Análise Empírica de Algoritmos de Busca

Disciplina: Estrutura de Dados Básicas I, Universidade Federal do Rio Grande do Norte (UFRN)

Alunos envolvidos

- Felipe Ramos
- Oziel Alves

Sumário

Objetivos

Analisar e avaliar o comportamento **assintótico** dos algoritmos em relação ao seu tempo de execução e número de iterações. Os cenários irão simular a execução dos algoritmos para diversos tamanhos de amostras com tamanhos de elementos *n* crescentes, até atingir o limite da máquina de testes.

Instruções de uso

Dependências

Python3, python3-pip, python3-tk

É necessário para executar o script de plotagem dos dados.

```
# Como instalar no Ubuntu 16.04 LTS por exemplo:
 sudo apt-get install python3 python3-pip python3-tk
```

Matplotlib

É necessário para exibir os gráficos gerados pelo script de plotagem.

```
# Como instalar no Ubuntu 16.04 LTS por exemplo:
pip3 install matplotlib
```

G++ Compiler

É necessário para a compilação do programa, visto que ele é feito em c++.

```
# Como instalar no Ubuntu 16.04 LTS por exemplo:
sudo apt-get install g++
```

Compilação

Para compilar, basta usar o seguintes comandos na raiz do repositório:

```
# Compilar somente
make

# Para compilar e executar o algoritmo com parâmetros pré-definidos
# ./bin/analise 10 200 6 25 1111111; python3 src/gen_plot.py
make run

# Para limpar dados residuais (objetos, executáveis e dados gerados)
make clean
```

Execução

Feito isso, basta rodar o programa com os parametros desejados:

```
./bin/analise [TIC] [NT] [I] [NI] [AS]
```

- [TIC] - ***Tamanho inicial do Conjunto***

Como o programa é projetado para testar e analisar algoritmos de busca, é necessário definir o número de elementos que o conjunto irá possuir para então, testar nele o algoritmo de busca.

- [NT] - ***Número de testes***

  Irá estipular quantas vezes o algoritmo de busca será testado para cada número de incrementos [NI], para depois gerar uma média de tempo/iterações que ele consome.

- [I] - ***Incremento***

  Irá definir o incremento obedecendo a seguinte função: *f*(i) = 10^i. Portando, é necessário tomar cuidado com valores acima de 6 (pois já é suficiente para travar um computador de médio porte dependendo da quantidade de incrementos).

- [NI] - ***Numero de Incrementos*** Esse parametro irá definir o número de incrementos que irão acontecer obedecendo os parâmetros estipulados anteriormente.

- [AS] - **Algoritmos Selecionados**

  Esse parâmetro receberá uma string binária, que baseado nela, irá definir quais algoritmos serão selecionados para compor a bateria de testes.

  Exemplos:

  `0101011` Irá rodar apenas os algoritmos 2, 4, 6 e 7.

  `1000000` ou `1` irá rodar apenas o 1• algoritmo.

  `0001000` ou `0001` irá rodar apenas o 4• algoritmo.

  Então, se rodarmos o programa com as seguintes definições:

```
./bin/analise 10000 100 5 25 1111111
```

Teremos um programa que irá analisar 100 vezes um conjunto constituido com:

```
10000 elementos
10000 + (1 * 10^5) elementos
10000 + (2 * 10^5) elementos
10000 + (3 * 10^5) elementos
...
10000 + (23 * 10^5) elementos
10000 + (24 * 10^5) elementos
10000 + (25 * 10^5) elementos
```

E armazenará a média de tempo e iterações das 100 vezes, gerando gráficos para interpretação posterior.

Obs: Vale lembrar que o programa pode ficar **bem** pesado caso o incremento estipulado seja muito forte (acima de 6, por exemplo) dada a função crescimento.

Plotagem de gráficos

O programa realiza a plotagem de gráficos utilizando um script em python 3 se aproveitando da lib matplotlib. Para utilizar o script, basta na raiz do repositório executar:

```
python3 src/plot.py [A] [I]
```

Caso queira fazer a plotagem individual de todos os gráficos, basta utilizar:

```
python3 src/gen_plot.py
```

- [A] - ***Algoritmos***

  É um parâmetro composto por uma string binária, que segue a regra:

  `0111011` irá um gráfico comparando os algoritmos 2, 3, 4, 6 e 7.

  `1110000` ou `111` irá gerar um gráfico comparando os algoritmos 1, 2, 3 apenas.

  `1000000` ou `1` irá gerar um gráfico exclusivo do primeiro algoritmo.

  `0001000` ou `0001` irá gerar um gráfico exclusivo do 4º algoritmo.

  A ordem dos algoritmos é:

  - 1º: ***Busca Sequencial Iterativa***
  - *2º:* ***Busca Binária Iterativa***
  - *3º:* ***Busca Binária Recurssiva***
  - 4º: ***Busca Ternária Iterativa***
  - 5º: ***Busca Ternária Recurssiva***
  - 6º: ***Jump Search***
  - 7º: ***Busca Fibonacci***

- [I] - ***Informações***

  Este parâmetro segue a mesma regra do parametro **[A]** e define quais informações irão compor os eixos X e Y, porem é composto por uma string binária que só pode conter 2 repetições do `1`, visto que a plotagem do gráfico é em $2D$.

  Exemplos:

  `1010` Irá comparar o número de elementos com o tempo médio gasto nos testes.

  `1001` Irá comparar o número de elementos com o número de iterações médias gastas nos testes.

  A ordem das informações é:

  1º: ***Número de elementos***

  2º: ***Número de testes***

  3º: ***Tempo médio gasto***

  4º: ***Iterações gastas***

Organização das amostras

É estipulado como parâmetro de execução o tamanho inicial do conjunto **[TIC]**, o número de incrementos **[NI]** que irão acontecer a cada caso teste e o tamanho do incremento utilizado **[TIU]**.

Após isso, o programa irá gerar um conjunto de $TIC + (NI * TIU)$ elementos e preenche-lo com elementos sequenciais crescentes, visto que a grande maioria dos algoritmos testados tem como requisição básica um conjunto **ordenado** de elementos e **sem repetições**.

Informações sobre as buscas

Todos os algoritmos serão testados contra seu pior caso, ou seja, quando k não se encontra no arranjo A.

Apresentação dos Algoritmos

Busca Sequencial

Também conhecida como busca linear, é um algoritmo de busca que procura por um elemento dado um conjunto, iterativamente, e checa se aquele é o elemento buscado.

Iterativa

Dado um conjunto L de n elementos, com um alvo T, a seguinte sub-rotina é implementada.

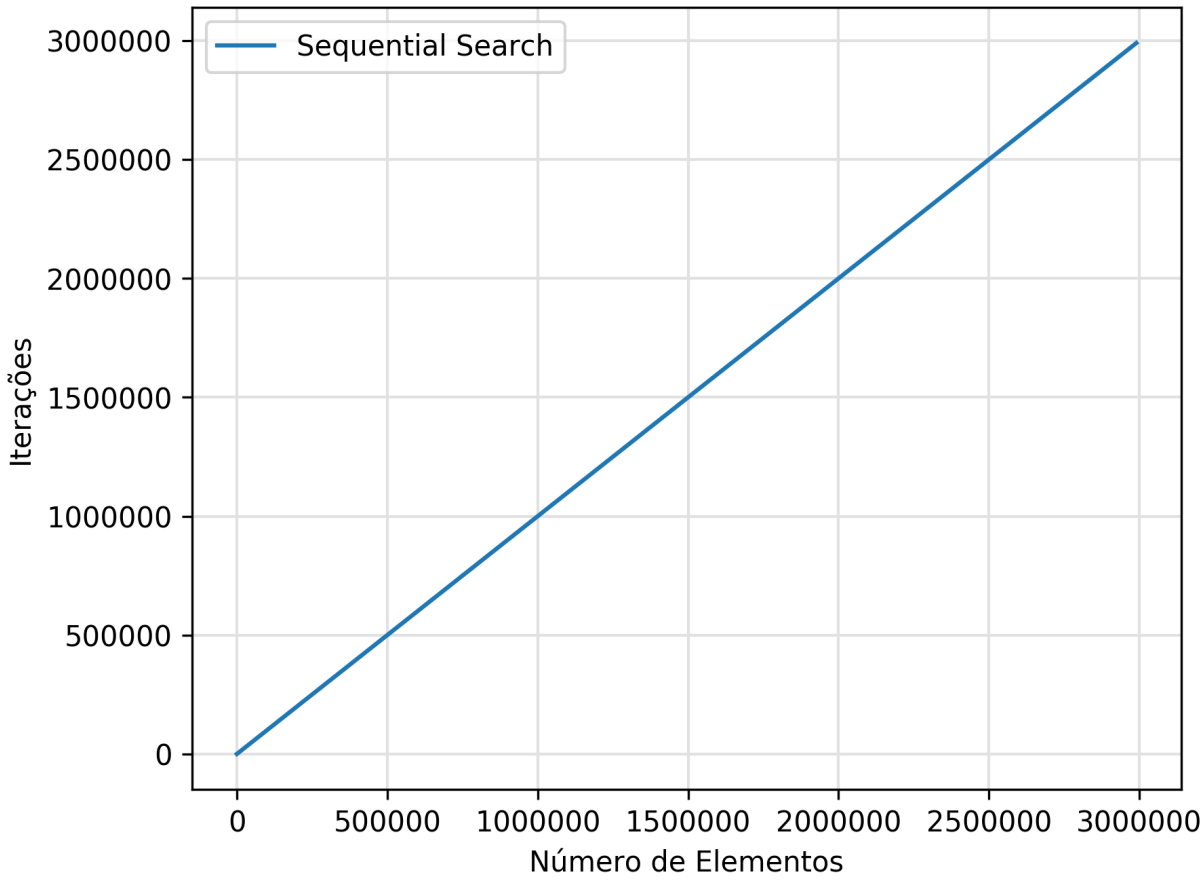
1. É setado $i = 0$.
2. Se $L_i = T$, a busca termina e retorna-se i (que é o local do elemento no vetor).
3. Caso seja falso, i é incrementado em uma unidade.
4. Se $i < n$, iremos para o passo 2, se não, a busca termina sem sucesso.

Para uma lista com n elementos, o melhor cenário possivel é quando o valor buscado está na primeira posição do conjunto, enquanto que seu pior caso é quando T não pertence ao conjunto L.

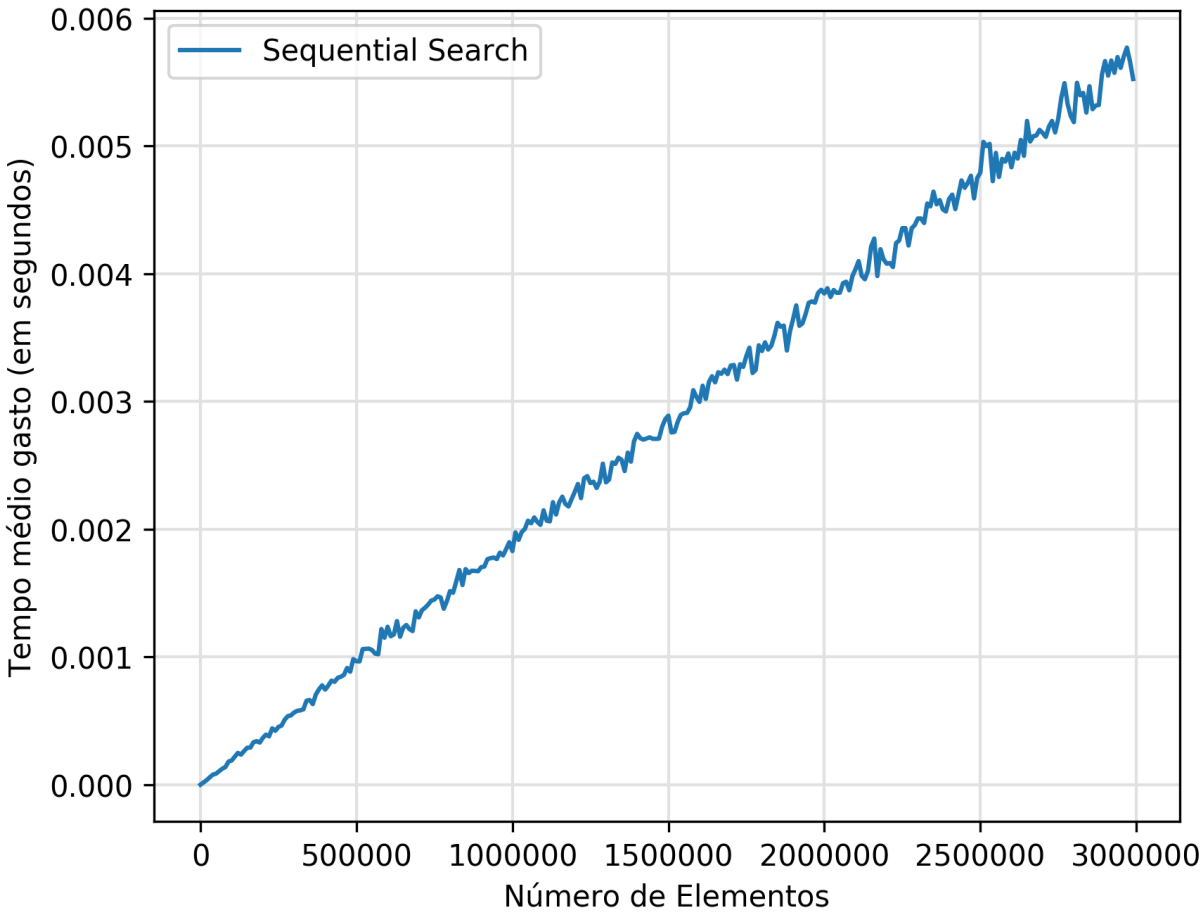
Podemos esperar que $\mathcal{O}(n)$ descreva o comportamento do algoritmo em relação ao número de elementos do conjunto:

Gráficos exclusivos

Tamanho x Iterações



Tamanho x Tempo médio



Busca Binária

Também conhecida como busca logaritmica, é um algoritmo de busca baseado na técnica *divide and conquer* e dependente de uma ordenação, que melhora notávelmente sua performance, visto que a cada iteração, é descartado metade do conjunto.

Iterativa

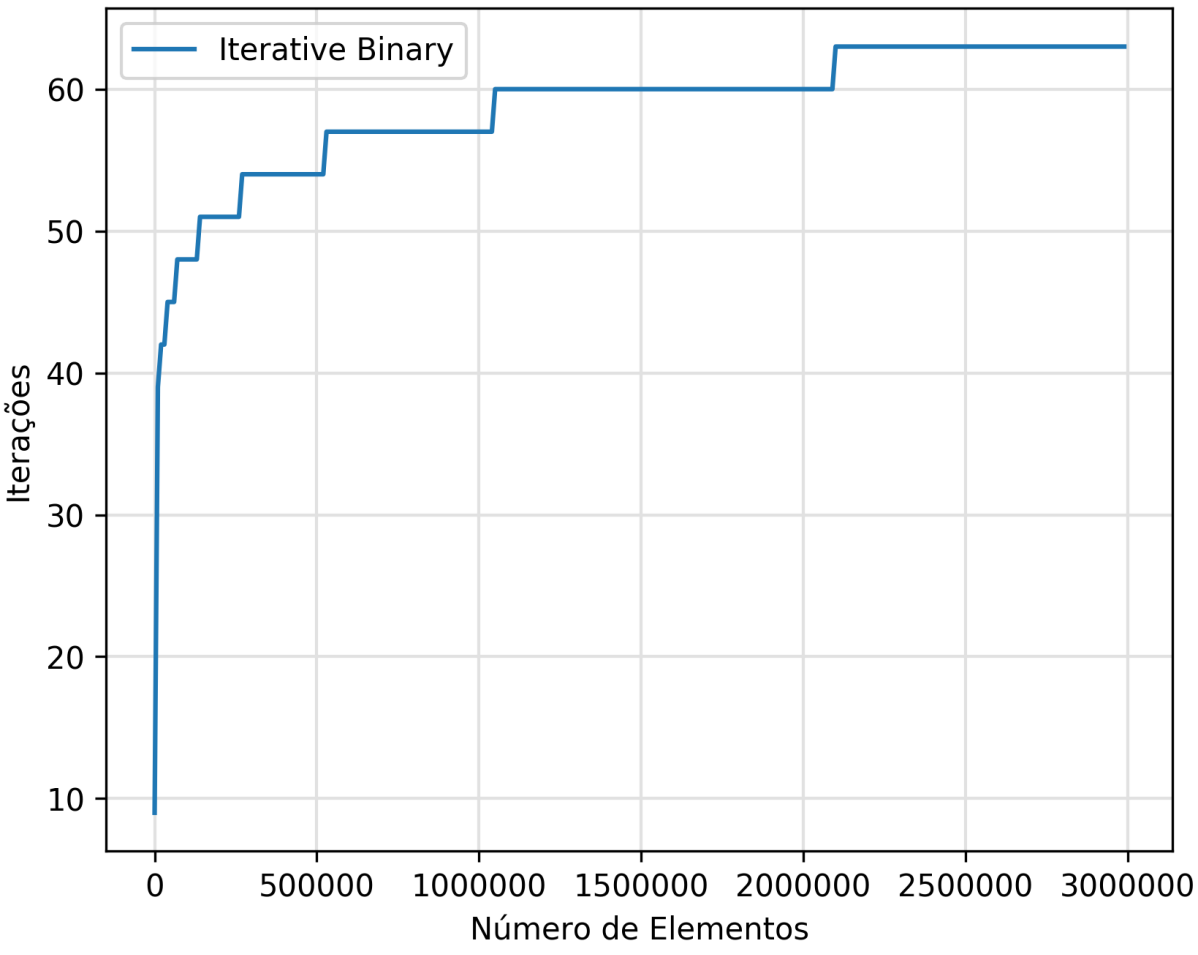
Dado um conjunto A de n elementos ordenados e um alvo T, é seguida a sub-rotina para encontrar o elemento:

1. É setado $L = 0$ e $R = n - 1$.
2. Se $L < R$, a busca termina sem sucesso.
3. É setado a posição do meio (m) do elemento para $\left\lfloor \frac{L+R}{2} \right\rfloor$.
4. Se $A_m < T$, será setado $L = m + 1$ e retorna ao passo 2.
5. Se $A_m > T$, será setado $R = m - 1$ e retorna ao passo 2.
6. Se $A_m = T$, a busca é finalizada retornando m.

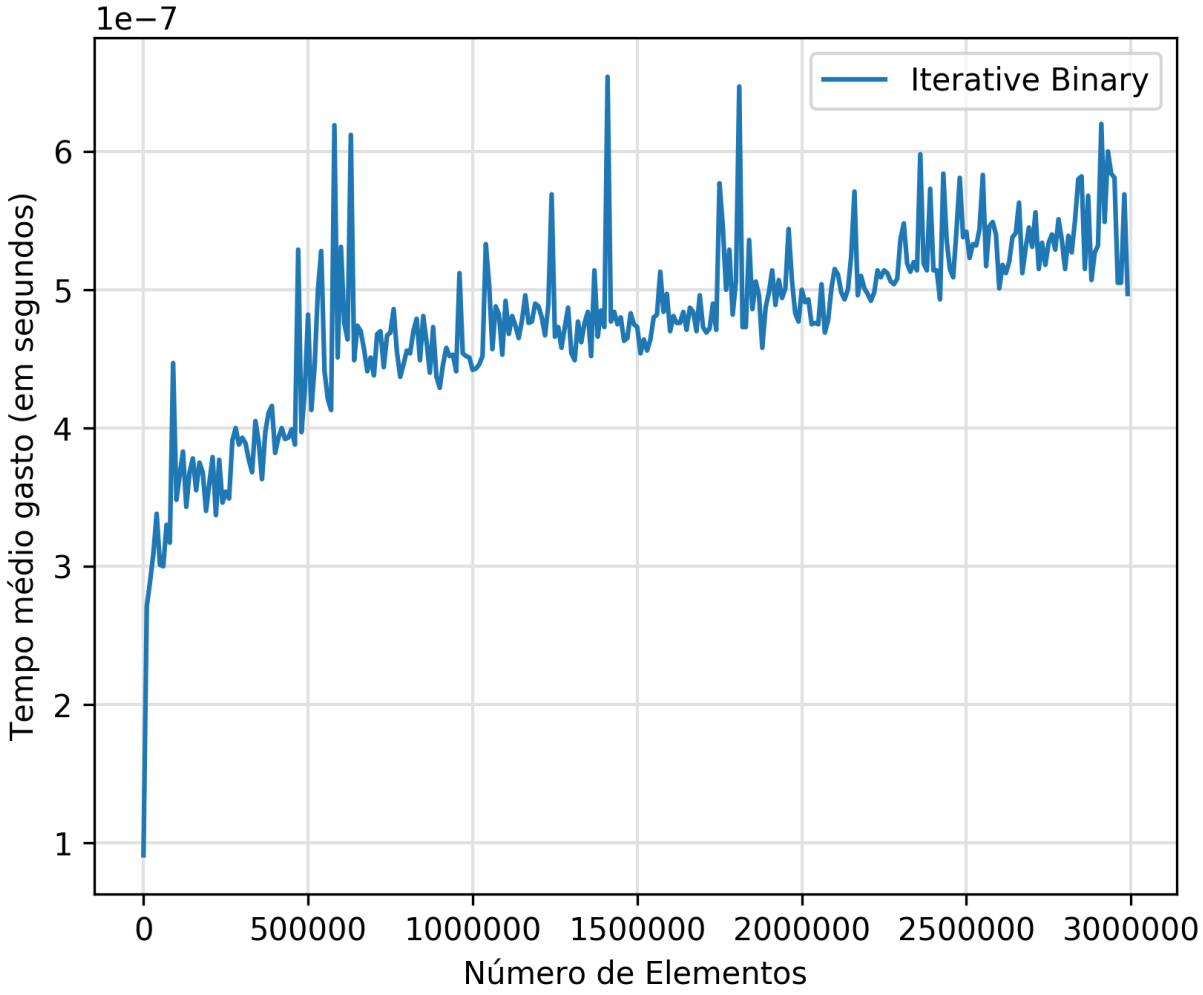
Pelo fato da busca binária utilizar-se do método *divide and conquer*, acaba se tornando uma função logaritma que, em seu melhor caso, o elemento procurado estará exatamente em $\frac{n}{2}$, já em seu pior caso, irá ter $\mathcal{O} \log_2 n$ comparações.

Gráficos exclusivos

Tamanho x Iterações



Tamanho x Tempo médio

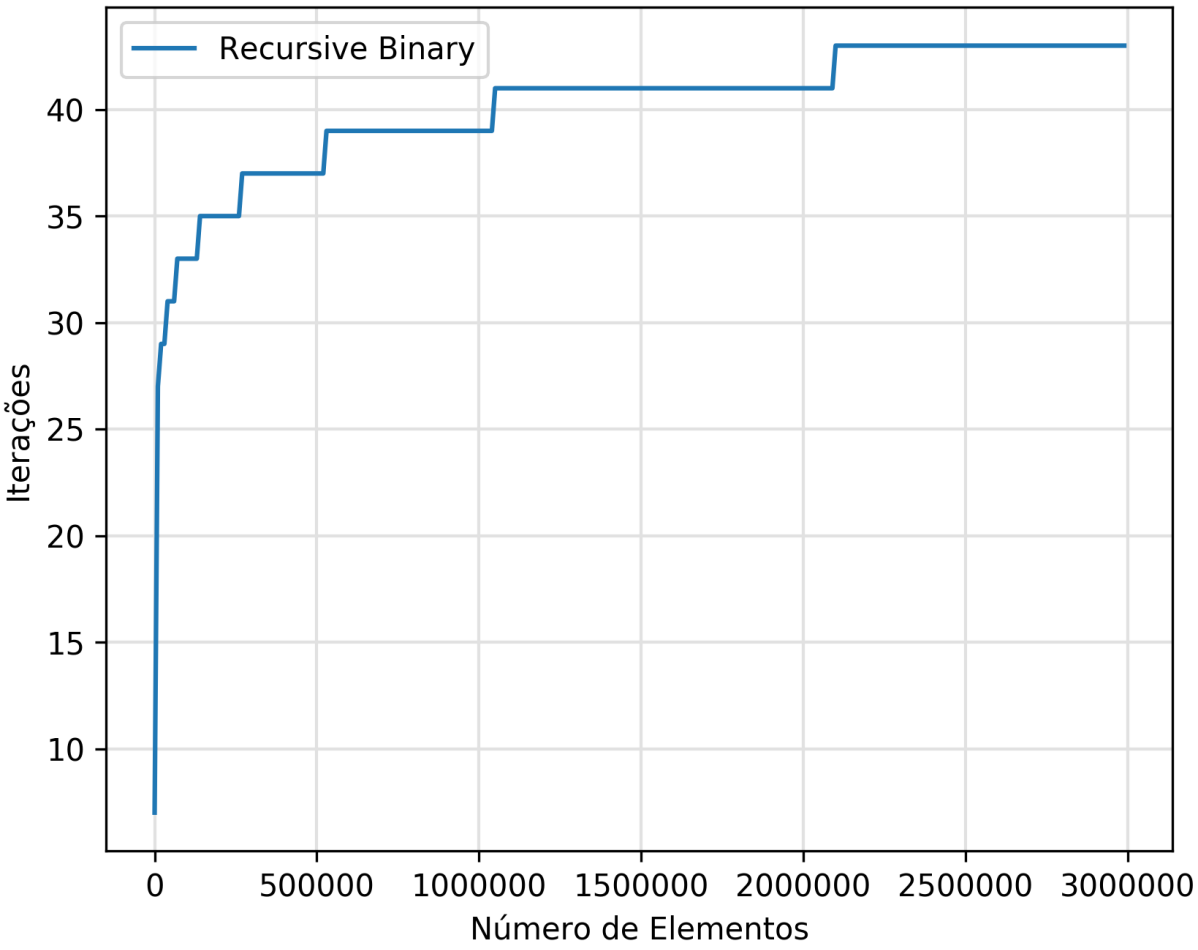


Recursiva

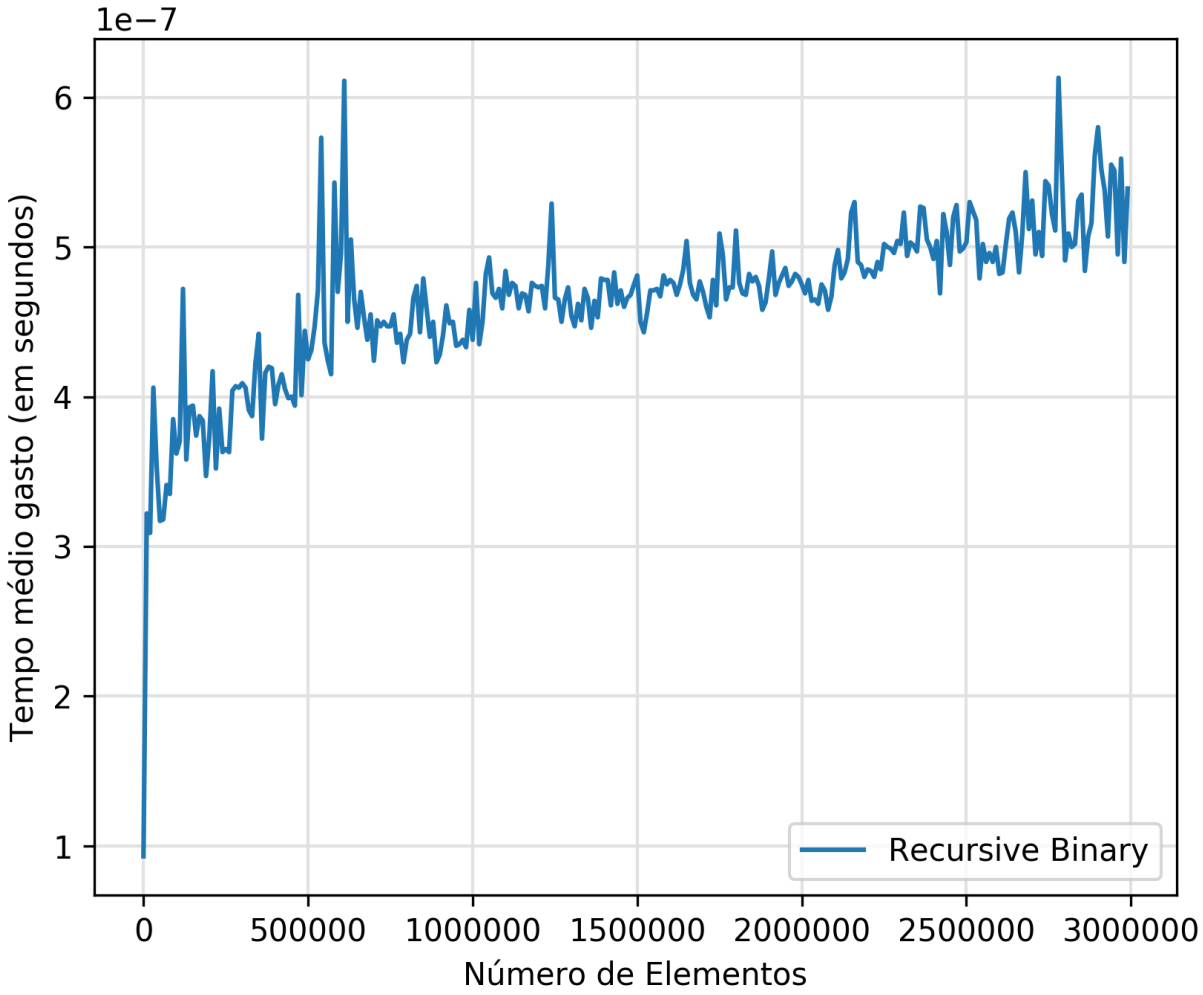
Apesar de possuir a mesma complexidade de tempo da Binária Iterativa, sua versão recursiva pode variar drasticamente em termos de memória consumida durante a execução (irá depender tanto do compilador quanto das otimizações feitas pelo programador).

Gráficos exclusivos

Tamanho x Iterações



Tamanho x Tempo médio



É de fato um método de busca imensamente mais eficiente que o algoritmo de busca linear. Porém uma de suas desvantagens aparece quando o vetor não é ordenado (seja por impossibilidade, fluxo de entrada constante, números repetidos ...) ou quando o vetor implementado não suporta *random acess*, como por exemplo, listas encadeadas.

Busca Ternária

Busca Ternária é uma técnica em ciência da computação para encontrar o mínimo ou o máximo de uma função unimodal. Uma busca ternária determina se o mínimo ou o máximo podem ou não estar no primeiro terço do domínio ou se ele pode ou não estar no último terço do domínio e, em seguida, repete o passo para o terceiro restante.

Iterativa

Seja $f(x)$ uma função unimodal num intervalo $[l; r]$. Pega-se dois pontos $m1$ e $m2$ no segmento: $l < m1 < m2 < r$

Depois, segue-se três passos:

1. Se $f(m1) < f(m2)$: Então o elemento buscado não pode ser localizado no lado esquerdo $[l; m1]$. Então olha-se apenas no intervalo $[m1; r]$
2. Se $f(m1) > f(m2)$, que a situação é semelhante à anterior, até a simetria. Agora, o elemento buscado não pode estar no lado direito $[m2; r]$, então olha-se no segmento $[l; m2]$

3. Se $f(m1) = f(m2)$, a busca deve ser realizada em $[m1; m2]$, mas este caso pode ser atribuído a qualquer um dos dois anteriores (para simplificar o código).

Recomendações para $m1$ e $m2$:

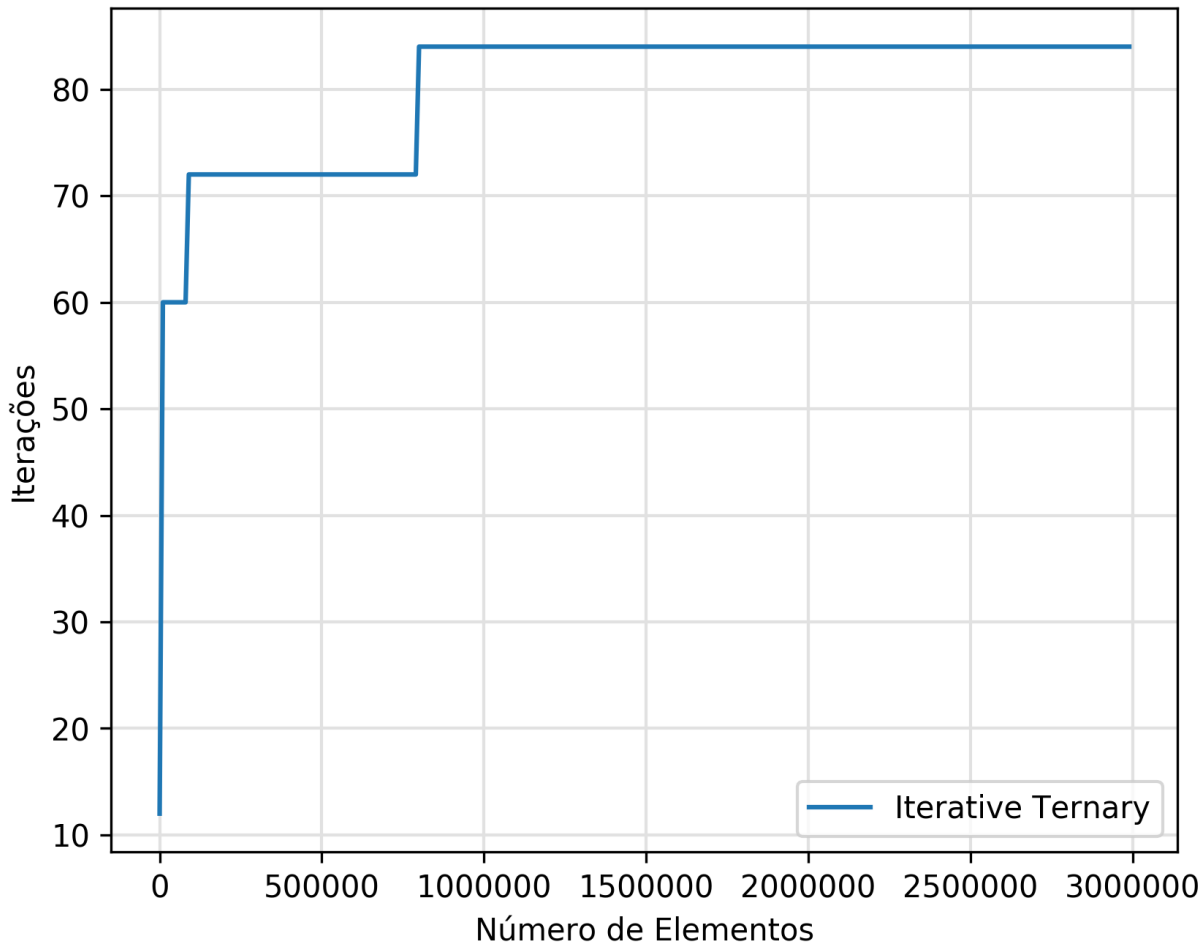
$m1 = \frac{l+(r-l)}{3}$

$m2 = \frac{r-(r-l)}{3}$

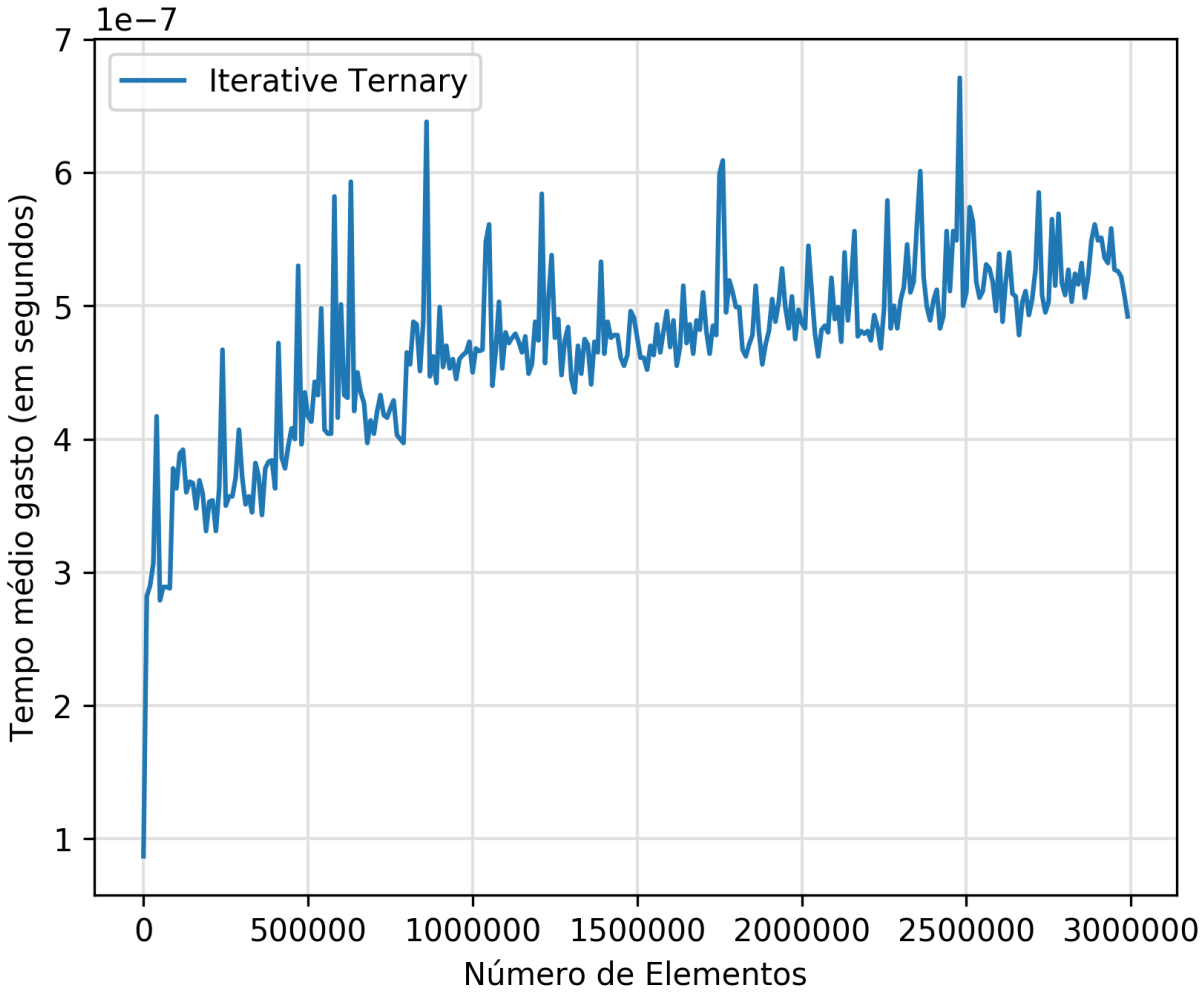
Devido a divisão recurssiva do vetor por 3, a função, em seu pior caso, assume comportamento logarítimico, o que gera uma complexidade de $\mathcal{O}(log_3 n)$.

Gráficos exclusivos

Tamanho x Iterações



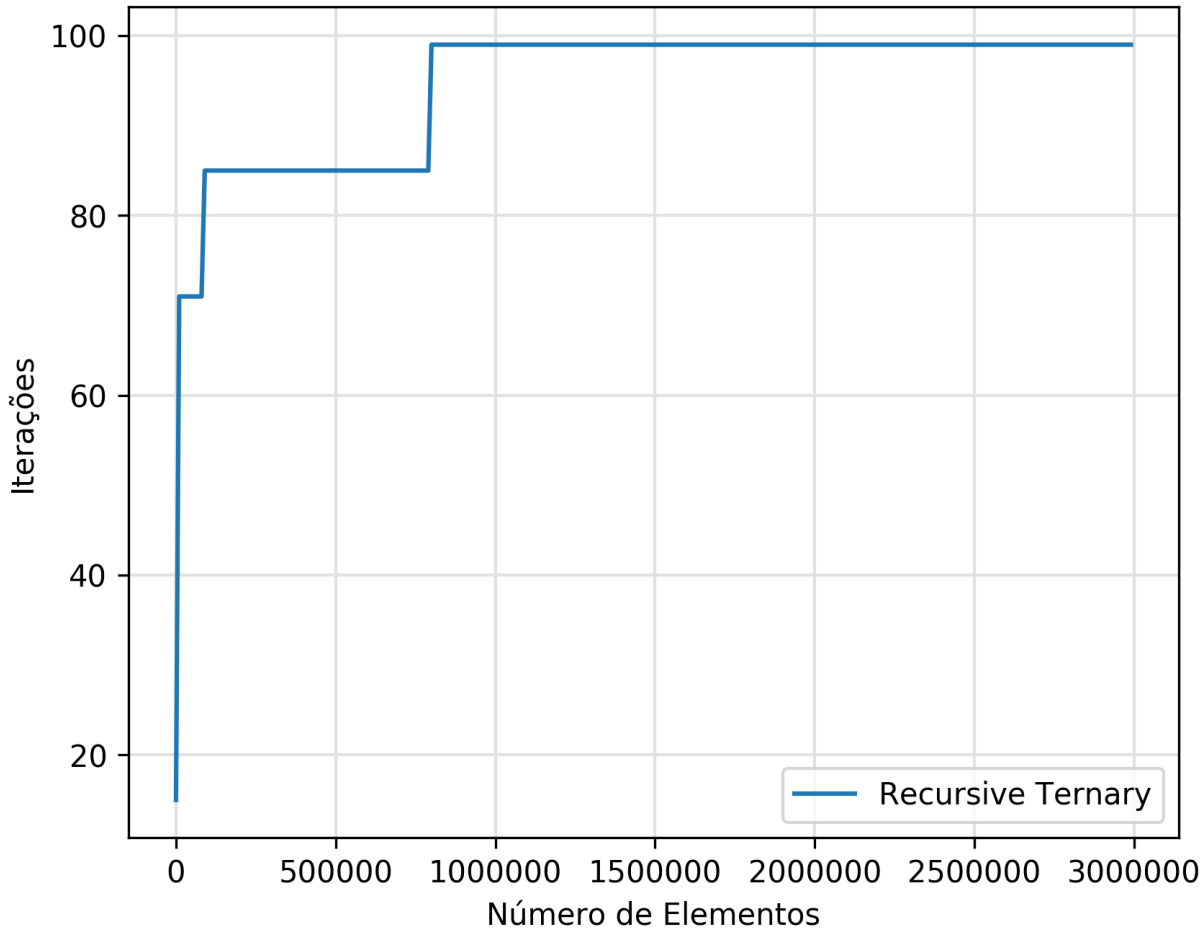
Tamanho x Tempo médio



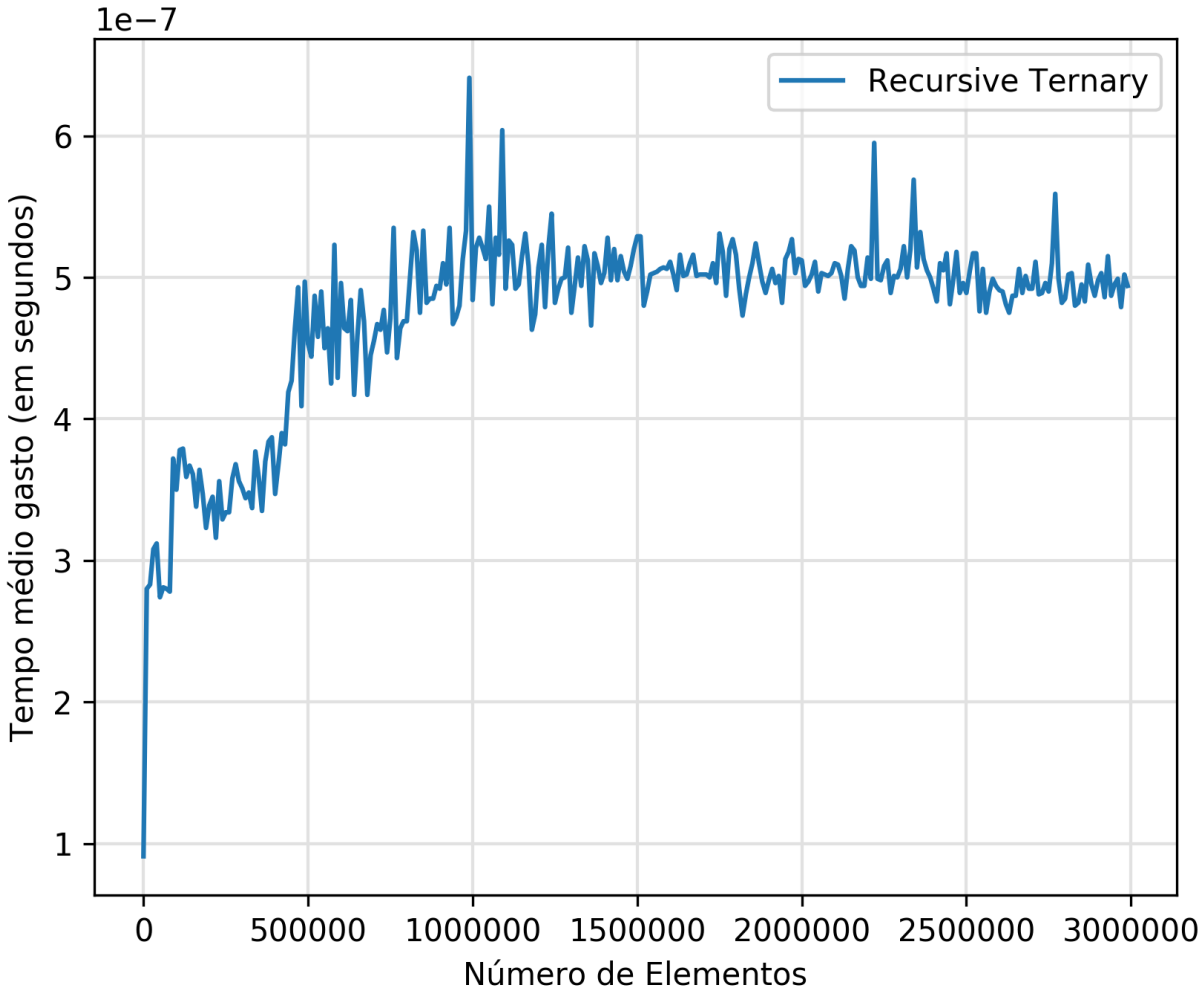
Recursiva

Gráficos exclusivos

Tamanho x Iterações



Tamanho x Tempo médio



É uma função pouco usada, visto que a busca binária faz basicamente o mesmo trabalho só que com uma comparação a menos em cada iteração, causando uma complexidade de $2\log_3(n)$, contra $\log_2(n)$ da busca binária.

Jump Search

Como a Binary Search, Jump Search, ou Block Search é um algoritmo de pesquisa para arrays ordenados. A idéia básica é verificar menos elementos (do que a busca linear) saltando por etapas fixas ou ignorando alguns elementos no lugar de pesquisar todos os elementos. Dada um vetor ordenado L, seu comprimento , queremos achar o elemento S.

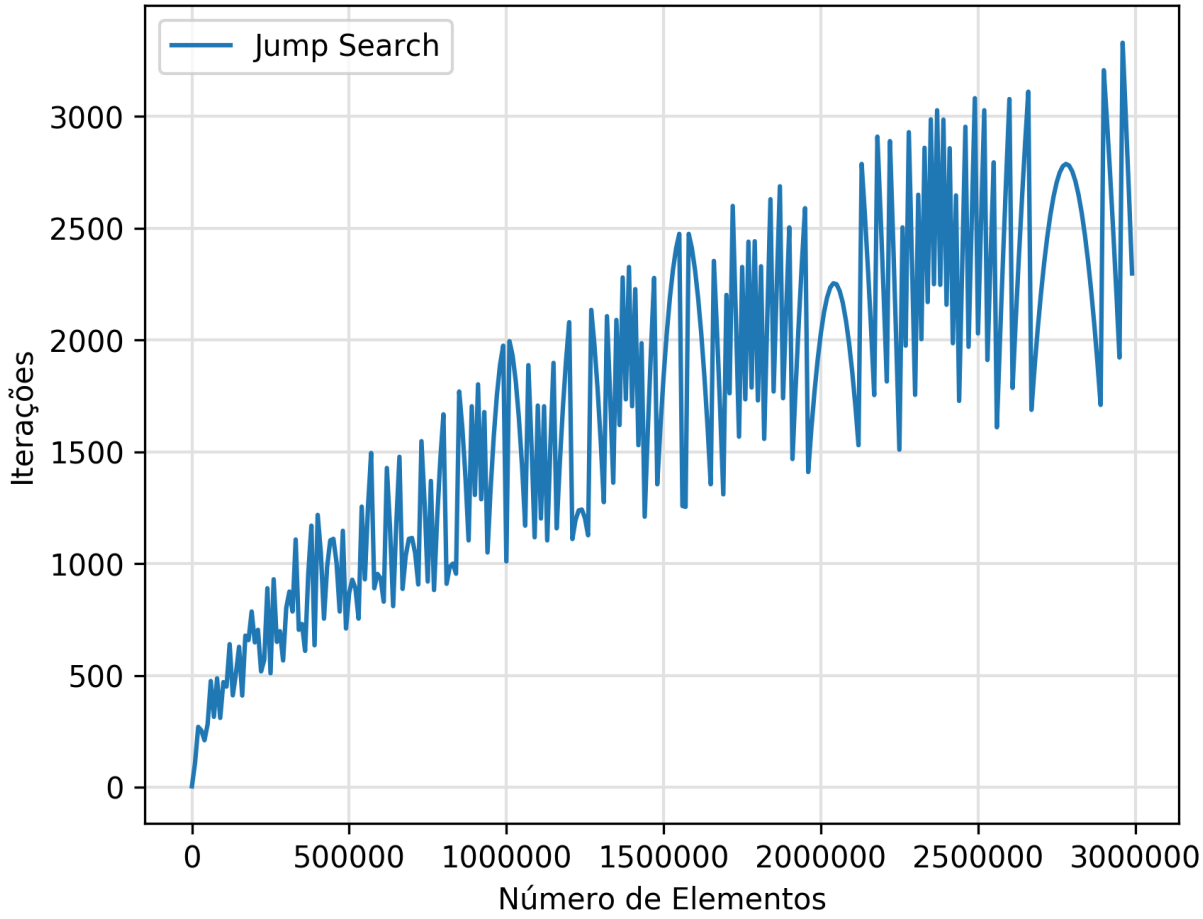
Sendo $L_{min(a,b)-1}$ o menor elemento entre a e b:

1. Iremos setar $a = 0$ e $b = 0$
2. Enquanto que $L_{min((b,n)-1)} < S$: Iremos setar $a = b$ e $b = b + \lfloor\sqrt{n}\rfloor$
3. Se $a \geq n$, o programa termina e é retornado `null`.
4. Enquanto que $L_a < S$: Iremos incrementar a em 1 unidade.
5. Se $a = min(b, n)$, o programa termina e é retornado `null`.
6. Se $L_a = S$, será retornado a.
7. Se não: é retornado `null`.

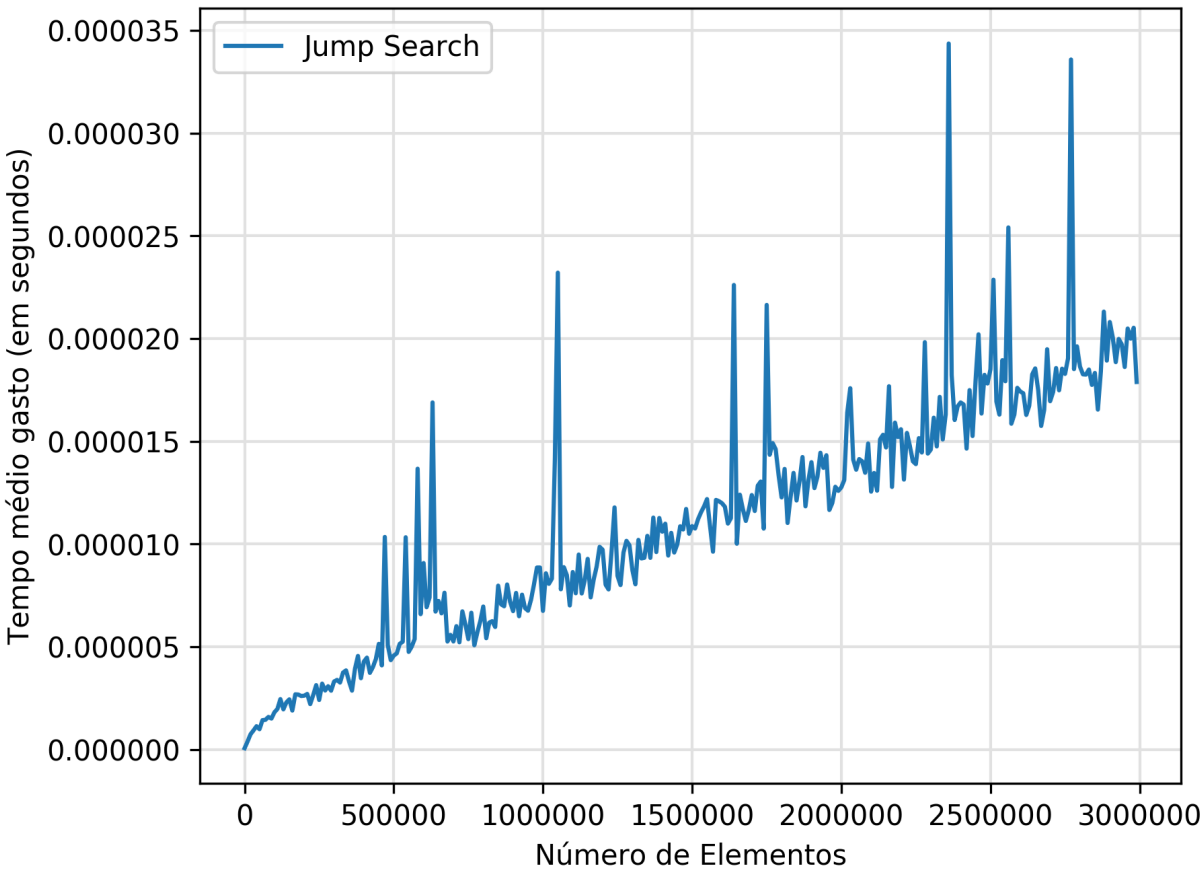
Devido a comparação dos blocos formados,ambas as etapas do algoritmo observam, no máximo, √n itens, o que leva o algoritmo a apresentar em seu pior caso complexidade $\mathcal{O}\sqrt{n}$.

Gráficos exclusivos

Tamanho x Iterações



Tamanho x Tempo médio



É de fato uma alternativa melhor que a busca sequencial, porém pior do que a busca binária. Pode ser um algoritmo útil caso empregado em casos específicos bem planejados, tais como um vetor que as chances do elemento buscado estar em uma das posições multiplas de $\sqrt{n}$ seja alta.

Busca Fibonacci

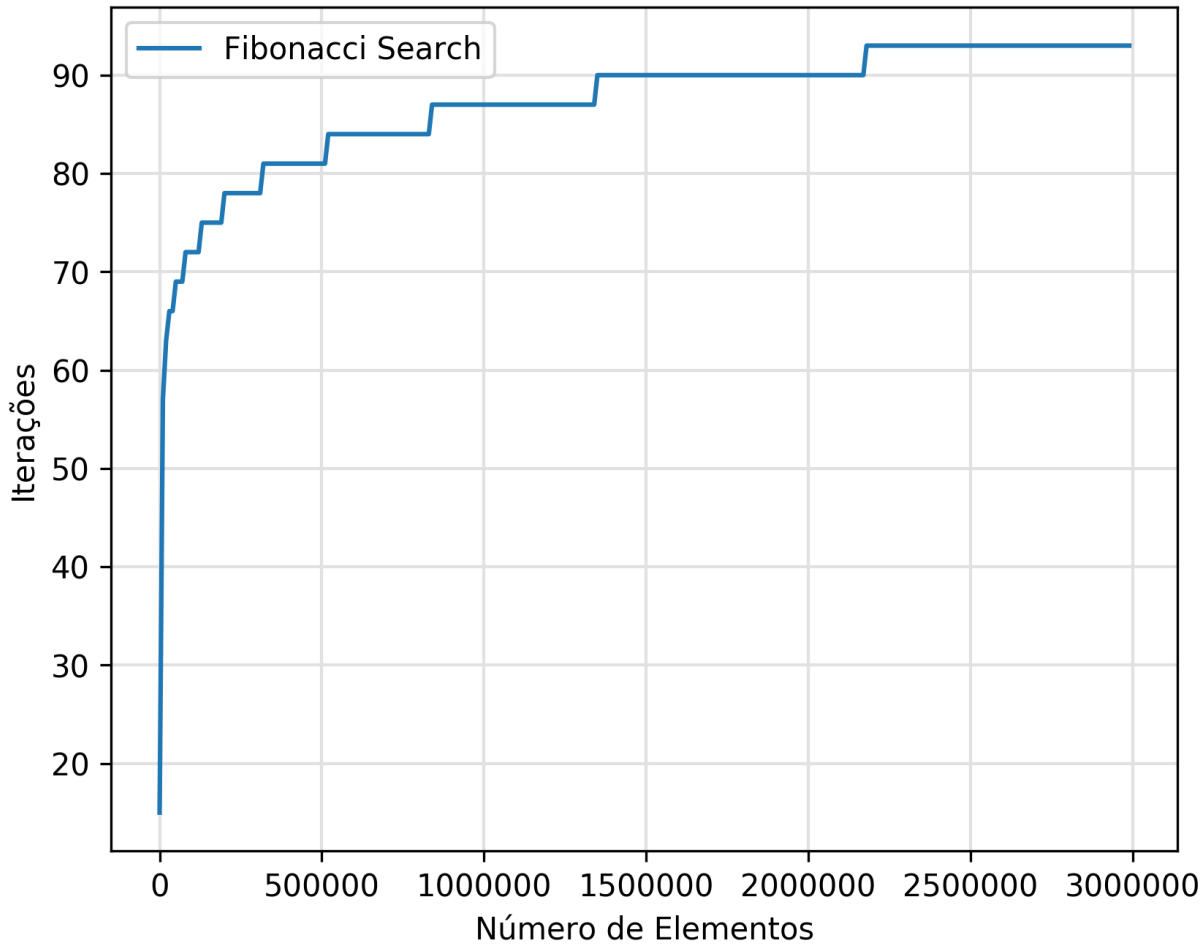
Fibonacci Search é uma técnica baseada em comparação, em um array ordenado, que empregando a filosofia *divide and conquer*, usa números da sequ6encia Fibonacci para pesquisar um elemento em um determinado vetor.

1. Encontra o menor número de Fibonacci maior ou igual ao valor passado para a busca (`value`).
2. Deixe que este número seja fib_m
3. Deixe que os dois números de Fibonacci que precedem sejam fib_{m-1} e fib_{m-2}.
4. Enquanto o vetor possui elementos a serem inspecionados:
    1. Compara `value` com o último elemento da gama coberta por fib_{m-2}
    2. Se `value` corresponder, retorna endereço.
    3. Se fib_m < `value`, reduz fib_{m-2}, fib_{m-1} e fib_m em dois números de fibonacci, indicando a eliminação de cerca de dois terços do vetor restante
    4. Se fib_m for menor que value, reduz $fibm-2$, fib_{m-1} e fib_m em três números de fibonacci, redefina o deslocamento para o endereço. Juntos, estes passos indicam a eliminação de cerca de um terço do vetor restante.

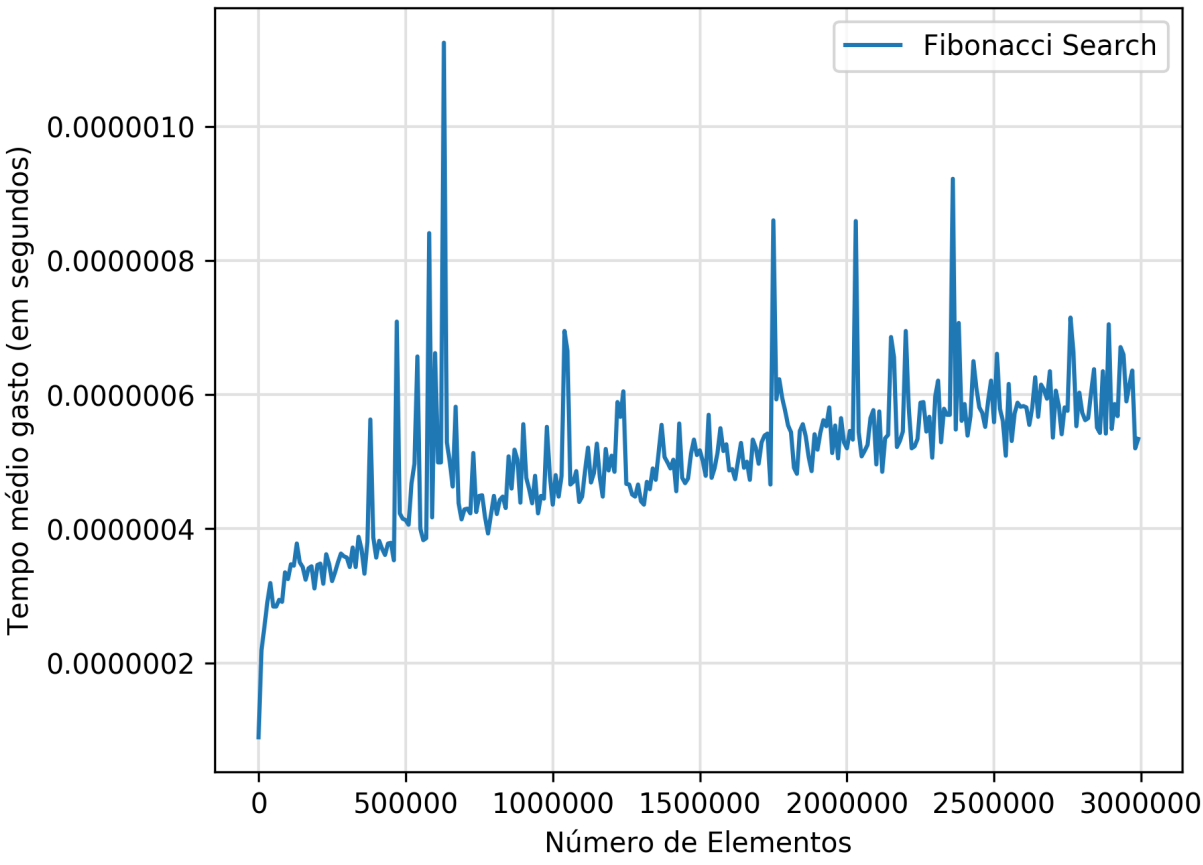
Devido a eliminação de ranges, em seu pior caso a busca Fibonacci acaba se tornando uma função logarítimica de complexidade $\mathcal{O} \log n$.

Gráficos exclusivos

Tamanho x Iterações



Tamanho x Tempo médio



No geral, a busca fibonacci leva a aproximadamente 4% mais comparações que a busca binária. Porém, sua real vantagem está no fato que só necessita de adições e subtrações (se implementada corretamente), o que torna o trabalho da CPU muito menos danoso do que em divisões (como na binária).

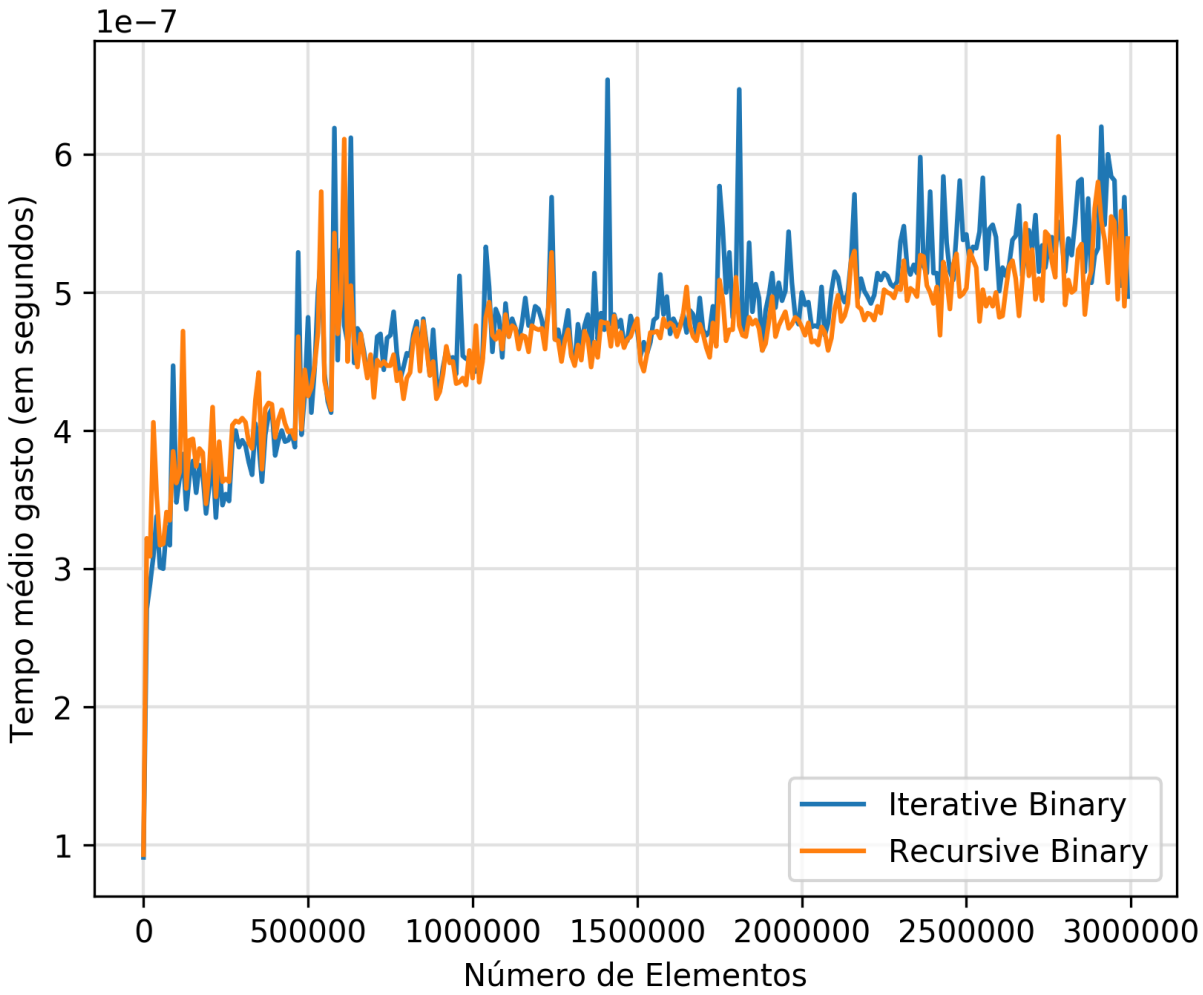
Quando os elementos estão em uma memória não totalmente uniforme (i.e. quando o tempo de acesso a determinadas partes da memória pode variar), ela também pode levar uma pequena vantagem em relação a busca binária pois reduz pouca coisa a quantidade de acessos à memória.

Comparações Gerais

Recursivos x Iterativos

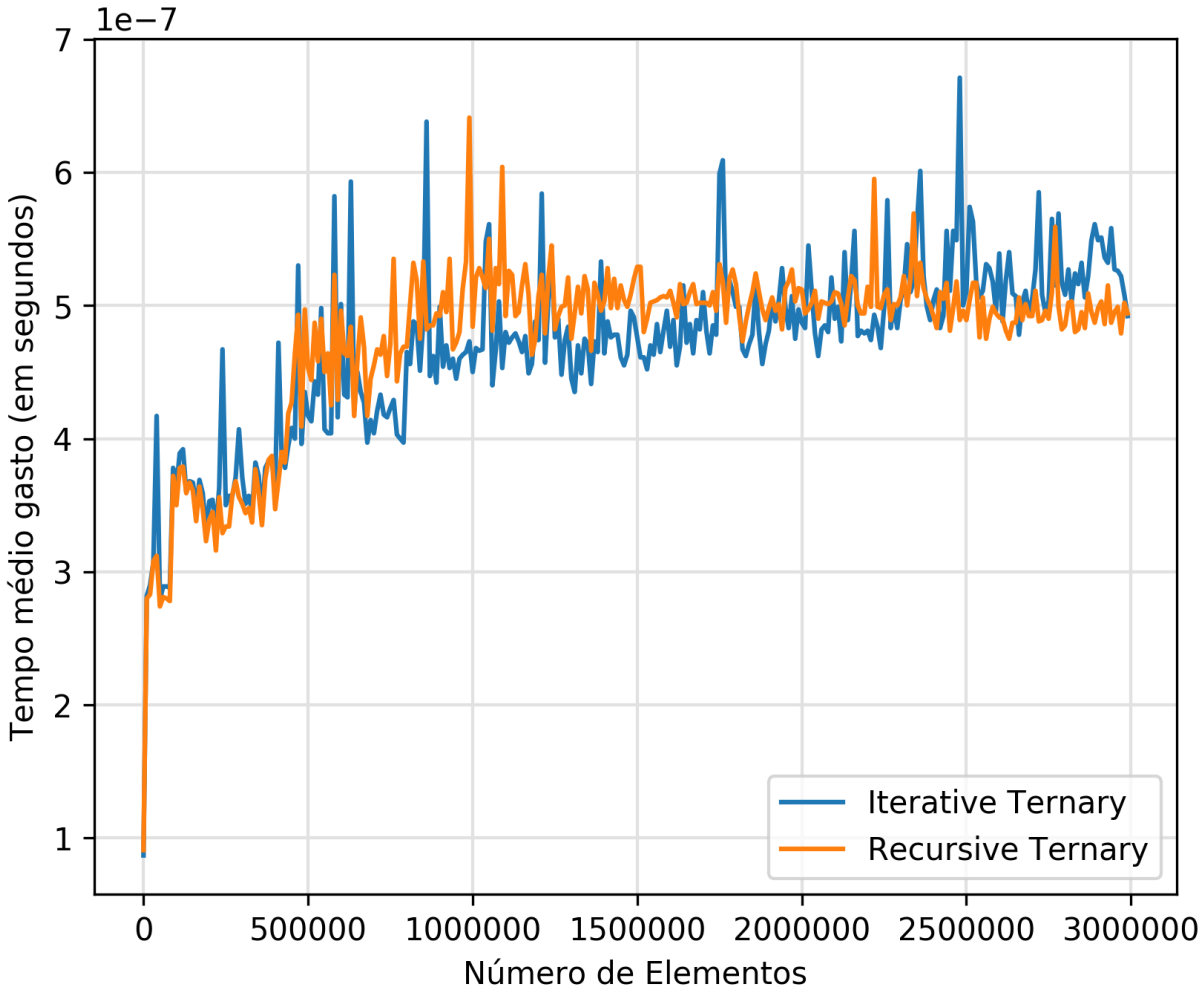
Busca Binária

A busca binária em suas duas versões apresenta complexidade de tempo semelhante, visto o gráfico abaixo. A única diferença entre as duas é que uma irá (provavelmente) consumir mais memória devido a recursividade.



Busca Ternária

Mesma aspecto observado na binária, única diferença é que a versão recursiva (provavelmente) consumirá mais memória devido a sua recursividade.



Recursivos x Busca Fibonacci

Tamanho x Iterações

Fibonacci x Recursive Binary

Fibonacci x Recursive Ternary

Tamanho x Tempo

Fibonacci x Recursive Binary

Fibonacci x Recursive Ternary

Recursivos x Jump Search

Tamanho x Iterações

Jump Search x Recursive Binary

Jump Search x Recursive Ternary

Tamanho x Tempo

Jump Search x Recursive Binary

Jump Search x Recursive Ternary

Condições de Testes

Informações sobre a maquina utilizada

- **Macbook Pro** (13-inch, 2017)
- **Processador** 2,3 GHz Intel Core i5
- **Memória** 8 GB 2133 MHz LPDDR3
- **Sistema** macOS High Sierra 10.13.3

Informações sobre os parametros utilizados

Todos as informações interpretadas neste documento foram obtidas utilizando o seguinte comando::

```
# Irá compilar e executar o arquivo binário, em seguida o script gerador de gráficos (src/gen_plot.py)
make run
```

Softwares utilizados

```
~$: g++ --version
Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 9.0.0 (clang-900.0.39.2)
Target: x86_64-apple-darwin17.4.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

```
~$: python3 --version
Python 3.6.4
```

```
~$: pip3 --version
pip 9.0.1 from /usr/local/lib/python3.6/site-packages (python 3.6)
```