Relatório do exercício 2 de Estruturas de Dados
Giovana Piorino Vieira de Carvalho – matrícula: 2022035989

Análise da função de cálculo do fatorial

Os códigos recursivos e iterativos foram testados, em C, numa faixa de entrada entre 1 – 50 para todas as análises. Assim, o tempo de relógio foi calculado com auxílio da biblioteca “time.h”. Usando o número 40 como entrada, que gera um resultado consideravelmente grande, os resultados de tempo de relógio foram obtidos:

```
40
Fatorial recursivo: 18376134811363311616
Tempo fatorial recursivo: 0.35276
Fatorial iterativo: 18376134811363311616
Tempo fatorial iterativo: 0.2867
```

Realizando um comparativo com os relatórios do gprof, verificam-se dados curiosos:

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        1     0.00     0.00  fatorial_iterativo
  0.00      0.00     0.00        1     0.00     0.00  fatorial_recursivo
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       1/1           main [10]
[1]      0.0    0.00    0.00       1          fatorial_iterativo [1]
-----------------------------------------------
                                  40             fatorial_recursivo [2]
                0.00    0.00       1/1           main [10]
[2]      0.0    0.00    0.00       1+40       fatorial_recursivo [2]
                                  40             fatorial_recursivo [2]
-----------------------------------------------
```

Assim, infere-se que as chamadas ao sistema foram bem rápidas, e com pouca diferença, logo o gprof nem considerou o tempo necessário para executar as funções, de tão baixo. Focando no algoritmo recursivo, ao medir o tempo de execução de cada chamada, com uma entrada 5, obtêm-se:

```
Tempo de execucao para fatorial(2) = 0.000001 segundos
Tempo de execucao para fatorial(3) = 0.000074 segundos
Tempo de execucao para fatorial(4) = 0.000084 segundos
Tempo de execucao para fatorial(5) = 0.000093 segundos
Fatorial(5) = 120
```

Um teste realizado, afim de consumir os recursos computacionais da função, foi de inserir, dentro da função recursiva de cálculo do fatorial, uma pequena função que calcula o seno do número de entrada milhares de vezes. Logo depois o programa foi executado usando a entrada 40 de parâmetro. O resultado obtido foi:

```
40
Fatorial recursivo: 18376134811363311616
Tempo fatorial recursivo: 8.983110651
Fatorial iterativo: 18376134811363311616
Tempo fatorial iterativo: 0.12420
```

Percebe-se a diferença nítida no tempo de relógio da função recursiva nesse experimento. A diferença também é explícita no relatório do gprof:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00      8.98     8.98        1     8.98     8.98  fatorial_recursivo
  0.00      8.98     0.00        1     0.00     0.00  fatorial_iterativo
```

O relatório mostra um tempo semelhante ao mostrado pelo programa, além de que agora, a função recursiva consome praticamente todo o tempo dele.

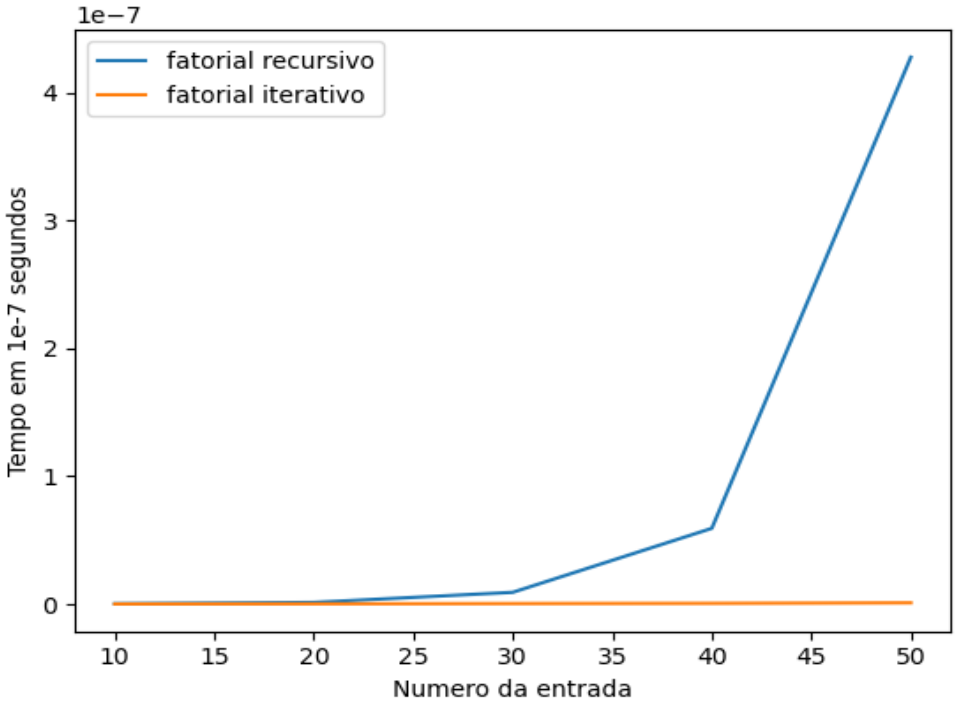
O teste também foi realizado inserindo a função de cálculo de seno em cada chamada, porém com uma entrada menor por questões de visualização, e também apresentando um número consideravelmente maior de tempo de relógio para cada chamada:

```
Tempo de execucao para fatorial(2) = 0.235600 segundos
Tempo de execucao para fatorial(3) = 0.465554 segundos
Tempo de execucao para fatorial(4) = 0.695373 segundos
Tempo de execucao para fatorial(5) = 0.936398 segundos
Fatorial(5) = 120
```

Além desses experimentos, também foram medidos os tempos de utilização de recursos em ambas funções, com auxílio do comando “time make run” ao compilar o programa, e a biblioteca “sys/resource.h”:

```
40
Fatorial recursivo: 18376134811363311616
Tempo fatorial recursivo: 0.32841
Tempo recursivo: 0.000046 segundos (usuário: 0.000000, sistema: 0.000000)

Fatorial iterativo: 18376134811363311616
Tempo fatorial iterativo: 0.3043
Tempo iterativo: 0.000004 segundos (usuário: 0.000000, sistema: 0.000000)


real    0m1,317s
user    0m0,006s
sys     0m0,000s
```

Dessa forma, verifica-se que o tempo de utilização de recursos é bem pequeno, e a biblioteca utilizada no código teste provavelmente não considerou uma entrada tão baixa, considerando um uso “zero”.
Resumidamente, tanto a função iterativa quanto a recursiva oferecem pouco tempo de execução, sendo consideravelmente eficientes em seu propósito, justificado pelo fato de que ambas funções apresentam complexidade computacional de tempo O(n). Ainda assim, caso a função recursiva receba um alto valor de entrada, ela pode ser ligeiramente mais lenta que a função iterativa, uma vez que precisa realizar diversas chamadas recursivas até realizar todas as chamadas da pilha de execução. Um exemplo disso foi a medição do tempo de relógio (usando a função clock()) para entradas de 1 a 50 em ambas funções, e a partir desses dados, a realização de um gráfico comparativo:



Assim, verifica-se que ambas as funções apresentam um tempo de execução muito baixo, mas conforme a entrada fica maior, a função recursiva toma um pouco mais de tempo para ser executada até o final.

Análise da função da sequência de Fibonacci:

Comparativamente, a função Fibonacci gera um tempo de relógio e de utilização de recursos consideravelmente maior em relação à função fatorial:

```
40
Fibonacci recursivo: 102334155
Tempo Fibonacci recursivo: 3.391474548
Tempo recursivo: 3.391411 segundos (usuário: 3.000000, sistema: 0.000000)

Fibonacci iterativo: 102334155
Tempo fibonacci iterativo: 0.2054
Tempo iterativo: 0.000004 segundos (usuário: 3.000000, sistema: 0.000000)


real    0m5,622s
user    0m3,396s
sys     0m0,000s
```

Essa conclusão também pode ser observada pelo relatório do gprof, que apresentou um tempo de execução consideravelmente menor que o tempo de relógio, e atribuiu uma pequena parcela de tempo de execução ao main:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 91.11      0.41     0.41        1   410.00   410.00  fibonacci_recursivo
  8.89      0.45     0.04                             main
  0.00      0.45     0.00        1     0.00     0.00  fibonacci_iterativo
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 2.22% of 0.45 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.04    0.41                 main [1]
                0.41    0.00       1/1           fibonacci_recursivo [2]
                0.00    0.00       1/1           fibonacci_iterativo [3]
-----------------------------------------------
                            204668308             fibonacci_recursivo [2]
                0.41    0.00       1/1           main [1]
[2]     91.1    0.41    0.00       1+204668308 fibonacci_recursivo [2]
                            204668308             fibonacci_recursivo [2]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[3]      0.0    0.00    0.00       1         fibonacci_iterativo [3]
-----------------------------------------------
```

Também foi realizado o teste de verificar o tempo de relógio de cada chamada na função recursiva, novamente com a entrada 5, sendo uma entrada pequena, os valores eram consideravelmente baixos:

```
Tempo de execucao para fibonacci(2) = 0.000001 segundos
Tempo de execucao para fibonacci(3) = 0.000067 segundos
Tempo de execucao para fibonacci(2) = 0.000000 segundos
Tempo de execucao para fibonacci(4) = 0.000077 segundos
Tempo de execucao para fibonacci(2) = 0.000001 segundos
Tempo de execucao para fibonacci(3) = 0.000005 segundos
Tempo de execucao para fibonacci(5) = 0.000092 segundos
Fibonacci(5) = 5
```

O teste foi novamente realizado inserindo a mesma função de consumir recursos computacionais, que também gerou aumento considerável do tempo:

```
Tempo de execucao para fibonacci(2) = 0.231704 segundos
Tempo de execucao para fibonacci(3) = 0.458727 segundos
Tempo de execucao para fibonacci(2) = 0.000001 segundos
Tempo de execucao para fibonacci(4) = 0.684980 segundos
Tempo de execucao para fibonacci(2) = 0.226645 segundos
Tempo de execucao para fibonacci(3) = 0.226660 segundos
Tempo de execucao para fibonacci(5) = 1.137606 segundos
Fibonacci(5) = 5
```

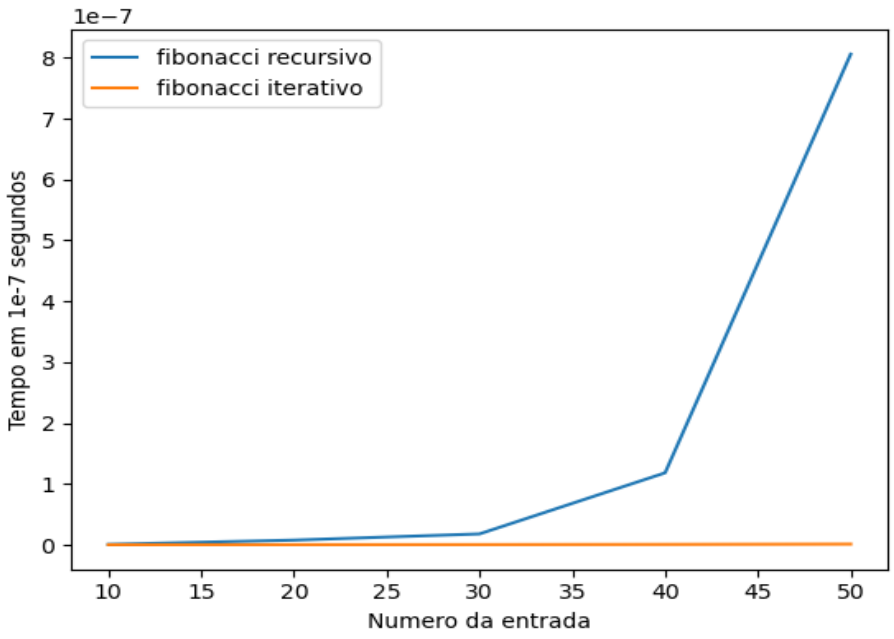
Realizando esse experimento com uma entrada maior (15), e considerando o tempo de relógio para a realização de todo o programa, temos:

```
15
Fibonacci recursivo: 610
Tempo Fibonacci recursivo: 87.241514220
Tempo recursivo: 87.206100 segundos (usuário: 87.000000, sistema: 0.000000)

Fibonacci iterativo: 610
Tempo fibonacci iterativo: 0.2054
Tempo iterativo: 0.000003 segundos (usuário: 87.000000, sistema: 0.000000)


real    1m29,384s
user    1m27,210s
sys     0m0,012s
```

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00     87.22    87.22        1    87.22    87.22  fibonacci_recursivo
  0.00     87.22     0.00        1     0.00     0.00  fibonacci_iterativo
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.01% of 87.22 seconds

index % time    self  children    called     name
                                     1218             fibonacci_recursivo [1]
                0.00   87.22       1/1               main [2]
[1]    100.0    0.00   87.22       1+1218        fibonacci_recursivo [1]
                                     1218             fibonacci_recursivo [1]
-----------------------------------------------
                                                      <spontaneous>
[2]    100.0    0.00   87.22                     main [2]
                0.00   87.22       1/1               fibonacci_recursivo [1]
                0.00    0.00       1/1               fibonacci_iterativo [3]
-----------------------------------------------
                0.00    0.00       1/1               main [2]
[3]      0.0    0.00    0.00       1             fibonacci_iterativo [3]
-----------------------------------------------
```

O “tempo iterativo”(tempo total de utilização de recursos) é bem semelhante entre o calculado pelo programa e pelo gprof: observa-se que a chamada recursiva ocupa praticamente todo o espaço de tempo, enquanto a chamada iterativa apresenta tempo tão pequeno que é “zerada” no relatório gprof, assim como os dados de sistema, consideravelmente pequenos em relação ao tempo do usuário.
Sendo assim, é possível concluir que a função recursiva de Fibonacci é consideravelmente mais lenta e ineficiente que sua versão iterativa. Isso pode ser comprovado ao comparar suas complexidades computacionais de tempo: a função iterativa apresenta complexidade O(n), já a recursiva, O(2^n), uma vez que ela deve realizar duas chamadas recursivas para cada chamada recursiva até atingir o caso base. Essa diferença é explícita em um gráfico de comparação, realizado nos mesmos moldes do gráfico comparativo do fatorial, em que a discrepância das retas é bem mais evidente:



Comparação entre os algoritmos

Por fim, foi realizado um programa envolvendo as 4 funções analisadas, para fazer uma comparação global de tempo de execução dos algoritmos, usando novamente a entrada 40 como base:

```
40
Fatorial recursivo: 18376134811363311616
Tempo fatorial recursivo: 0.27947
Tempo recursivo: 0.000041 segundos (usuário: 0.000000, sistema: 0.000000)

Fatorial iterativo: 18376134811363311616
Tempo fatorial iterativo: 0.3063
Tempo iterativo: 0.000005 segundos (usuário: 0.000000, sistema: 0.000000)

Fibonacci recursivo: 102334155
Tempo Fibonacci recursivo: 3.433522135
Tempo recursivo: 3.433245 segundos (usuário: 3.000000, sistema: 0.000000)

Fibonacci iterativo: 102334155
Tempo fibonacci iterativo: 0.3287
Tempo iterativo: 0.000005 segundos (usuário: 3.000000, sistema: 0.000000)


real    0m4,508s
user    0m3,434s
sys     0m0,004s
```

Relatório gprof, apresentando novamente um tempo de execução menor em uma análise global, abrangendo as quatro funções:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 94.00      0.47     0.47        1   470.00   470.00  fibonacci_recursivo
  6.00      0.50     0.03                             main
  0.00      0.50     0.00        1     0.00     0.00  fatorial_iterativo
  0.00      0.50     0.00        1     0.00     0.00  fatorial_recursivo
  0.00      0.50     0.00        1     0.00     0.00  fibonacci_iterativo
```

Assim, é possível concluir que a função Fibonacci recursiva é um ponto fora da curva nas análises, uma vez que ocupa 94% do tempo de execução no gprof, e muito mais tempo de relógio que as outras funções, levando a considerá-la com alto custo computacional. Além disso, outra parcela do tempo é consumida pelo main, tendo algumas chamadas de saída para realizar. As outras funções não têm presença significativa no tempo de relógio e do gprof, possuindo um custo computacional razoavelmente menor.

Por fim, um gráfico, gerado a partir das coletas dos tempos de relógio apresentados pelas quatro funções do programa, demonstra essa discrepância: enquanto ambas funções fatoriais e a função Fibonacci iterativa apresentam certa linearidade e baixo tempo de relógio, tendo retas praticamente sobrepostas no gráfico, a função de Fibonacci recursiva consome muito mais tempo exponencialmente para ser executada: