Lista de exercício 1

Aluno	Matrícula
Arthur Temporim	14/0016759
Bruno Bragança	09/0107853

Respostas

Todas as respostas podem ser encontradas no seguinte repositório: EDA

1.

- Músicos:
  - João
  - Antônio
  - Francisco
- Instrumentos:
  - Harpa
  - Violino
  - Piano
- Informações:
  - Antônio não é pianista.
  - Pianista ensaia sozinho na terça.
  - João ensaia com o Violinista às Quintas.
- Resposta:
  - João toca Harpa
  - Antônio toca Violino
  - Francisco toca Piano

2.

- Informações:
  - 1 prisioneiro em 1 cela com 2 saidas.
  - Cada saída tem 1 guarda.
  - 1 saída -> liberdade.
  - 1 saída -> morte.
  - Os guardas sabem a resposta.
  - Um só mente e outro só fala a verdade.
- Resposta: A pergunta a ser feita deve ser:

Se eu perguntar para o outro guarda qual é a saída, qual será a resosta dele?

Pois, o guarda que fala a verdade vai indicar que a saída certa é a errada, e o guarda que fala a mentira vai indicar também a saída errada, pois iria mentir sobre a resposta do guarda que fala a verdade.

Logo para se livrar, o prisioneiro deve ir para a saída contrária ao que o guarda indicar.

3.

- Informações:
  - Foi pago R$ 30,00.
  - O valor da diária é R$ 25,00.
  - Troco a ser entregue R$ 5,00.
  - Valor embolsado pelo mensageiro R$ 2,00.
- Resposta: O enunciado tende a induzir o leitor a uma informação inválida.

  Se cada pessoa recebeu R$ 1,00, na verdade o total pago foi R$ 27,00. Levando em conta que o mensageiro pegou R$ 2,00, os R$ 25,00 restantes são exatamente, o valor pago da diária.

  Não existe R$ 1,00 sumido, a questão é considerar os R$ 3,00 dados a cada pessoa como desconto dos R$ 30,00 dados inicialmente.

4.

```
#include <iostream>
#include <cstdio>
#include <string>

using namespace std;

int * initialize_vector(int * vector, int vector_length);
void print_vector(int * vector, int vector_length);
int * initialize_primary_index(int * primary_index, int primary_index_lenght, int * vector,
void print_primary_index(int * primary_index, int primary_index_length);
void print_menu();
string search(int value, int * primary_index, int* vector, int gap);


int main() {

    const int length = 100;
    const int primary_index_lenght = 10;
    const int gap = 10;
```

```
    int vector[length];
    int primary_index[primary_index_lenght];

    initialize_vector(vector, length);
    print_vector(vector, length);

    initialize_primary_index(primary_index, primary_index_lenght, vector, gap);
    print_primary_index(primary_index, primary_index_lenght);

    print_menu();
    int value=0;
    cin >> value;

    string result;
    result =  search(value, primary_index, vector, gap);
    cout << result << endl;

    return 0;
}

string search(int value, int *primary_index, int * vector, int gap) {
    string response = "NOT FOUNDED!";
    for(int i=0;i<sizeof(primary_index);i++) {
        if(value < primary_index[i]) {
            for(int j=0;j<gap;j++) {
                if(value == vector[j]) {
                    response = "FOUNDED!";
                } else {
                    // nothing to do.
                }
            }
        } else {
            // nothing to do.
        }
    }
    return response;
}

int * initialize_vector(int * vector, int vector_length) {
  for(int i=0;i<vector_length;i++) {
    vector[i] = i;
  }
    return vector;
}

void print_vector(int * vector, int vector_length) {
```

```
    cout << "---------------" << endl;
    cout << "VECTOR" << endl;
    cout << "---------------" << endl;
  cout << '[';
  for(int i=0;i<vector_length;i++) {
    cout << vector[i] << " ";
  }
  cout << ']';
    cout << endl;
    cout << "---------------" << endl;
}

int * initialize_primary_index(int * primary_index,
                                            int primary_index_lenght, int * vector,
                                            int gap) {
    int j=0;
    for(int i=0;i<primary_index_lenght;i++) {
        primary_index[i] = vector[j];
        j +=gap;
    }
}

void print_primary_index(int * primary_index, int primary_index_length) {
    cout << "PRIMARY INDEX" << endl;
    cout << "---------------" << endl;
  for(int i=0;i<primary_index_length;i++) {
    cout << primary_index[i] << endl;
  }
    cout << "---------------" << endl;
}

void print_menu() {
    cout << "---------------" << endl;
    cout << "PRINT A VALUE TO SEARCH" << endl;
    cout << "---------------" << endl;
}
```

5.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
```

```
#define ARRAYS_SIZE 9

unsigned long long binary_search(unsigned long long*, unsigned long long,
                                                                unsigned long long, unsigne

int main (int argc, char *argv[]) {
  unsigned long long arrays_size[ARRAYS_SIZE] = {10, 25, 50, 100, 500, 1000,
                                                                        10000, 100000,
  unsigned long long *array, array_iterator, size_iterator, entry_value;
  clock_t inicio,fim;
  double tempo_lista;


  srand(time(NULL));

  for (size_iterator = 0; size_iterator < ARRAYS_SIZE; size_iterator++) {
    array = (unsigned long long*)malloc(sizeof(unsigned long long) *
                        arrays_size[size_iterator]);
    for (array_iterator = 0; array_iterator < arrays_size[size_iterator];
                 array_iterator++) {
      array[array_iterator] = array_iterator;
    }

    entry_value = rand() % arrays_size[size_iterator];
    inicio = clock();
    binary_search(array, 0, arrays_size[size_iterator]-1, entry_value);
    fim = clock();
    tempo_lista = (double) (fim-inicio)/CLOCKS_PER_SEC;
    printf("array size: %lu, random generated value: %lu, total time: %lf\n",
                        arrays_size[size_iterator], entry_value, tempo_lista);
  }


  return 0;
}

unsigned long long binary_search(unsigned long long *array,
                                                                unsigned long long min,
                                                                unsigned long long max,
                                                                unsigned long long value) {
  double mid_pos = min + (max - min) * ((value - array[min]) /
                                        (array[max] - array[min]));
  if (value == array[(unsigned long long) mid_pos]) {
    return (unsigned long long) mid_pos;
  }
  if (min >= max) {
```

```
    return -1;
  }else {
    if (array[(unsigned long long) mid_pos] > value) {
      return binary_search(array, min, (unsigned long long) mid_pos-1, value);
    } else {
      return binary_search(array, (unsigned long long) mid_pos+1, max, value);
    }
  }
}
```

6.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>


#define ARRAYS_SIZE 9

unsigned long long binary_search(unsigned long long*, unsigned long long,
                                                                  unsigned long long, unsigne

int main (int argc, char *argv[]) {
  unsigned long long arrays_size[ARRAYS_SIZE] = {10, 25, 50, 100, 500, 1000,

  unsigned long long *array, array_iterator, size_iterator, entry_value;
  clock_t inicio,fim;
  double tempo_lista;


  srand(time(NULL));

  for (size_iterator = 0; size_iterator < ARRAYS_SIZE; size_iterator++) {
    array = (unsigned long long*)malloc(sizeof(unsigned long long) *
                        arrays_size[size_iterator]);
    for (array_iterator = 0; array_iterator < arrays_size[size_iterator];
                array_iterator++) {
      array[array_iterator] = array_iterator;
    }

    entry_value = rand() % arrays_size[size_iterator];
    inicio = clock();
    binary_search(array, 0, arrays_size[size_iterator]-1, entry_value);
    fim = clock();
```

```
    tempo_lista = (double) (fim-inicio)/CLOCKS_PER_SEC;
    printf("array size: %lu, random generated value: %lu, total time: %lf\n",
                        arrays_size[size_iterator], entry_value, tempo_lista);
  }


  return 0;
}

unsigned long long binary_search(unsigned long long *array,
                                                                  unsigned long long min,
                                                                  unsigned long long max,
                                                                  unsigned long long value) {
  double mid_pos = min + (max - min) * ((value - array[min]) /
                                        (array[max] - array[min]));
  if (value == array[(unsigned long long) mid_pos]) {
    return (unsigned long long) mid_pos;
  }
  if (min >= max) {
    return -1;
  }else {
    if (array[(unsigned long long) mid_pos] > value) {
      return binary_search(array, min, (unsigned long long) mid_pos-1, value);
    } else {
      return binary_search(array, (unsigned long long) mid_pos+1, max, value);
    }
  }
}
```