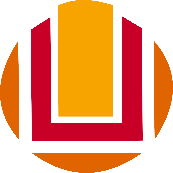
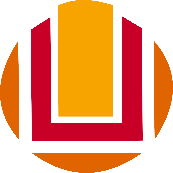
**UNIVERSIDADE FEDERAL DO RIO GRANDE – FURG**

**CENTRO DE CIÊNCIAS COMPUTACIONAIS**

**CALCULADORA DE DUAS OPERAÇÕES**

Rio Grande – RS

2020

**UNIVERSIDADE FEDERAL DO RIO GRANDE – FURG**

**CENTRO DE CIÊNCIAS COMPUTACIONAIS**

**CALCULADORA DE DUAS OPERAÇÕES**



Rio Grande – RS

2020

**INTRODUÇÃO**

O exercício retratado aborda uma calculadora de duas operações, ADIÇÃO (A) e MULTIPLICAÇÃO (M), o algoritmo deve permitir operações com operandos de qualquer tamanho “n”, o qual representa o número de algarismos do operando. Para cada algoritmo realizado efetuamos as seguintes tarefas.

Primeiramente, definimos um algoritmo tradicional para adição, somando algarismo por algarismo e para multiplicação usamos o método russo por deixar o algoritmo mais simples. Em seguida, escrevemos o pseudocódigo de cada um dos algoritmos. Aplicamos uma linguagem de programação de nossa preferência, em que, adotamos a linguagem Python para criar o código. Além disso, selecionamos dois computadores com especificações diferentes para comparar o funcionamento dos algoritmos. Por último, criamos 4 gráficos de desempenho (Tamanho de Entrada X Tempo de Execução) para verificar o comportamento de cada algoritmo em cada computador.

O principal objetivo desse exercício é comparar o desempenho dos algoritmos de ADIÇÃO (A) e MULTIPLICAÇÃO (M) em computadores obrigatoriamente com processadores diferentes e possivelmente com uma diferença consideravelmente nos processadores e memória RAM. Ainda, implemente em cada algoritmo, pelo menos, 10 tamanhos de entradas distintas.

**PSEUDO-CÓDIGO**

**Algoritmo “Menu”**

Início

// Seção de Comandos

Procedimento main (object)

Função \_\_init\_\_ (self)

c <- calculadora.calculadora()

self.run <- verdadeiro

Enquanto self.run então

Invalid <- falso

Escreva(“Escolha uma opção: (...) 1 - Somar dois valores (...) 2 - Multiplicar dois valores (...) 3 - Sair (...) ”)

Leia (menu)

Se não a 3 então

Escreva(“Escolha o modo de operação: (...) 1 - Digitar o valor completo (...) 2 - Completar automaticamente (...)”)

Leia (execution\_type)

Escreva(“Digite o 1° número”)

Leia (number1)

Escreva(“Digite o 2° número”)

Leia (number2)

Se 1 então

filled\_number1 <- c.convert (number1)

filled\_number2 <- c.convert (number2)

Senão

Se 2 então

Escreva(“Digite o número de casas decimais do 1° numero”)

Leia (decimal\_places1)

Escreva(“Digite o número de casas decimais do 2° numero”)

Leia (decimal\_places2)

filled\_number1 <- c.fill(c.convert (number1), decimal\_places1, Verdadeiro)

filled\_number2 <- c.fill(c.convert (number2), decimal\_places2, Verdadeiro)

Fimse

Senão

Invalid <- Verdadeiro

Fimse

Se não a invalid então

Se 1 então

Escreva (c.unconvert(c.sum(filled\_number1, filled\_number2, Verdadeiro)))

Senão

Escreva (c.unconvert(c.sum(filled\_number1, filled\_number2, Verdadeiro)))

Fimse

Fimse

Senão

Escreva “Sair”

self.run <- falso

Fim-Se

Fim-Enquanto

Fim-Função

Fim-Procedimento

Fim-algoritmo

**Algoritmo “Calculadora”**

Início

// Seção de Comandos

Procedimento Calculadora (object)

Função \_\_init\_\_ (self)

Escreva “Inicializando...”

Fim-Função

Função convert (self, number\_string)

number\_int <- []

Para i de i até number\_string faça

number\_int.append(int(i))

Retorne number\_int

Fim-Para

Fim-Função

Função unconvert (self, number)

number\_s <- ' '

Para i de i até number faça

number\_s <- number\_s + str(i)

Retorne number\_s+

Fim-Para

Fim-Função

Função fill (self, number\_fill, decimal\_places, start)

Se start então:

Se decimal\_places > len(number\_fill) então

fill\_places <- decimal\_places - len(number\_fill)

Fim-Se

Senão:

fill\_places <- decimal\_places

Fim-Se

Para i de i até range (0, fill\_places) faça

number\_fill.append(0)

Retorne number\_fill

Fim-Para

Fim-Função

Função sum (self, number1, number2)

number\_sum <- []

rest <- Falso

last <- Falso

Se len(number1) >= len(number2) então

x <- len(number2) - 1

y <- len(number1) - 1

menor <- number2

maior <- number1

Senão

x <- len(number1) - 1

y <- len(number2) - 1

menor <- number1

maior <- number2

Fim-Se

Para i de i até range (y, -1, -1) faça

Se i = 0 então

last <- Verdadeiro

Senão

last <- Falso

Fim-Se

Se x < 0 então

num <- 0

Senão

num <- menor[x]

Fim-Se

Se não a rest então

soma <- maior[i] + num

Senão

soma <- maior[i] + num + 1

Fimse

Se (soma > 9) e não a (last) então

rest <- True

number\_sum.append(soma - 10)

Senão

number\_sum.append(soma)

rest <- Falso

x -= 1

Fim-Para

number\_sum.reverse()

Retorne number\_sum

Fim-Função

Função mult (self, number1, number2)

number\_mult <- []

y <- len(number1) - 1

rest <- 0

Para i de i até range(y, -1, -1) faça

mult <- (number1[i] \* number2) + rest

cont <- 0

rest <- 0

Enquanto mult >= 10 então

mult -= 10

cont += 1

Fim-Enquanto

number\_mult.append(mult)

rest <- cont

Fim-Para

number\_mult.reverse()

Retorne number\_mult

Fim-Função

Função multiply (self, number1, number2)

number\_mult <- []

decimal\_places <- 0

Se len(number1) >= len(number2) então

x <- len(number2)-1

y <- len(number1)-1

menor <- number2

maior <- number1

Senão

x <- len(number1)-1

y <- len(number2)-1

menor <- number1

maior <- number2

Fim-Se

soma <- []

Para i de i até range (x, -1, -1) faça

mult <- self.mult(maior, menor[i])

mult\_fill <- self.fill(mult, decimal\_places, False)

soma <- self.sum(soma, mult\_fill)

decimal\_places += 1

Fim-Para

Retorne soma

Fim-Função

Fim-Procedimento

Fim-Algoritmo

**ESPECIFICAÇÕES DOS COMPUTADORES**

Para testar o programa, utilizamos dois computadores com tecnologias distintas. Além disso utilizamos uma biblioteca específica para medir o tempo de execução do programa. Vale lembrar que, para testar a eficiência de um software, é necessários avaliar outros quesitos como por exemplo a complexidade de cada comando ou ainda o tanto de memória gasta no processo. Todavia, como é um teste mais simples, apenas medir o tempo de execução é suficiente.

**Computador 1:**

Processador: Intel Core i5- 6200 2.30GHz

Base do Processador: x64 bits

Sistema Operacional: Windows 10 Pro x64 bits

Memória Ram: 8 GB DDR4

**Computador 2:**

Processador: AMD E-300 APU with Radeon(™) HD Graphics 1.30GHz

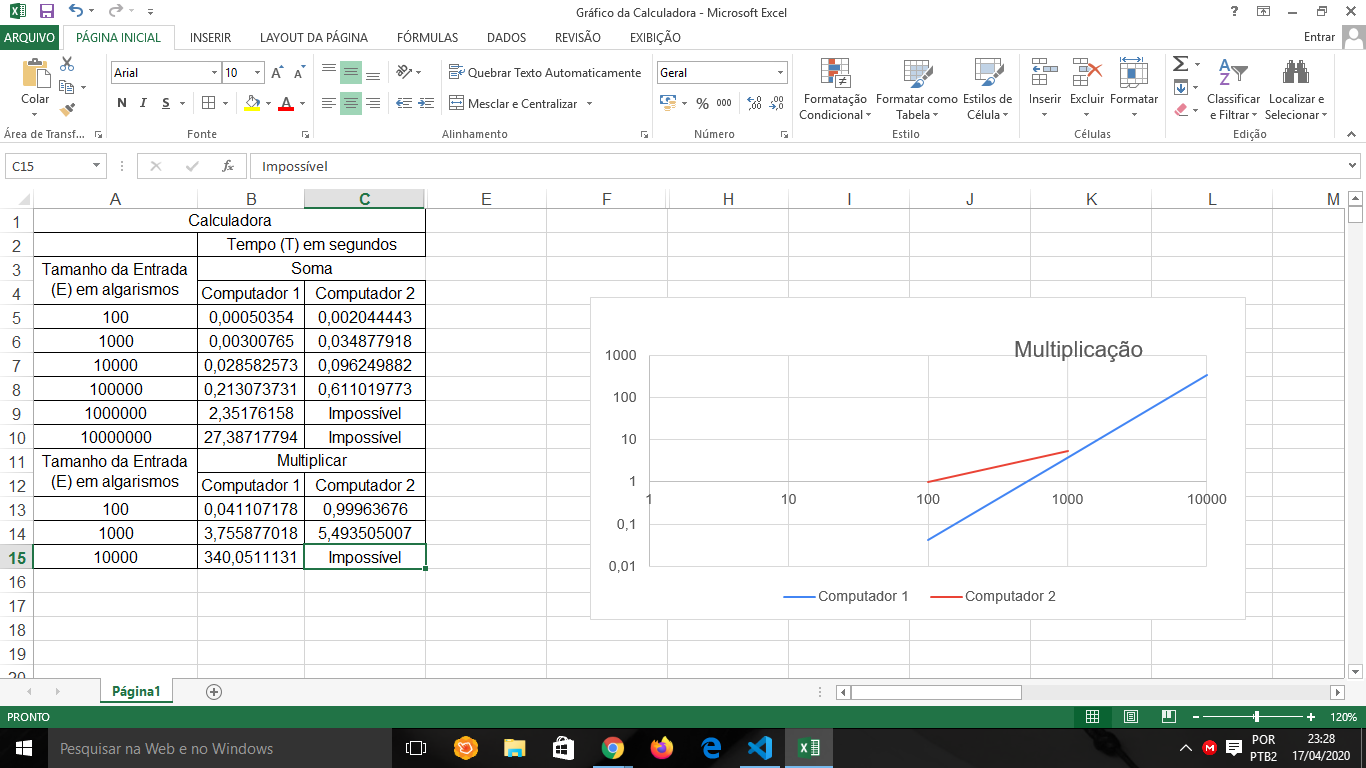
Base do Processador: x64 bits

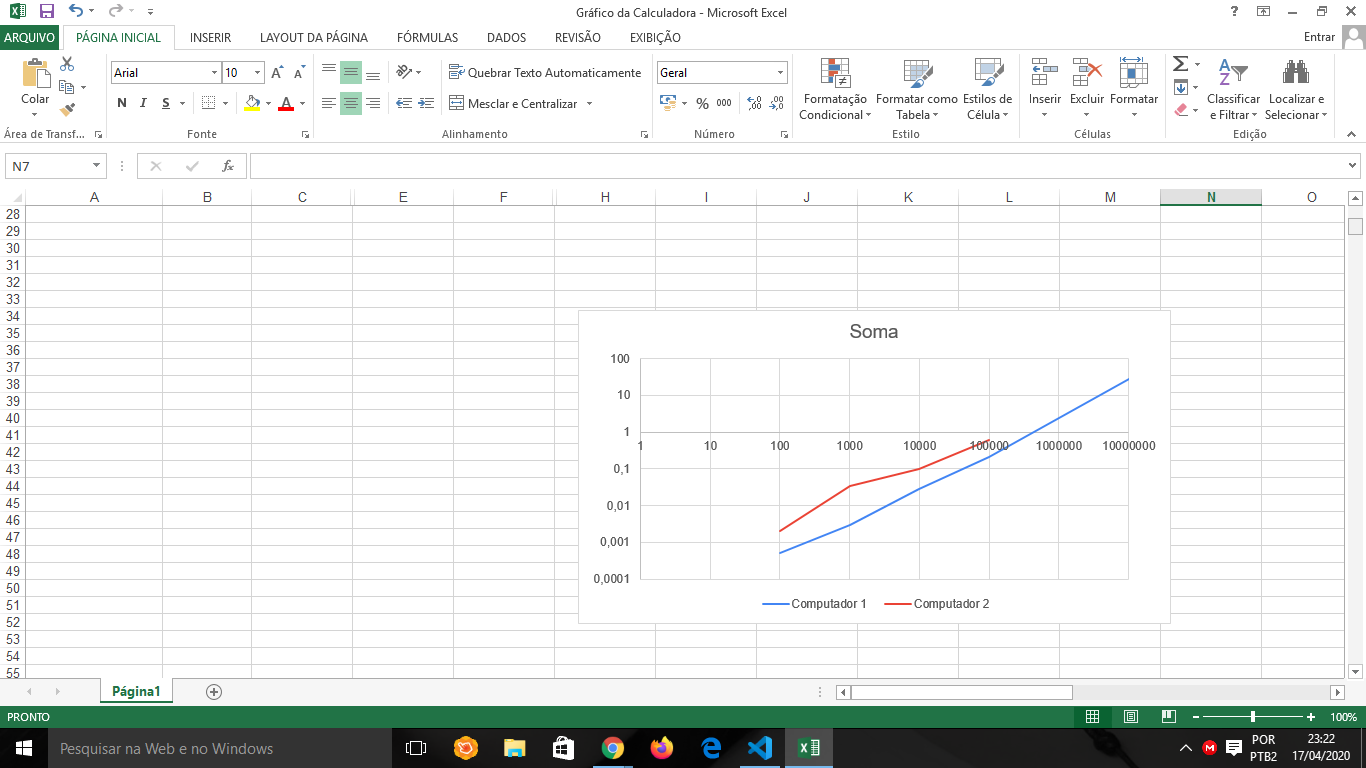
Sistema Operacional: Windows 10 Home x32 bits

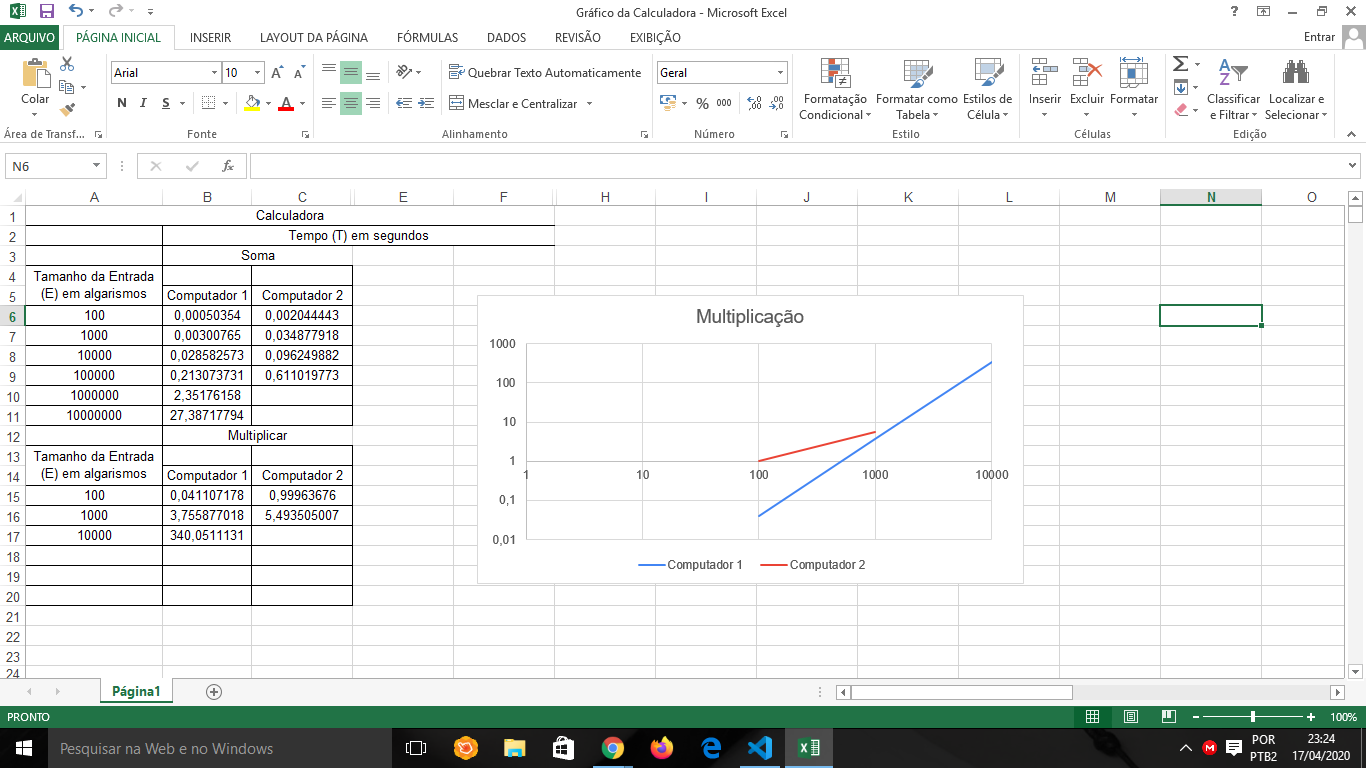
Memória Ram: 2 GB

Em cada computador efetuou-se uma limpeza nos processos para efetuar o teste de desempenho, dessa forma minimiza-se a interferência de aplicações que não deveriam ser executadas naquele momento. Para medida do tempo de execução, utilizamos da biblioteca time. Desta forma, conseguimos calcular a eficiência do algoritmo no quesito Tempo de Execução.

**GRÁFICOS**

De acordo com os testes efetuados nos computadores citados, efetuamos gráficos para cada computador. Estes gráficos tornam mais visíveis a eficiência de cada computador na execução do software. O gráfico é determinado pela variável “T” que determina o tempo em segundos e pela variável “E” que determina o tamanho da entrada. Logo temos um gráfico do tamanho da entrada (E) pelo tempo(T).





**ANÁLISE DOS RESULTADOS**

O conjunto de dados analisados consistem de informações obtidas através de dois computadores com processadores e memórias diferentes.

O computador 1 com processador e memória superiores ao outro, analisou esses dados bem mais rápido de maneira que, já era previsto. No algoritmo da soma conseguiu chegar a casa dos 10000000 milhões de algarismos, logo na parte de multiplicar chegou a casa dos 10000 mil algarismos.

Analisando o computador 2 com processador e memória inferiores ao primeiro, foi verificado que o tempo foi mais longo, assim chegando a casa menores de algarismos. Na soma chegou a 100000 mil e multiplicação a mil.

**CONCLUSÕES FINAIS**

Podemos concluir que tanto para entradas pequenas como para entradas grandes o algoritmo da soma executa o programa mais rapidamente, assim como era previsto. A utilização de um processador melhor diminuiu o tempo de execução do algoritmo, desse modo, mostrando que um computador melhor faz muita diferença, mas por outro lado não podemos esquecer que um algoritmo mais eficiente ajuda bastante também.

**REFERÊNCIAS**

STACKOVERFLOW. Medir o tempo de execução de uma função. Site: <<https://pt.stackoverflow.com/questions/97364/medir-o-tempo-de-execu%C3%A7%C3%A3o-de-uma-fun%C3%A7%C3%A3o>>