UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO
ESTRUTURA DE DADOS BÁSICA I

RELATÓRIO DA IMPLEMENTAÇÃO E ANÁLISE EMPÍRICA DOS ALGORITMOS DE BUSCA

ANA CAROLINA BARBOSA DA SILVA
MARIANA EMERENCIANO MIRANDA

NATAL-RN
2023

ANA CAROLINA BARBOSA DA SILVA
MARIANA EMERENCIANO MIRANDA

RELATÓRIO DA IMPLEMENTAÇÃO E ANÁLISE EMPÍRICA DOS ALGORITMOS DE BUSCA

Primeiro Projeto apresentado à disciplina de Estrutura de Dados Básica I correspondente à 1ª unidade do semestre 2023.1 do 3º período do Bacharelado em Tecnologia da Informação da Universidade Federal do Rio Grande do Norte sob a orientação do **Prof. Selan Rodrigues dos Santos.**

Professor: Selan Rodrigues dos Santos.

NATAL-RN

2023

1. RESUMO

Este relatório trata-se do projeto inicial da disciplina Estruturas de Dados Básica I. Para o projeto, foram desenvolvidos cinco algoritmos de busca, sendo três deles executados para a comparação de desempenho e a análise de complexidade, com as mesmas instâncias de um problema. Para a análise, foram utilizados o programa *gnuplot* e a biblioteca Chrono C++ com a finalidade de visualizar graficamente os piores casos dos algoritmos.

Palavras-chave: Algoritmos de busca; Análise de complexidade; Estruturas de dados.

2. INTRODUÇÃO

Os algoritmos de busca em ciência da computação têm como base as operações de procura de um determinado elemento dentro de um conjunto de itens. Suas configurações em linguagens computacionais foram desenvolvidas em consonância com o surgimento dos primeiros computadores, mas seus registros começaram a surgir a partir de 1970.

Nos estudos de computação, são apresentados algoritmos que usam diferentes estratégias para solucionar os problemas, a depender do grupo de dados no qual será feita a pesquisa.

Desse modo, geramos os dados de cada algoritmo, assumindo como critério o tempo levado para operação, de modo que retornemos o pior caso para os gráficos para assim, analisar uma análise empírica de eficiência.

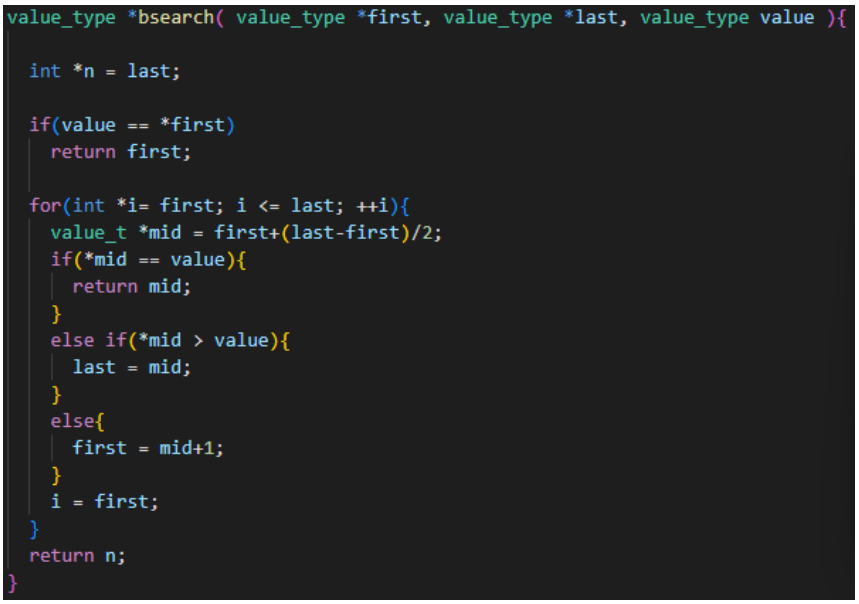
3. DESCRIÇÃO DAS ATIVIDADES E RESULTADOS

A fim de compreender verdadeiramente a execução desses algoritmos (e as variações deles), escrevemos códigos com a finalidade de encontrar um dado valor em um array ordenado. Ainda, comparamos a performance das operações sob as mesmas circunstâncias e incremento do tamanho do array.

Inicialmente, implementamos duas funções interativas: busca binária e busca linear. A busca binária parte do pressuposto de que o array está ordenado e faz sucessivas divisões do conjunto até encontrar o valor procurado. Caso o valor seja menor que o elemento que se encontra no meio, o último elemento passa a ter o valor do meio; caso contrário, o primeiro

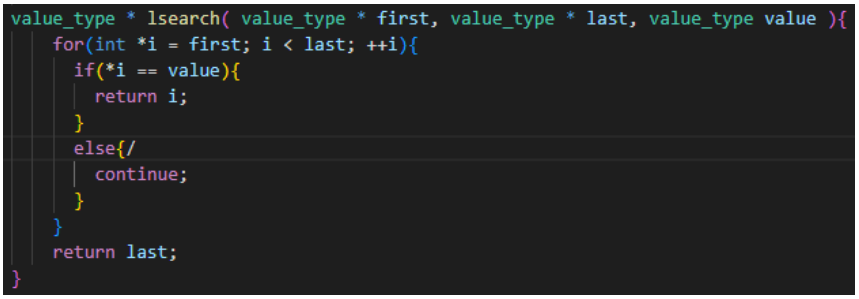
elemento passa a ter o valor seguinte ao do elemento do meio. Já a busca linear procura sequencialmente o valor indicado no grupo de elementos. Caso o valor não seja igual ao indicado, passa para o elemento seguinte.

Imagem 1 - Código implementado para busca binária (iterativa)



```
value_type *bsearch( value_type *first, value_type *last, value_type value ){

  int *n = last;

  if(value == *first)
    return first;

  for(int *i= first; i <= last; ++i){
    value_t *mid = first+(last-first)/2;
    if(*mid == value){
      return mid;
    }
    else if(*mid > value){
      last = mid;
    }
    else{
      first = mid+1;
    }
    i = first;
  }
  return n;
}
```

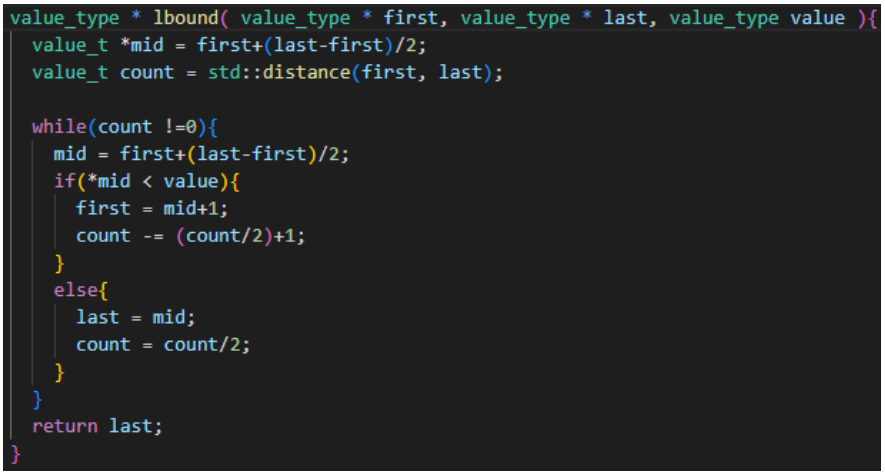
Imagem 2 - Código implementado para busca linear



```
value_type * lsearch( value_type * first, value_type * last, value_type value ){
    for(int *i = first; i < last; ++i){
      if(*i == value){
        return i;
      }
      else{/
        continue;
      }
    }
    return last;
}
```

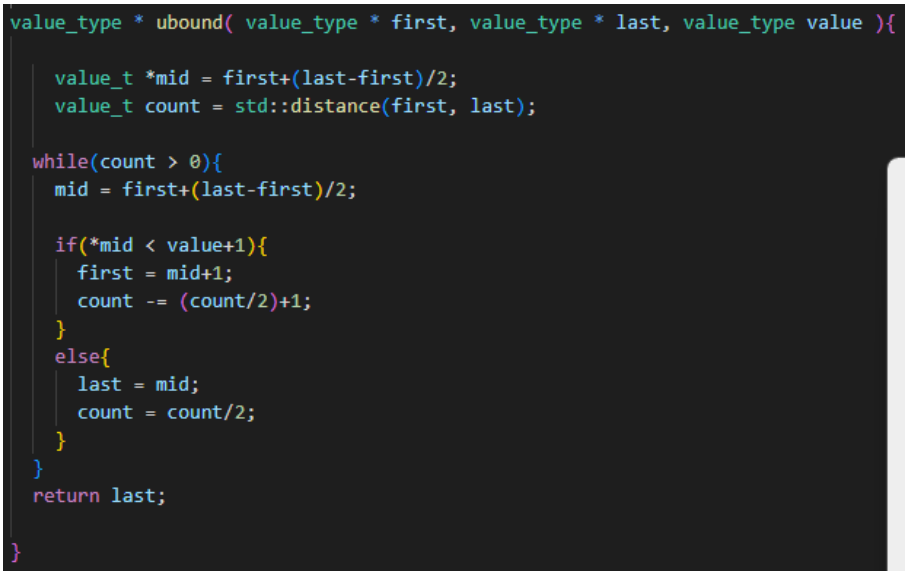
Implementados os códigos iterativos de busca, começamos as variações de busca binária: lower bound e upper bound. Ambos partem do pressuposto de que o array está ordenado, porém com elementos repetidos. A primeira variação retorna o índice da primeira ocorrência do valor procurado e, caso não o encontre, retorna o primeiro valor que não é menor que o elemento alvo. Já a segunda variação, o índice retornado é o do item seguinte à última ocorrência de elementos repetidos e, caso não encontre o elemento indicado, retornará o primeiro item maior que o alvo.

Imagem 3 - Implementação do lower bound (variação da busca binária)



```
value_type * lbound( value_type * first, value_type * last, value_type value ){
  value_t *mid = first+(last-first)/2;
  value_t count = std::distance(first, last);

  while(count !=0){
    mid = first+(last-first)/2;
    if(*mid < value){
      first = mid+1;
      count -= (count/2)+1;
    }
    else{
      last = mid;
      count = count/2;
    }
  }
  return last;
}
```

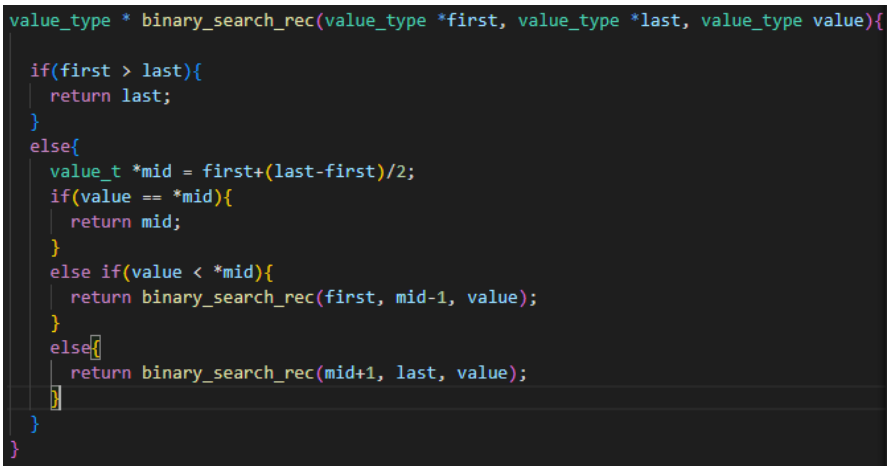
Imagem 4 - Implementação do upper bound (variação da busca binária)



```
value_type * ubound( value_type * first, value_type * last, value_type value ){

    value_t *mid = first+(last-first)/2;
    value_t count = std::distance(first, last);

  while(count > 0){
    mid = first+(last-first)/2;

    if(*mid < value+1){
      first = mid+1;
      count -= (count/2)+1;
    }
    else{
      last = mid;
      count = count/2;
    }
  }
  return last;

}
```

Após estes algoritmos e variações, trabalhamos na escrita do algoritmo recursivo de busca binária, com a mesma estratégia da iterativa, mas sem mudança de ponteiros, e sim com o retorno da função com argumentos diferentes.

Imagem 5 - Código implementado da busca binária (recursiva)



```
value_type * binary_search_rec(value_type *first, value_type *last, value_type value){

  if(first > last){
    return last;
  }
  else{
    value_t *mid = first+(last-first)/2;
    if(value == *mid){
      return mid;
    }
    else if(value < *mid){
      return binary_search_rec(first, mid-1, value);
    }
    else{
      return binary_search_rec(mid+1, last, value);
    }
  }
}
```

Por fim, testamos todos os códigos com os testes do professor e, com a aprovação total dos casos, começamos a comparação de desempenho dos algoritmos dado o incremento de tamanho do array.

Para isso, utilizamos uma máquina de sistema operacional Linux Mint 21 Cinnamon com processador Intel © Core i5-7600K CPU @ 3.80 GHz x 4, 16GB de memória RAM, disco rígido com 2TB e placa de vídeo NVIDIA Corporation GP106 [GeForce GTX 1060 6GB].

Gráfico 1 - O tempo de execução dos algoritmos de buscas (binária e linear) vs. o tamanho do array

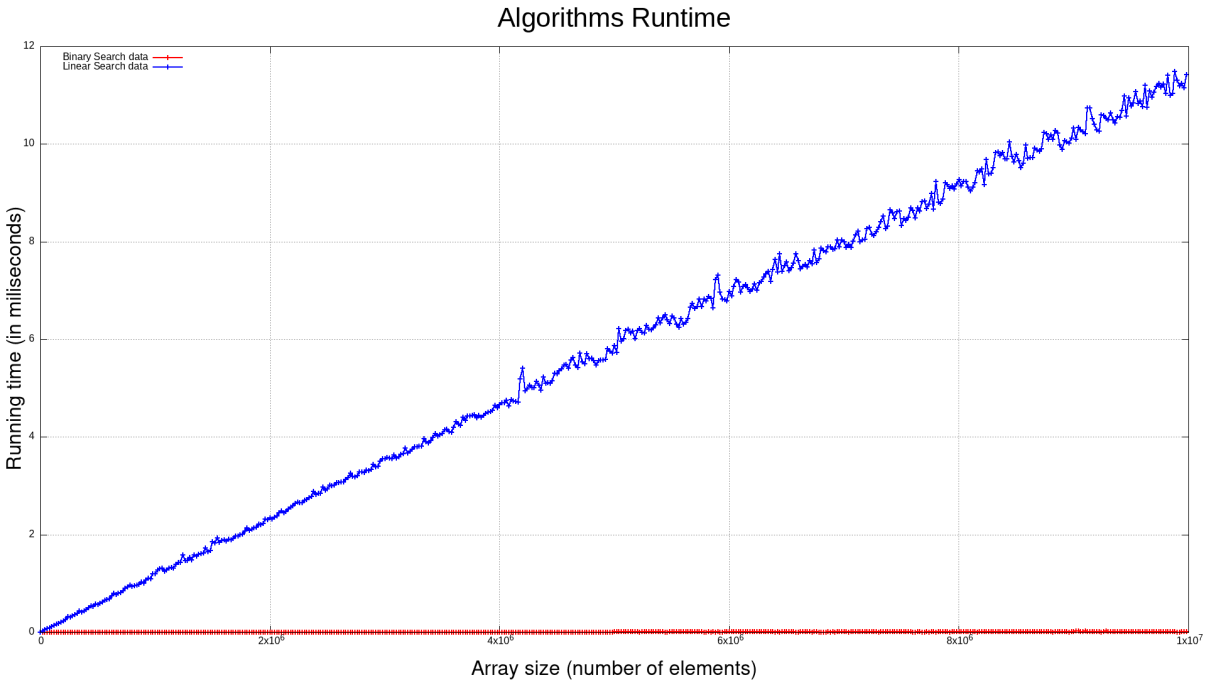


Gráfico 2 - O tempo de execução dos algoritmos de buscas binárias (iterativa e recursiva) vs. o tamanho do array

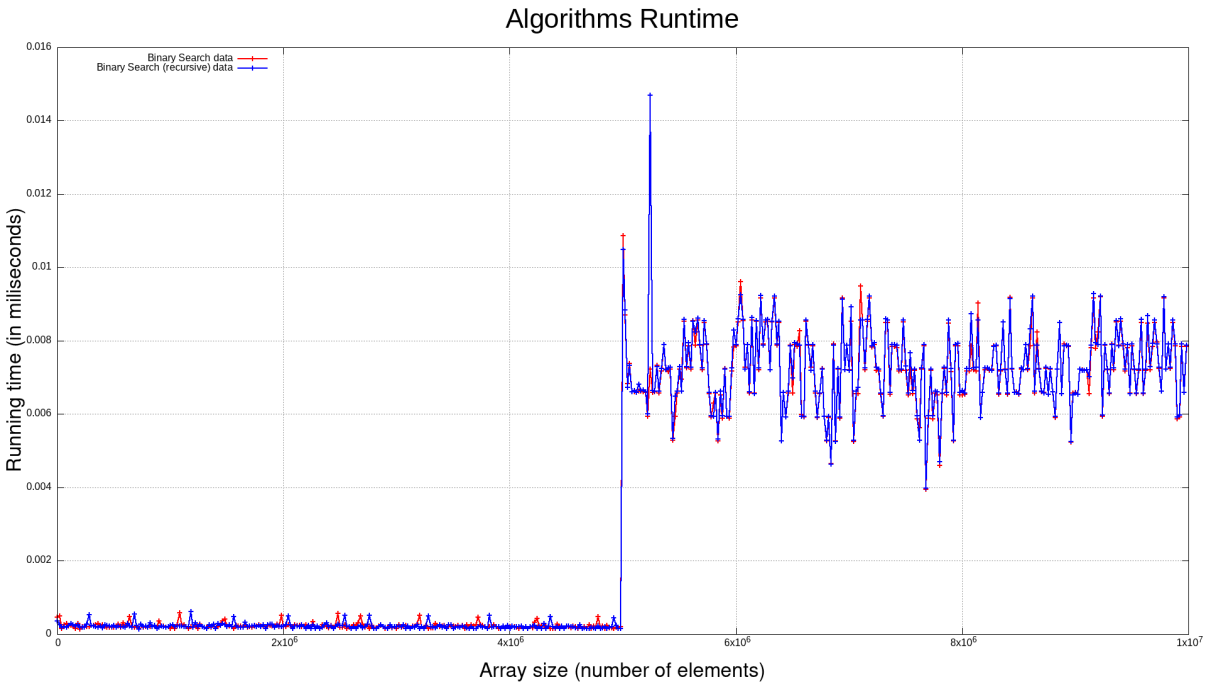
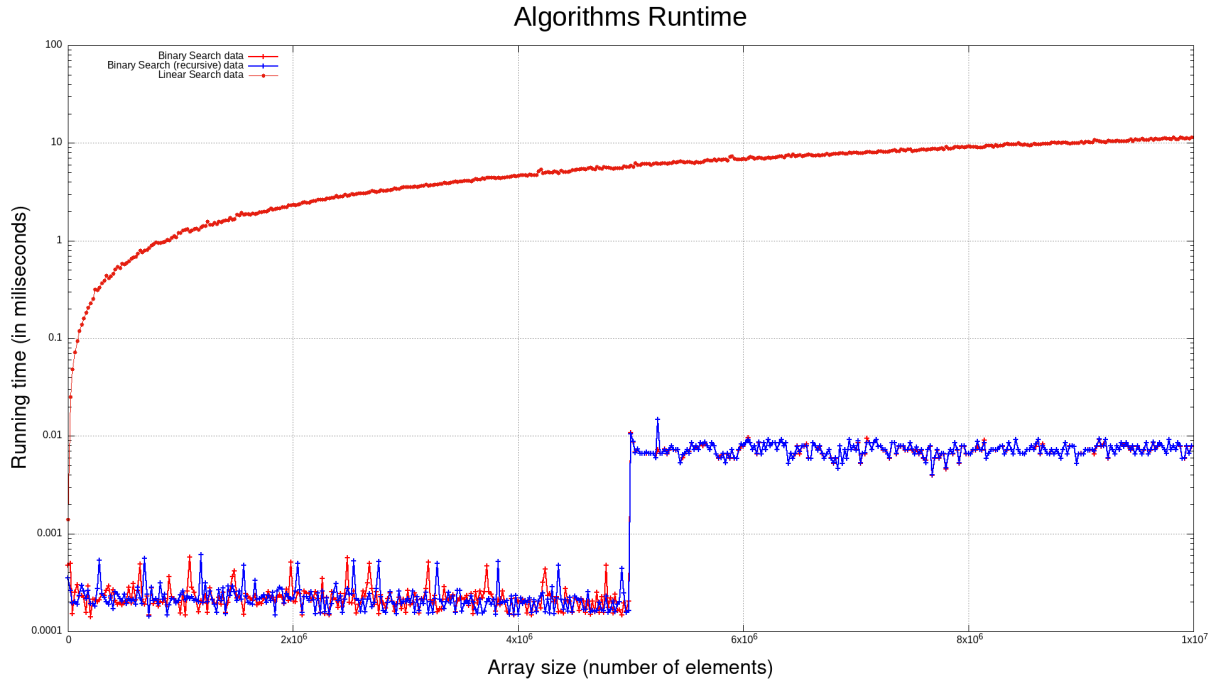


Gráfico 3 - O tempo de execução dos três algoritmos de buscas vs. o tamanho do array (em escala logarítmica)



4. DISCUSSÃO

É perceptível que, à medida que o tamanho do vetor aumenta, a busca linear leva mais tempo para ser finalizada, em contraste com a busca binária, que mantém sua variação de tempo entre 10^{-4} e 10^{-3} ms, conforme o gráfico 1. Certamente, de acordo com a notação Big O, utilizada para classificar os algoritmos pelas respostas (neste caso, o tempo de processamento) dada uma variação na entrada de dados, o pior caso da busca linear apresenta complexidade O(n), isto é, realiza n operações sobre n elementos de entrada , em contraste com a complexidade do pior caso da busca binária, que é O(log n), ou seja, divide o problema em pedaços menores.

Além disso, observando o Gráfico 2 das funções binárias iterativa e recursiva, houve uma certa discrepância entre o que se esperava da recursão e os resultados apresentados. Apesar da busca binária recursiva aumentar as chamadas da própria função de acordo com o aumento do tamanho do vetor, ela não apresentou uma perda de desempenho significativa. Como mencionado, a complexidade da busca binária em seu pior caso equivale a O(log n) e, mesmo que fosse esperado maior consumo de memória e tempo para finalizar as sucessivas

operações, a busca binária recursiva mostra-se tecnicamente empatada com a busca iterativa, uma vez que o pior caso de variação de tempo não ultrapassou 0.007ms.

5. CONCLUSÃO

Diante do exposto, a partir das análises da eficiência dos algoritmos em seus piores casos, é possível inferir que, executando a busca linear sob as mesmas circunstâncias das demais operações, esta possui o pior desempenho na análise de complexidade, pois em contrapartida, a busca binária iterativa apresenta complexidade logarítmica. Note que a busca binária transforma uma grande entrada em valores ínfimos, mas necessita da garantia de ordenação dos elementos, divergindo do diferencial da busca linear, que é capaz de localizar o item procurado mesmo em um conjunto desordenado.

Sob esta perspectiva, convém ressaltar também os desempenhos distintos das buscas binárias e iterativa e recursiva. Apesar da esperada discrepância entre os resultados, não foi possível observar uma vantagem da busca iterativa sobre a recursiva, mesmo que esta consuma mais memória do que aquela.

Portanto, entende-se que, sob os mesmos parâmetros, as funções binárias atendem otimamente ao critério proposto, ainda que somente a busca linear permita a análise de uma amostra desordenada.

6. REFERÊNCIAS BIBLIOGRÁFICAS

trabalho-01-algoritmos-de-busca-duplas-don-t-rain-on-my-parray created by Selan Rodrigues dos Santos, 2023. Disponível em: https://github.com/selan-active-classes/trabalho-01-algoritmos-de-busca-duplas-don-t-rain-on-my-parray. Acesso em 13 abr. 2023