Relatório Projeto 1 AED 2024-2025

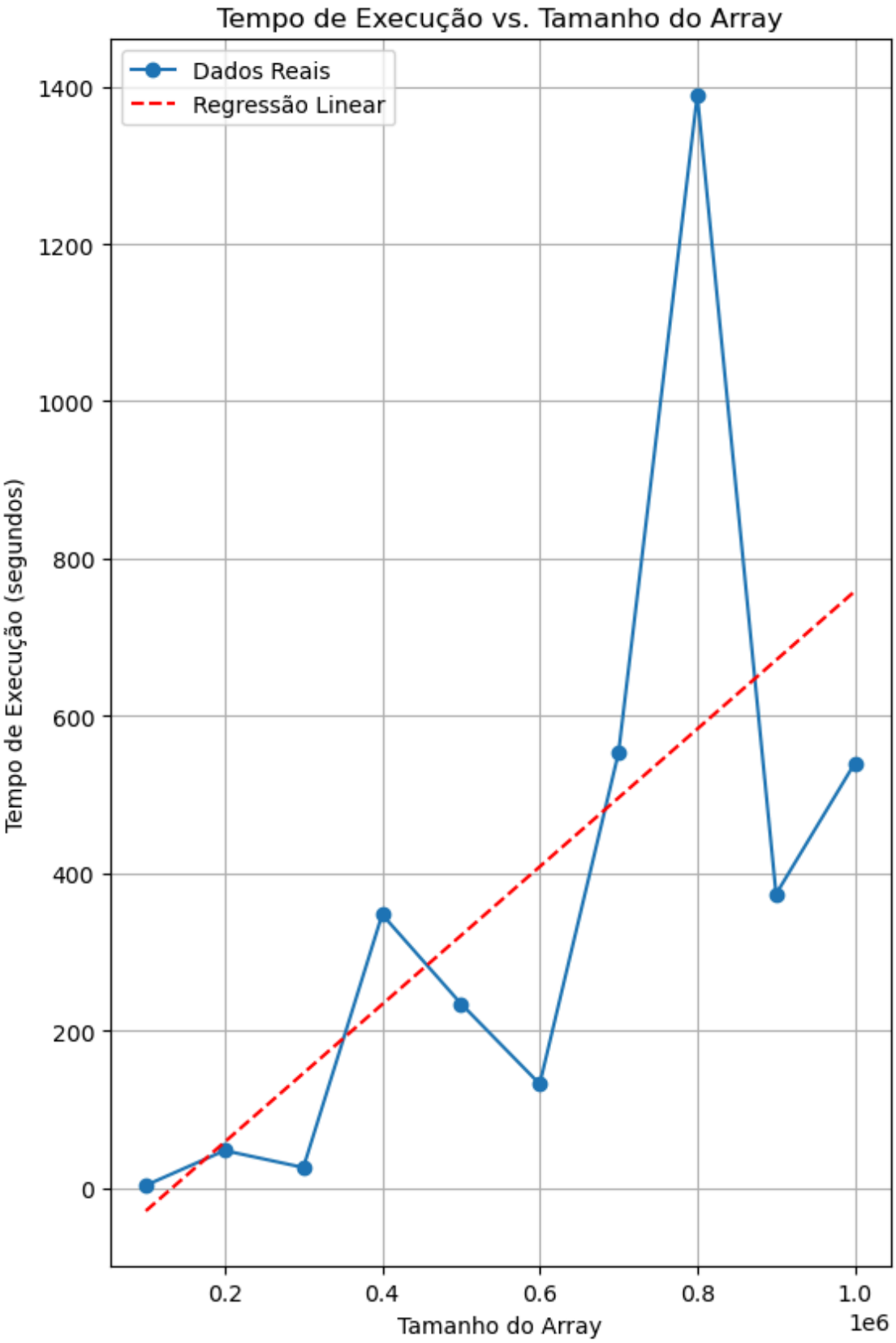
Nome: Teodoro Marques Nº Estudante: 2023211717
PL (inscrição): PL3

Registrar os tempos computacionais das 3 soluções. Os tamanhos dos arrays (N) devem ser: 100000, 200000, …, 1000000. Só deve ser contabilizado o tempo do algoritmo. Exclui-se o tempo de leitura do input e de impressão dos resultados. Devem apresentar e discutir as regressões para as 3 soluções, incluindo também o coeficiente de determinação/regressão (r quadrado).

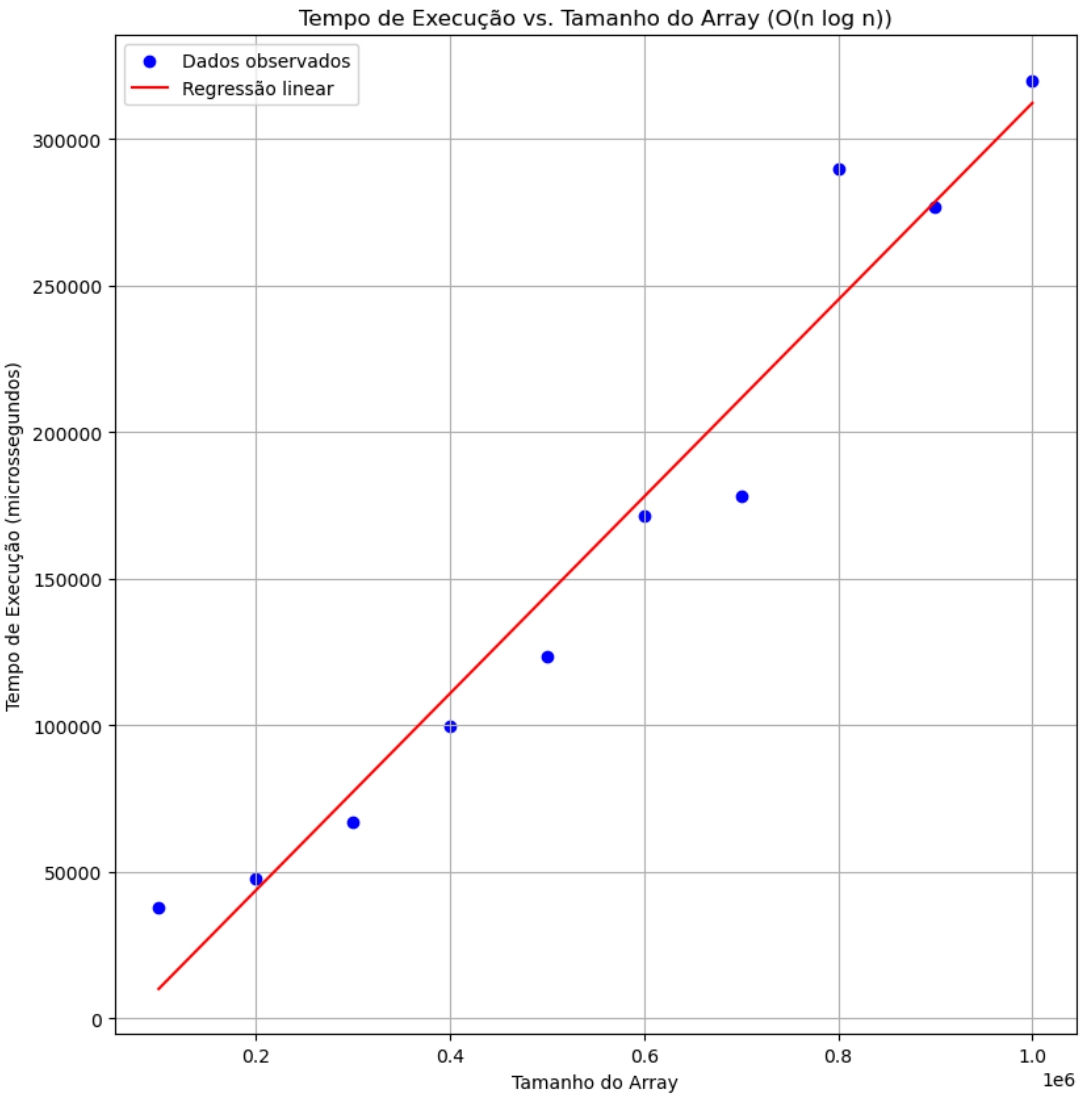
Tabela para as 3 soluções

	time (seconds)	time (microsseconds)	time (microsseconds)
N (nº de elementos)	First Approach	Second Approach	Third Approach
100000	326791	37760	1290
200000	801759	47804	2625
300000	1061224	66926	5156
400000	1408811	99611	5466
500000	1432232	123438	6962
600000	1564274	171272	11165
700000	2117388	178316	13286
800000	2256285	289992	12794
900000	2629781	276814	12637
1000000	3169829	319932	16301

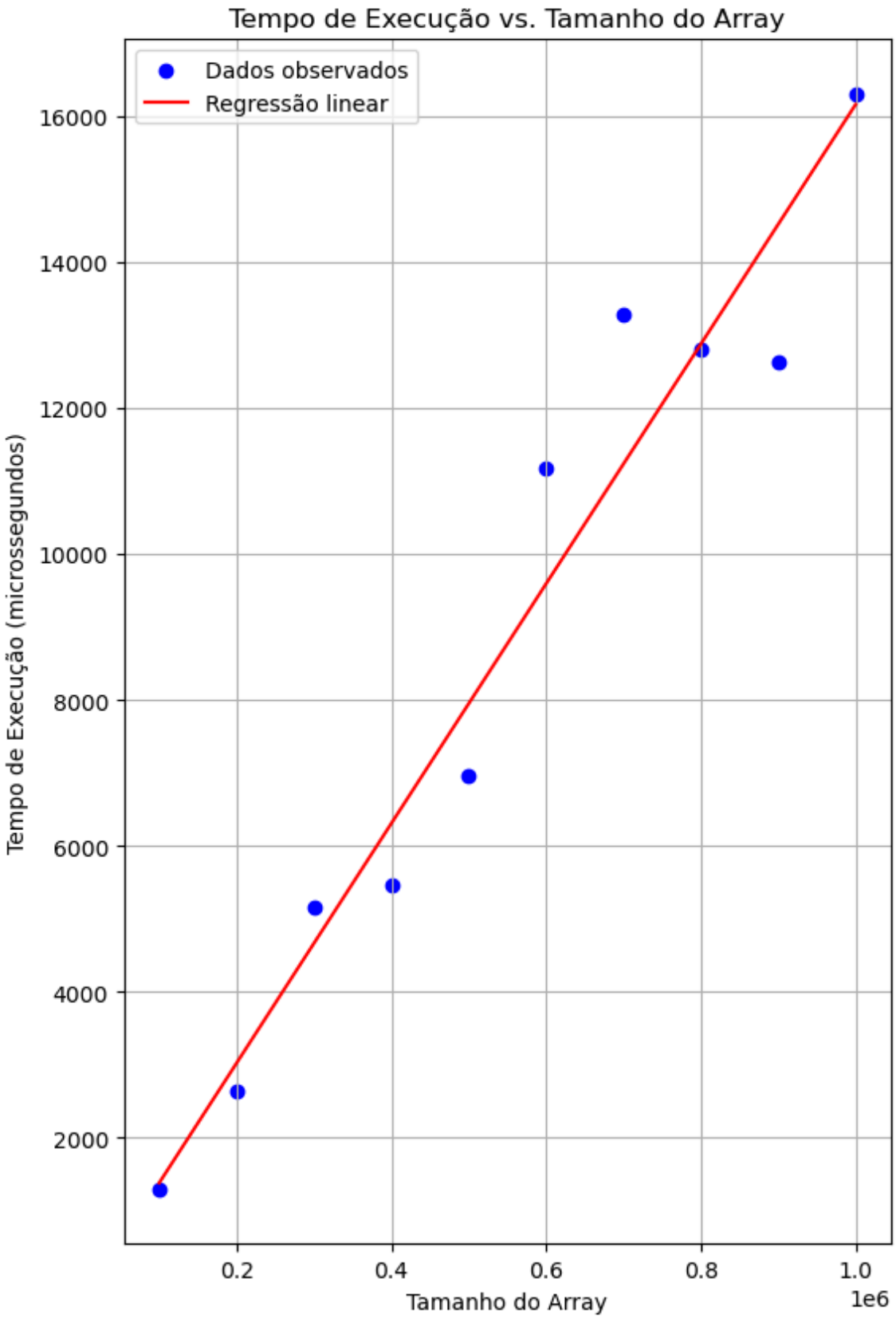
Algoritmo 1: Solução exaustiva


Tempo de Execução vs. Tamanho do Array
Dados Reais
Regressão Linear
Tempo de Execução (segundos)
0
200
400
600
800
1000
1200
1400
0.2
0.4
0.6
0.8
1.0
1e6
Tamanho do Array

Algoritmo 2: Solução c/ ordenamento



Algoritmo 3: Solução elaborada


Tempo de Execução vs. Tamanho do Array
Dados observados
Regressão linear
Tempo de Execução (microssegundos)
16000
14000
12000
10000
8000
6000
4000
2000
0.2
0.4
0.6
0.8
1.0
1e6
Tamanho do Array

1 -)

A complexidade teórica do algoritmo apresentado é $O(N^2)$, pois há dois ciclos *for* um dentro do outro. O primeiro ciclo percorre todos os N elementos do array e, para cada elemento, o segundo ciclo percorre novamente os N elementos para procurar o próximo número consecutivo na sequência. Como a verificação é feita para cada par de elementos, a complexidade é N^2.

A regressão quadrática, com $R^2 = 0.9999$, indica que a complexidade empírica está de acordo com a teórica, evidenciando que o tempo de execução cresce de forma proporcional ao quadrado do tamanho do array.

O algoritmo tem um desempenho sub óptimo para arrays grandes devido ao seu tempo de execução $O(N^2)$.

2 -)

A complexidade teórica do segundo algoritmo é O(NlogN), pois o algoritmo de ordenação utilizado, o `std::sort`, possui complexidade O(NlogN). Após a ordenação, a busca pelo número faltante é realizada com um ciclo `for`, que tem complexidade O(N) no pior caso, pois percorre os elementos do vetor para encontrar a diferença entre dois elementos consecutivos. Assim, a complexidade global do algoritmo é dominada pela ordenação, resultando em O(NlogN).

A regressão NlogN, com $R^2 = 0.9968$, também indica que os resultados empíricos estão alinhados com os resultados teóricos, evidenciando que o tempo de execução do algoritmo cresce de forma logarítmica em relação ao número de elementos.

3 -)

A complexidade teórica do algoritmo é O(N), pois percorre todos os N elementos do array uma única vez para calcular a soma total. A operação de soma é realizada em tempo constante O(1) para cada elemento, resultando em uma complexidade linear.

A regressão linear com R^2: 0.9474673320843641, indica que a complexidade teórica é refletida empiricamente, demonstrando uma forte correlação entre o tamanho do array e o tempo de execução observado.

```
/*Algorithm 1 -> time complexity O(N²)*/
#include <iostream>
#include <vector>
#include <fstream>
#include <chrono>
#include <algorithm>
int find_missing(const std::vector<int>& arr) {
	auto start_time = std::chrono::high_resolution_clock::now();
	int length = arr.size();
	int minEl = *std::min_element(arr.begin(), arr.end());
	int maxEl = *std::max_element(arr.begin(), arr.end());
	bool flag = false;
	int missing = -1;
	for (int i = 0; i < length - 1; ++i) {
	flag = false;
	for (int j = 0; j < length - 1; ++j) {
	if (arr[j] == arr[i] + 1 && arr[i] < maxEl) {
		flag = true;
		break;
	}
	}
	if (!flag && arr[i] < maxEl) {
	missing = arr[i] + 1;
	break;
	}
	}
	auto end_time = std::chrono::high_resolution_clock::now();
	std::chrono::duration<double> duration = end_time - start_time;
	std::cout << "Elemento faltante: " << missing << std::endl;
	std::cout << "Tempo de execução: " << duration.count() << "
segundos" << std::endl;
```

```
        return missing;
}
int main() {
        // Leitura do array a partir do arquivo
        std::vector<int> arr;
        std::ifstream file("array.txt");
        if (!file.is_open()) {
        std::cerr << "Erro ao abrir o arquivo array.txt" << std::endl;
        return 1;
        }
        int num;
        while (file >> num) {
        arr.push_back(num);
        }
        file.close();
        find_missing(arr);
        return 0;
}

/*Algorithm 2 -> time complexity O(N*log(N))*/
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <chrono>
int main(int argc, char* argv[]) {
        std::vector<int> sequence;
        std::ifstream file("array.txt");
        if (file.fail() || !file.is_open()) {
        std::cerr << "Could not open file" << std::endl;
        return 1;
        }
        int number;
        while (file >> number) {
        sequence.push_back(number);
```

```
        }
        file.close();
        auto start = std::chrono::high_resolution_clock::now();
        std::sort(sequence.begin(), sequence.end());
        for (size_t i = 0; i < sequence.size() - 1; ++i) {
        if (sequence[i + 1] - sequence[i] > 1) {
        std::cout << "O número faltante é: " << sequence[i] + 1 << std::endl;
        break; // Assume apenas um número faltante
        }
        }
        auto end = std::chrono::high_resolution_clock::now();
        auto duration =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);
        std::cout << "Tempo de execução: " << duration.count() << "
microssegundos" << std::endl;
        return 0;
}

/* Algorithm 3 -> time complexity O(N) */
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
int main() {
        std::vector<int> numbers;
        std::ifstream file("array.txt");
        int num;
        if (!file.is_open()) {
        std::cerr << "Erro ao abrir o arquivo." << std::endl;
        return 1;
        }
```

```
        while (file >> num) {
        numbers.push_back(num);
        }
        file.close();
        auto start = std::chrono::high_resolution_clock::now();
        long long sum = 0;
        for (int n : numbers) {
        sum += n;
        }
        size_t S = numbers.size();
        long long expected_sum = (static_cast<long long>(S) * (S + 1)) / 2;
        long long missing_number = expected_sum - sum;
        auto end = std::chrono::high_resolution_clock::now();
        auto duration =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);
        std::cout << "O número faltante é: " << missing_number << std::endl;
        std::cout << "Tempo de execução: " << duration.count() << "
microssegundos" << std::endl;
        return 0;
}
```