemos observar no gráfico, esse programa é sim escalável, pois independente do aumento do número máximo, o uso da CPU se mantém constante, usando por volta dos 80% da CPU.

3. [EXTRA] Faça outras análises com o programa e tente extrair informações interessantes sobre desempenho. Você pode alterar o código para responder esta questão. Você também pode executar o programa em máquinas com hardware diferentes e comparar os resultados, discutindo as diferenças de desempenho.

   Na questão dos números primos letra C do primeiro, coloquei dois gráficos, um do programa executado na máquina da UF e outro gráfico com os dados do programa executado na minha, são máquinas com hardwares diferentes, e podemos observar uma grande diferença no desempenho, na máquina da UF o melhor desempenho foi utilizando 7 threads, que alcançou por volta de 80% de utilização da CPU e uma velocidade de 3768 ms, enquanto na máquina da minha casa, o melhor desempenho foi com 3 threads, alcançando 87% do uso da CPU e com velocidade total de execução de 16421 ms.

   E quando se executa com 7 threads na minha máquina alcança 99% do uso da CPU, deixando ela sobrecarregada, e uma velocidade total de execução de 14639, assim podemos observar uma grande diferença no desempenho de uma máquina para outra.

   Informações das máquinas:

Máquina da UF:
Intel(R) Core(TM) i3-10100T CPU @ 3.00GHz  2.90 GHz
16 GB de Memória
SSD de 240 GB de Disco

Máquina de Casa:
Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz  2.30 GHz
12 de RAM de Memória
HD de 1T de Disco

## Problema 2: Fazendo acessos à memória

**Visão geral**: o objetivo deste problema é avaliar o desempenho do acesso nos vários níveis de memória, quando se tem tamanhos variados de itens a serem buscados. Queremos saber como o desempenho dos acessos a um array armazenado na memória é afetado à medida que o seu tamanho indicado como entrada é alterado.

**Descrição do programa**: o programa Java chamado *BenchmarkAcessoMemoria* recebe como parâmetros de entrada o tamanho ***T*** do array em KiloBytes (KB) que será usado na execução e a quantidade de acessos ***X*** que serão feitos no array. O programa inicialmente aloca o array com o tamanho especificado, preenche o array com valores aleatórios e em seguida realiza X repetições de acessos a posições aleatórias do array. Um exemplo de linha de execução do programa:

```
java -cp bin/:lib/* BenchmarkAcessoMemoria 1024 90000000
```

Neste caso, o programa executará usando um array de 1024 KB (ou 1 MB), que será iniciado e em seguida serão feitos 110 milhões de acessos em posições aleatórias do array.

Saída do programa: Para cada execução do programa, um arquivo será criado contendo os resultados das medições feitas sobre o seu desempenho, localizado no diretório "output-bench-memoria" criado no diretório de execução. O arquivo criado será o seguinte:

- measurements-kb?-rep?-st?.txt: Contém informações sobre a execução e medições com relação à execução. Possui as seguintes colunas:

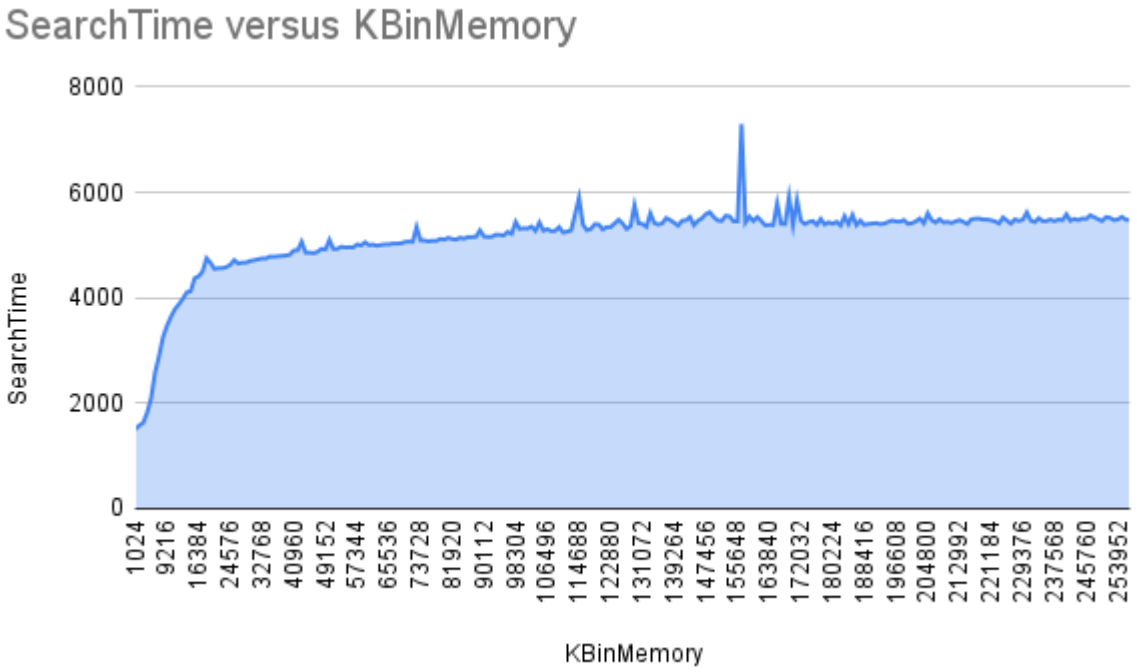
- KBinMemory: tamanho do array usado, especificado em KiloBytes (KB).
- ArraySize: tamanho do array em quantidade de elementos.
- SearchRepetitions: quantidade de repetições de busca realizadas no array (quantidade de acessos)
- StartTime: timestamp que registra o início da execução.
- PopulateTime: tempo para preencher o array.
- SearchTime: tempo para realizar as X repetições de buscas no array.
- TotalTime: tempo total de execução.

Nos nomes dos arquivos há identificações do tipo kb??-rep??-st??, onde o valor seguido de "kb" significa o tamanho do array em KB, "rep" é a quantidade de repetições de busca no array e "st" registra o timestamp inicial da execução, para diferenciar execuções iguais.

Para responder as questões, execute o programa fazer buscas em um array na memória variando o tamanho do array usado, e mantendo a quantidade de repetições de busca fixa em 90000000 (valor default). Com base nas informações dados sobre o programa, responda as questões abaixo:

## Questões

1) Queremos saber qual o impacto do tamanho do array no desempenho dos acessos feitos à memória. Podemos ter um problema parecido quando tratamos de um sistema banco de dados, por exemplo. Analise o desempenho das buscas feitas no array para uma grande variedade de tamanhos de array. Por exemplo, você pode executar tanto para tamanhos de algumas unidades de KB (1, 2, ...), quanto para algumas dezenas (10, 20, ...), centenas (100, 200, ...), da ordem de MB (1024, 2048, ...) até quanto for possível no seu sistema (provavelmente perto de 1GB = 1024 * 1024 KB). Com essas medições, você deve ser capaz de realizar as seguintes análises:
    a. Como o tempo de busca no array varia quando se aumenta o tamanho do array? Analise os resultados e discuta possíveis causas deste comportamento.




Analisando o gráfico podemos perceber, o tempo de busca no array aumenta à medida que o tamanho do array aumenta, mas essa relação não é linear e pode ser afetada por vários fatores, incluindo a arquitetura do computador, a forma como os dados são armazenados na memória e a distribuição dos dados no array.

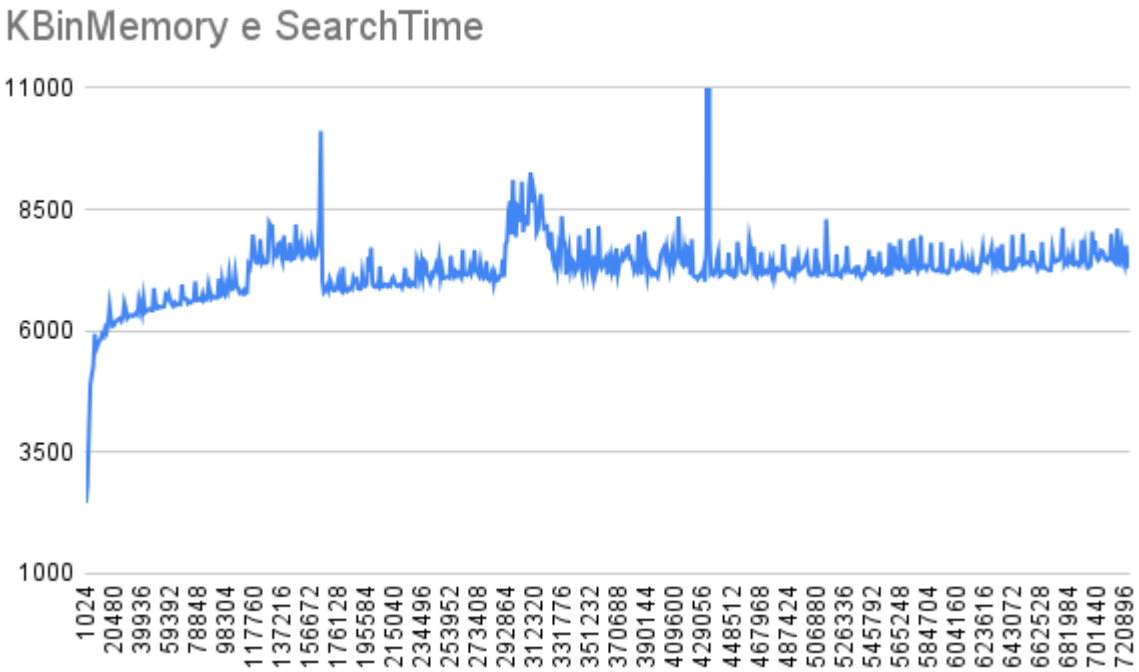
b. O aumento do tempo de busca é bem comportado quando se aumenta o tamanho do array? O sistema é escalável, ou em alguns momentos existem mudanças mais bruscas no tempo de busca ao variar o tamanho do array? Analise esta questão e discuta possíveis causas deste comportamento.

De acordo com o gráfico anterior, podemos observar que o aumento do tempo de busca não é bem comportado quando se aumenta o tamanho do array. Isso ocorre porque o tempo de busca é diretamente proporcional ao tamanho do array. Ou seja, quanto maior o array, mais tempo será necessário para encontrar um elemento específico. Mas o aumento não é exponencial. Após o array passar 18432 KB o tempo de procura, diminui a proporcionalidade que vinha aumentando, mas mesmo assim ainda tem uma crescente.

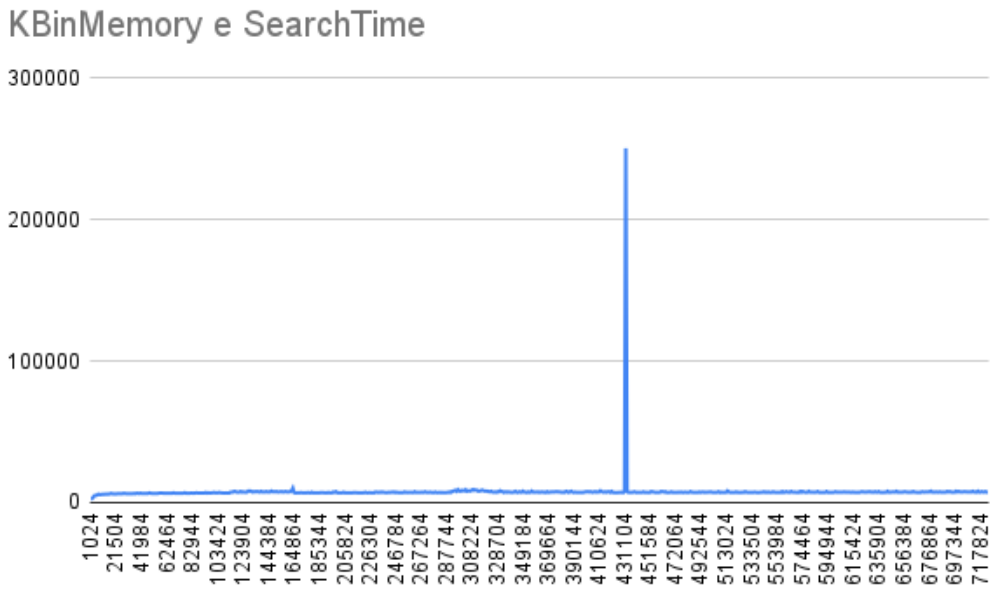
Com base nesses dados, podemos concluir que o sistema pode não ser escalável o suficiente para lidar com arrays muito grandes.

2) [EXTRA] Faça outras análises com o programa e tente extrair informações interessantes. Você pode alterar o código para esta tarefa.

Rodei o mesmo programa com um intervalo maior na minha máquina, varia de 1024 KB até 723968 KB quase 1GB, tentei rodar com mais valores para observar melhor os dados.




Observando esse gráfico podemos observar que o tempo de busca continua aumentando de acordo com o aumento do array, mas que tem algumas variações distintas, às vezes ele diminui, e outras vezes tem um pico muito grande. eu limitei o tempo de busca no gráfico acima para podermos observar melhor, mas quando o tamanho do array alcança 432128 KB tem um pico muito grande, chegando a alcançar 250572 ms. Podemos observar esse pico no gráfico abaixo.




Esse pico pode ter sido causado por diversos fatores, e um deles é limitação de hardware.

OBS: as questões extras somadas valem até 1 ponto adicional na atividade.