Universidade Federal do Rio Grande do Norte – UFRN
Instituto Metrópole Digital

Estrutura de Dados Básicas I – IMD0029

JOÃO EMMANUEL IZIDIO DA SILVA

RELATÓRIO TÉCNICO FINAL

ANÁLISE EMPÍRICA DE ALGORITMOS

Natal

17 de março de 2016

SUMÁRIO

1 APRESENTAÇÃO DO PROJETO.....3

2 MATERIAL E METODOLOGIAS USADAS.....3

3 RESULTADOS ALCANÇADOS.....7

4 CONCLUSÃO.....13

5 REFERÊNCIAS.....14

1 - APRESENTAÇÃO DO PROJETO

Este relatório tem como objetivo mostrar os resultados obtidos a partir da análise empírica aplicada em oito algoritmos (Busca Sequencial Iterativa, Busca Sequencial Recursiva, Busca Sequencial Padrão, Busca Binária Iterativa, Busca Binária Recursiva, Busca Binária Padrão, Busca Ternária Iterativa, Busca Ternária Recursiva), levando em consideração três cenários: Em que o elemento procurado não está no vetor, em que o elemento procurado está localizado no ¼ final do tamanho do vetor e onde devemos procurar a terceira ocorrência do elemento procurado, se existir. Os testes foram feitos em vetores com elementos crescentes para todos os algoritmos e em vetores não ordenados para os sequenciais.

2 - MATERIAL E METODOLOGIAS USADAS

O computador usado foi um Lenovo G40-80, que possui Intel Core i3 5005U 2.2 GHz até 2.7 GHz e 3MB de cache, 4GB de RAM do tipo DDR3 1600MHz, Intel HD Graphics 5500, Linux Ubuntu 14.04.4 LTS. A linguagem usada nos teste foi C++ e compilada com o gcc . Segue abaixo a lista de algoritmos implementados:

```
long int b_Sequencial_I (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
	int cond=0;
	for (int i = ini; i <= tam; i++) {
		if (vetor[i] == chave) {
			cond++;
			if (cond > ocor) {
				return i;
			}
		}
	}
	return -1;
}

long int b_Sequencial_R (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
	if (ini == tam) {
		return -1;
	}

	if (vetor[ini] == chave) {
		if (ocor == 0) {
			return ini;
		}
		return b_Sequencial_R(chave, vetor, ini+1, tam, ocor-1);
	} else {
		return b_Sequencial_R(chave, vetor, ini+1, tam, ocor);
	}
}

long int b_Binaria_R (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
	long int meio;
	meio = (ini+tam)/2;
	if (ini > tam) {
		return -1;
	}
```

```

	if (chave == vetor[meio]) {
		if (meio == 0) {
			if (vetor[meio+ocor] == chave) {
				return meio+ocor;
			} else {
				return -1;
			}
		}
		if (vetor[meio-1] == chave) {
			return b_Binaria_R(chave, vetor, ini, meio, ocor);
		}
		if (vetor[meio+ocor] == chave) {
			return meio+ocor;
		} else {
			return -1;
		}
	} else if (chave > vetor[meio]){
		return b_Binaria_R(chave, vetor, meio+1, tam, ocor);
	} else {
		return b_Binaria_R(chave, vetor, ini, meio-1, ocor);
	}
}

long int b_Binaria_I (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
	long int i=0, f=tam, meio, inicio;
	while (i <= f) {
		meio = (i+f)/2;

		if (chave == vetor[meio]) {
			for (int j = meio; j >= 0; j--) {
				if (chave == vetor[j]){
					inicio = j;
				}
			}
```

```
                        if (chave == vetor[inicio+ocor]) {
                                return inicio+ocor;
                        } else {
                                return -1;
                        }

                } else if (chave > vetor[meio]) {
                        i = meio + 1;
                } else {
                        f = meio - 1;
                }
        }
        return -1;
}

long int b_Ternaria_I (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
        int long p1=0, p2=0, i=0, f=tam, var, inicio;

        while (i <= f) {
                var = (f-i)/3;
                p1 = var + i;
                p2 = p1 + var;
                if (chave == vetor[p1]) {
                        for (int j = p1; j >= 0; j--) {
                                if (chave == vetor[j]){
                                        inicio = j;
                                }
                        }
                        if (chave == vetor[inicio+ocor]) {
                                return inicio+ocor;
                        } else {
                                return -1;
                        }
                } else if (chave == vetor[p2]) {
                        for (int j = p2; j >= 0; j--) {
```

```
                        for (int j = p2; j >= 0; j--) {
                                if (chave == vetor[j]){
                                        inicio = j;
                                }
                        }
                        if (chave == vetor[inicio+ocor]) {
                                return inicio+ocor;
                        } else {
                                return -1;
                        }
                } else if (chave < vetor[p1]) {
                        f = p1 - 1;
                } else if (chave > vetor[p2]) {
                        i = p2 + 1;
                } else {
                        i = p1 + 1;
                        f = p2 - 1;
                }
        }

        return -1;
}

long int b_Ternaria_R (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
        long int p1, p2;
        p1 = ((tam-ini)/3) + ini;
        p2 = ((tam-ini)/3) + p1;
        if (ini > tam) {
                return -1;
        }

        if (chave == vetor[p1]) {
                if (p1 == 0) {
                        if (chave == vetor[p1+ocor]) {
                                return p1+ocor;
```

```
                        if (chave == vetor[p1+ocor]) {
                                return p1+ocor;
                        } else {
                                return -1;
                        }
                }
                if (chave == vetor[p1-1]) {
                        return b_Ternaria_R(chave, vetor, ini, p1, ocor);
                }
                if (chave == vetor[p1+ocor]) {
                        return p1+ocor;
                } else {
                        return -1;
                }
        } else if (chave == vetor[p2]) {
                if (p2 == 0) {
                        if (chave == vetor[p1+ocor]) {
                                return p1+ocor;
                        } else {
                                return -1;
                        }
                }
                if (chave == vetor[p2-1]) {
                        return b_Ternaria_R(chave, vetor, ini, p2, ocor);
                }
                if (chave == vetor[p2+ocor]) {
                        return p2+ocor;
                } else {
                        return -1;
                }
        } else if (chave < vetor[p1]) {
                return b_Ternaria_R(chave, vetor, ini, p1-1, ocor);
        } else if (chave > vetor[p2]) {
                return b_Ternaria_R(chave, vetor, p2+1, tam, ocor);
        } else {
```

```
                return b_Ternaria_R(chave, vetor, p2+1, tam, ocor);
        } else {
                return b_Ternaria_R(chave, vetor, p1+1, p2-1, ocor);
        }
}

long int b_Sequencial_P (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
        std::vector<long int> c;
        for (int i = 0; i <= ocor; i++) {
                c.push_back(chave);
        }

        long int *it = (long int *) search(vetor, vetor+tam+1, c.begin(), c.end());

        if ( it != (vetor+tam+1) ) {
                return (it - vetor+ocor);
        } else {
                return -1;
        }


}

int comp(const void *ap, const void *bp) {
    const int *a = (int *) ap;
    const int *b = (int *) bp;
    return *a - *b;
}

long int b_Binaria_P (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
        long int *r = (long int*) bsearch(&chave, vetor, tam+1, sizeof(vetor[0]), comp);
        long int *aux = r;
        long int valor;
        if (r == NULL) {
                return -1;
        } else {
```

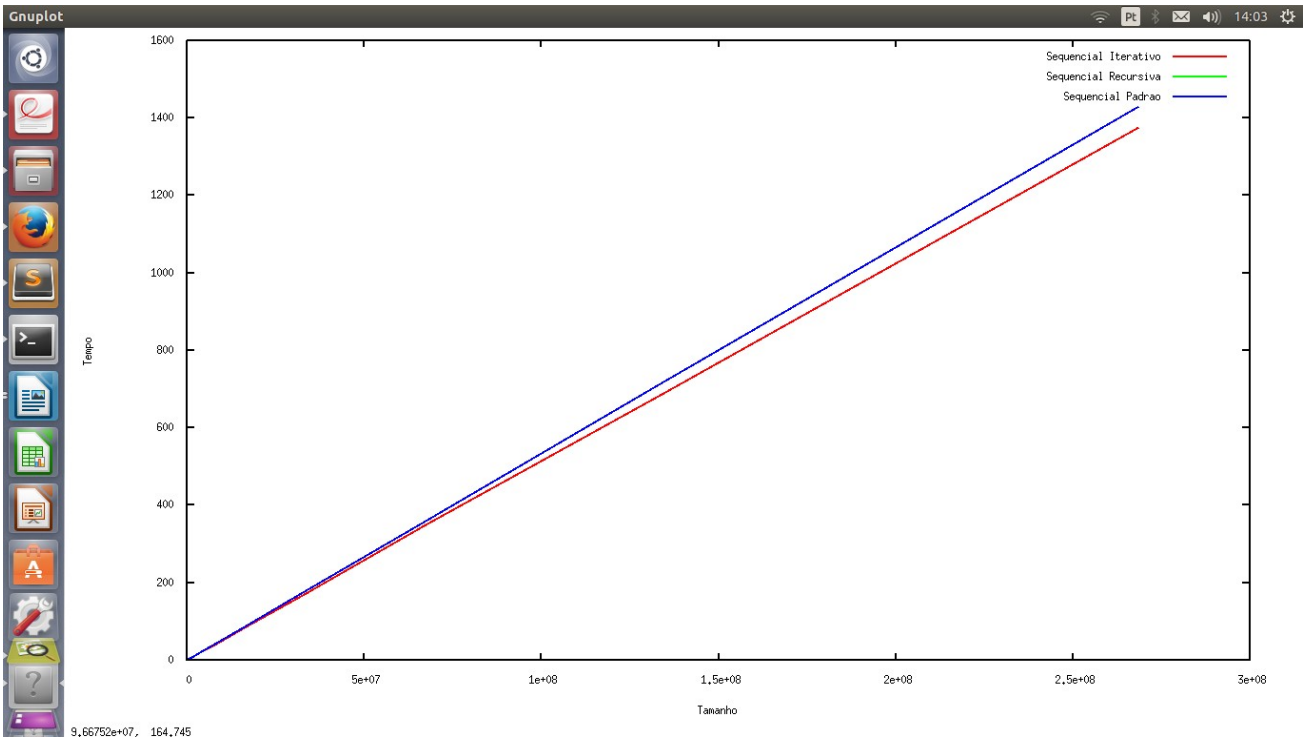
```

}

int comp(const void *ap, const void *bp) {
    const int *a = (int *) ap;
    const int *b = (int *) bp;
    return *a - *b;
}

long int b_Binaria_P (long int chave, long int* vetor, long int ini, long int tam, int ocor) {
        long int *r = (long int*) bsearch(&chave, vetor, tam+1, sizeof(vetor[0]), comp);
        long int *aux = r;
        long int valor;
        if (r == NULL) {
                return -1;
        } else {
                while (aux != NULL) {
                        valor = aux - vetor;
                        aux = (long int*) bsearch(&chave, vetor, valor, sizeof(vetor[0]), comp);
                }
                if (vetor[valor+ocor] == chave) {
                        return valor+ocor;
                }else {
                        return -1;
                }
        }
}
```

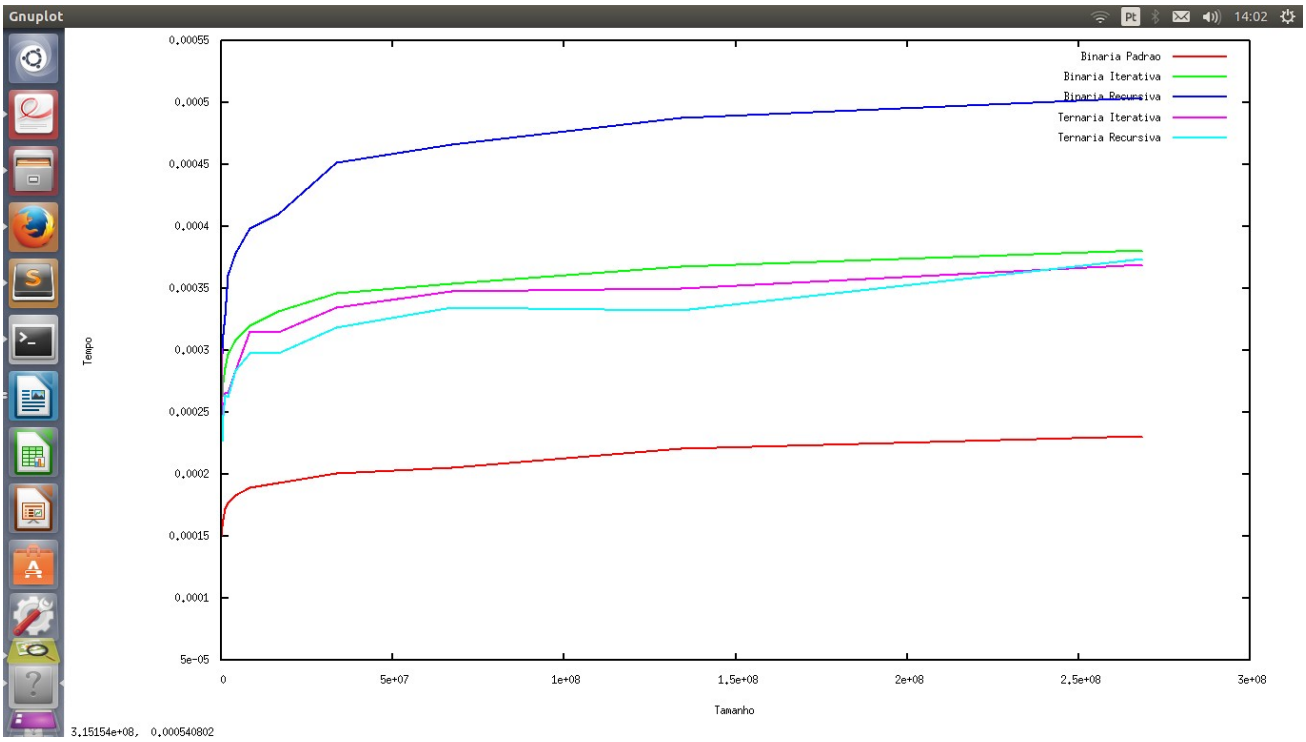
São feitos 100 testes para cada tamanho do vetor, que vai de 2^4 até 2^28. O tempo de execução de cada algoritmo é obtido pela média progressiva dos testes e usa como unidade de medida o Milissegundo (ms).

3 - RESULTADOS ALCANÇADOS

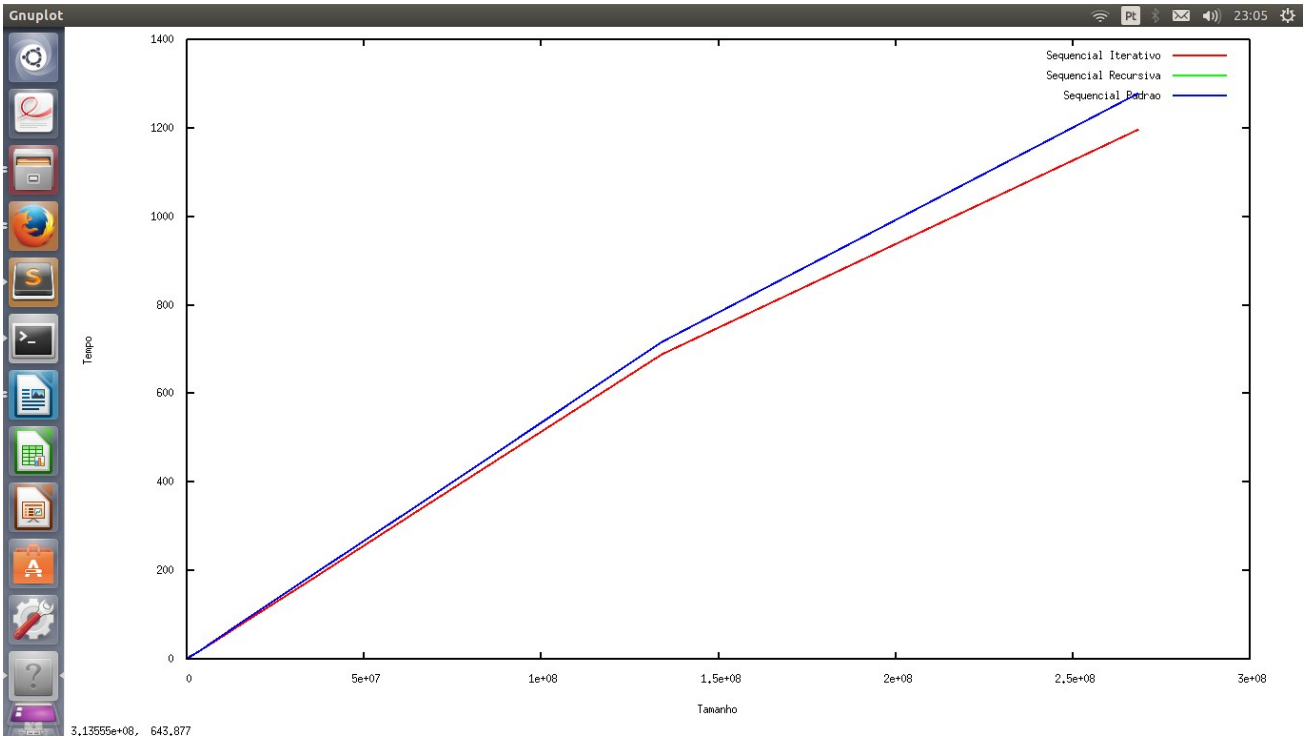
VETOR ORDENADO:

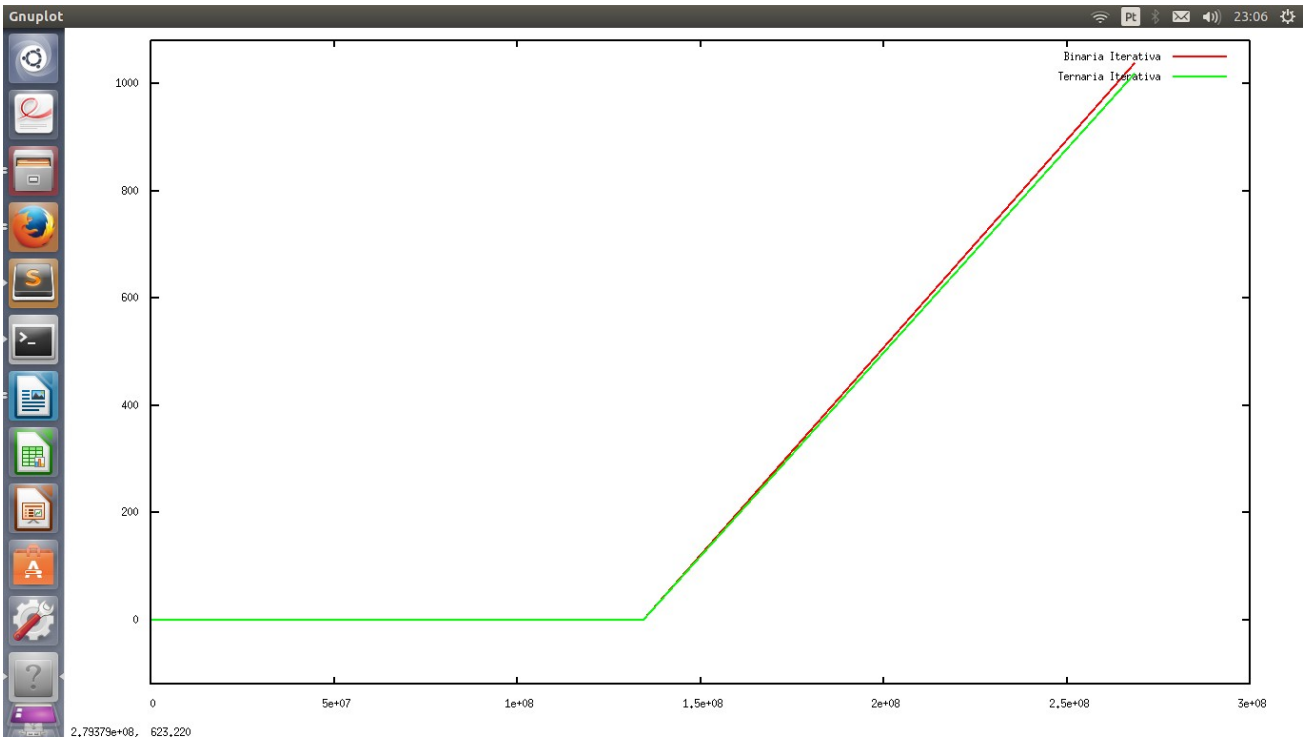
- Busca por um valor que não está no vetor:

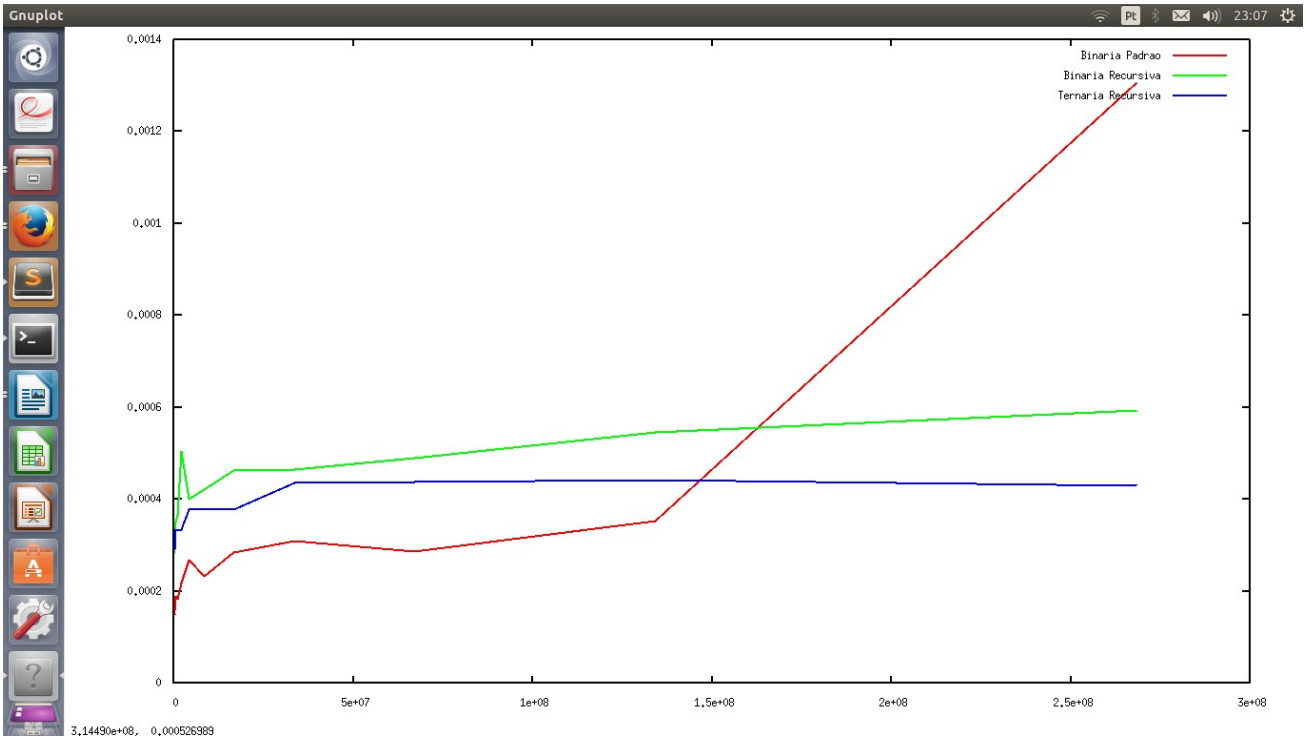




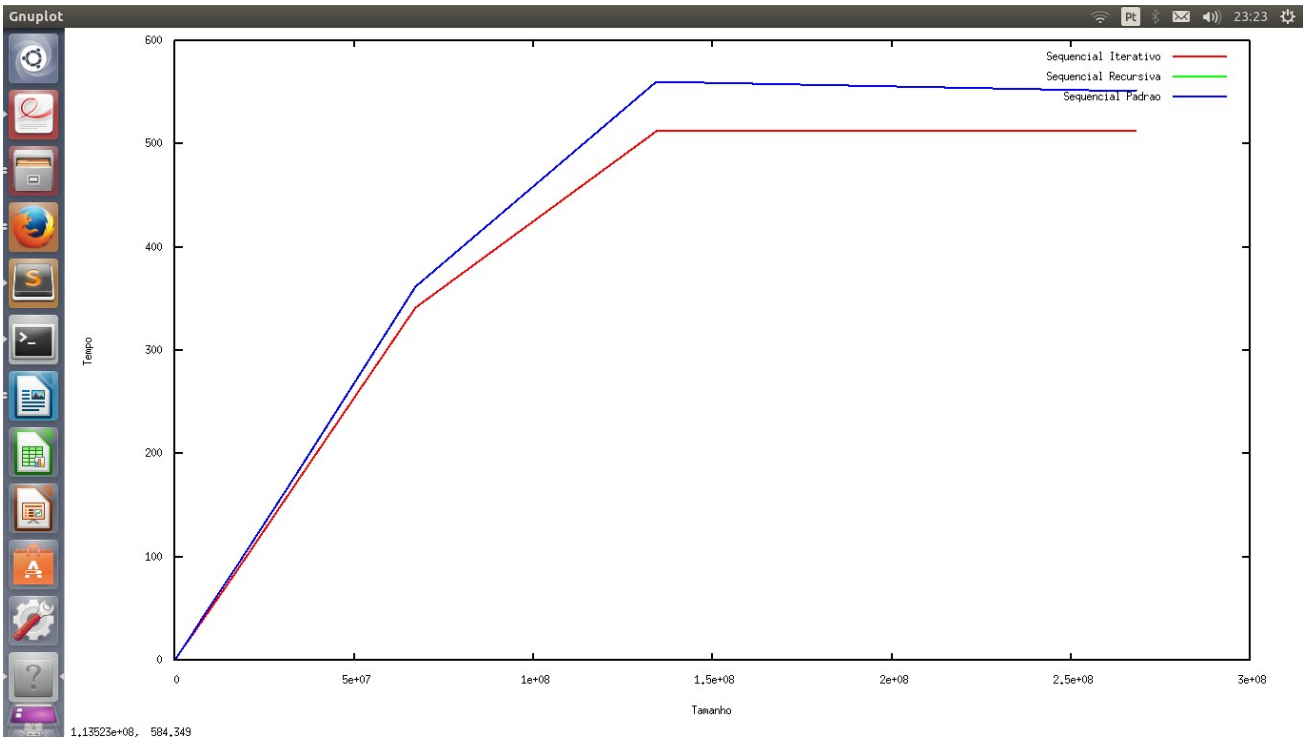
- Busca por um valor que está no ultimo ¼ do vetor:

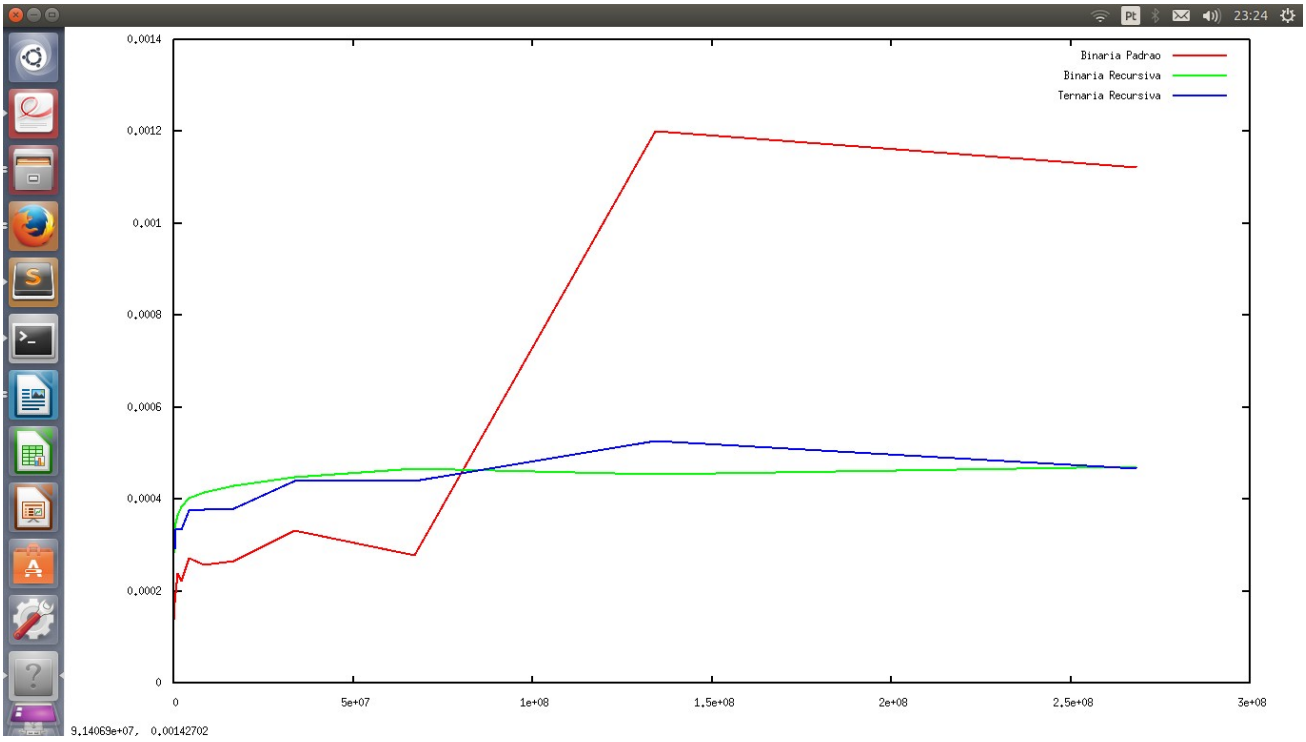


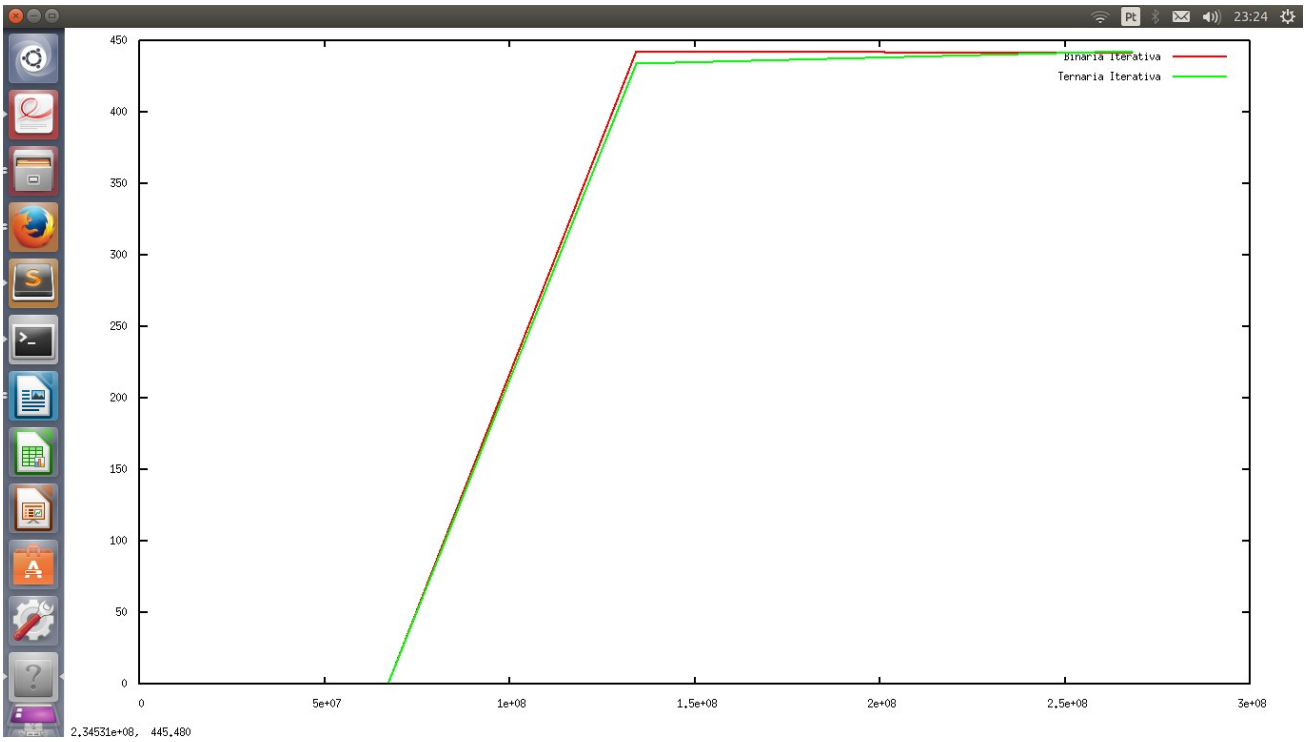




- Busca pela 3 ocorrência de um valor:

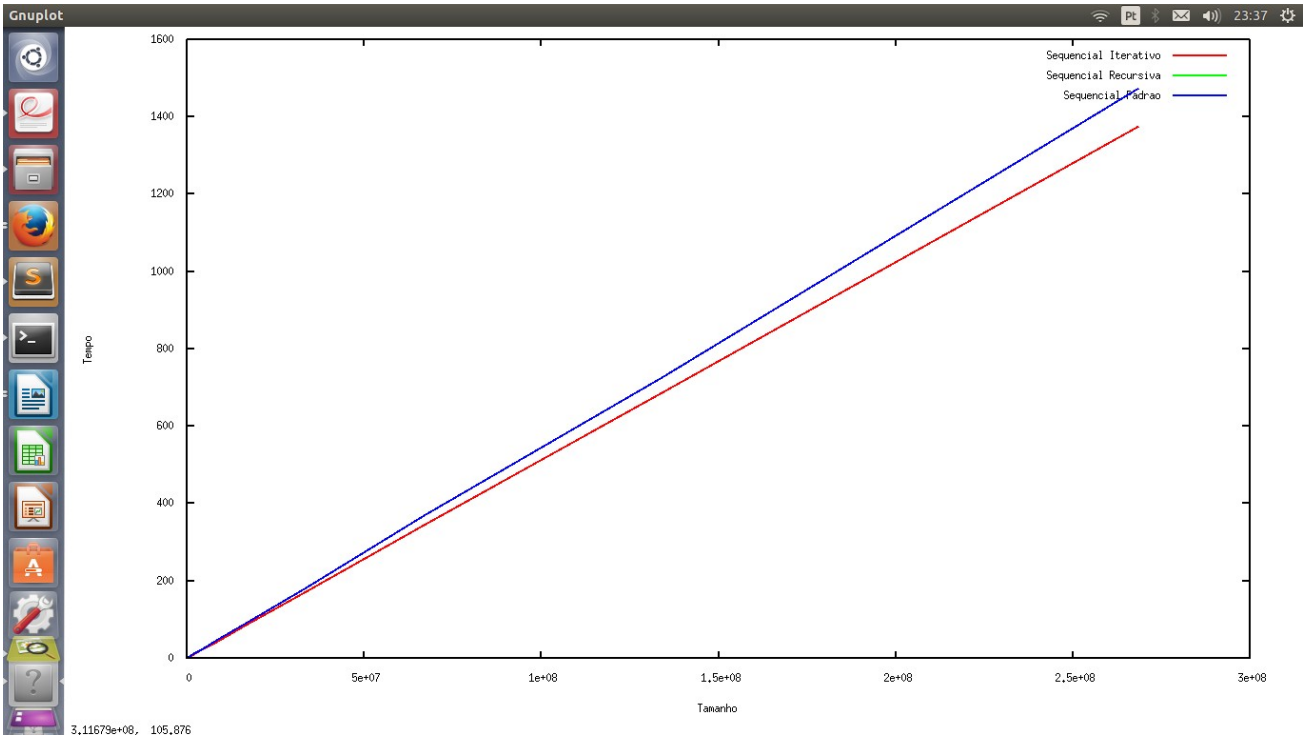

Gnuplot
23:23
Sequencial Iterativo
Sequencial Recursiva
Sequencial Padrao
Tempo
Tamanho
600
500
400
300
200
100
0
0
5e+07
1e+08
1.5e+08
2e+08
2.5e+08
3e+08
1.13523e+08, 584.349


23:24
Binaria Padrao
Binaria Recursiva
Ternaria Recursiva
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
5e+07
1e+08
1.5e+08
2e+08
2.5e+08
3e+08
9.14069e+07, 0.00142702

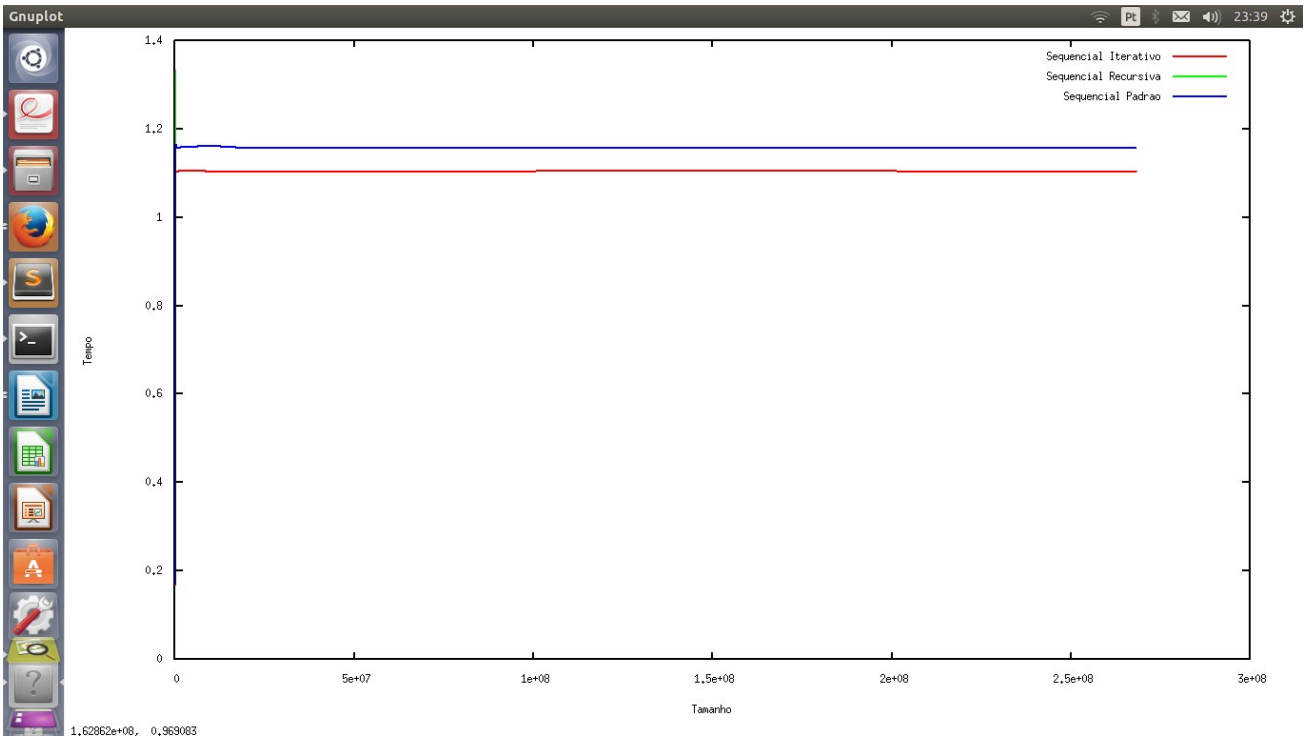


VETOR NÃO ORDENADO:

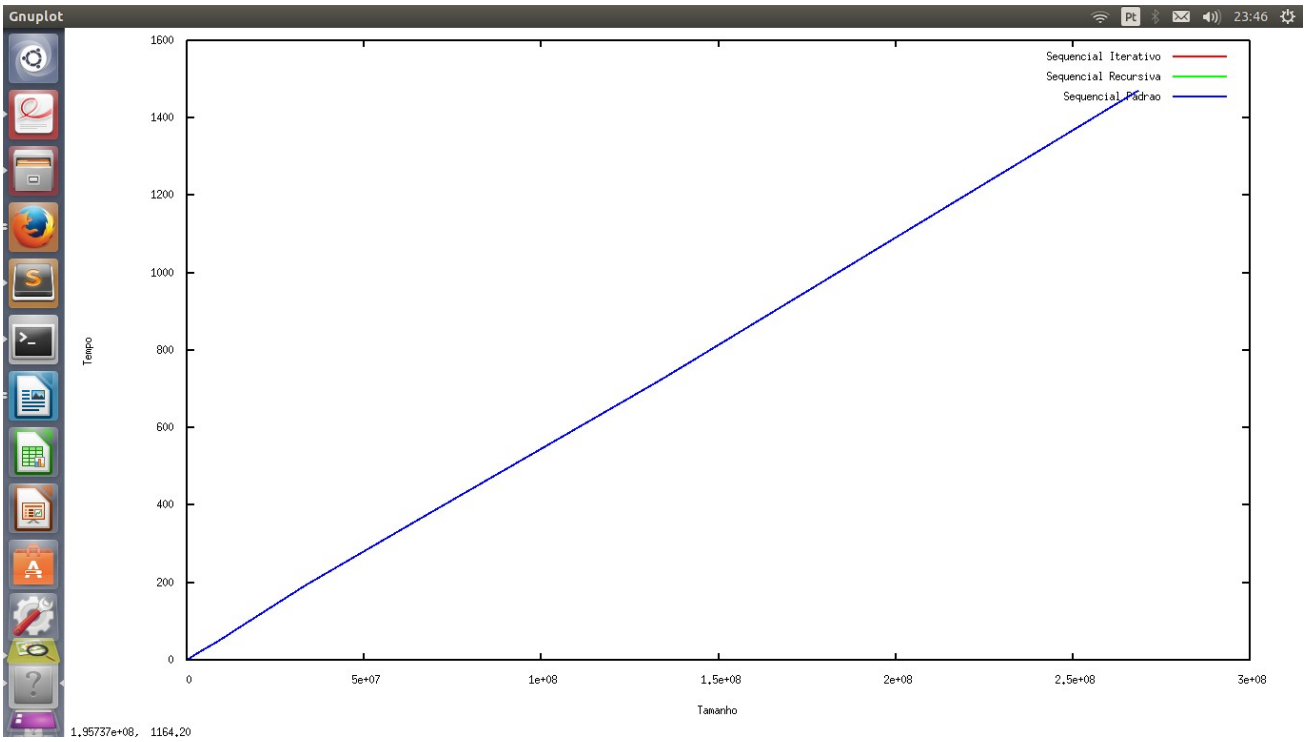
- Busca por um valor que não está no vetor:



- Busca por um valor que está no ultimo ¼ do vetor:



- Busca pela 3 ocorrência de um valor:



4 - CONCLUSÃO

(a) Descobrimos de maneira geral que as funções sequenciais demoram muito mais que as outras, a menos que o elemento que procuramos esteja bem no inicio do vetor.

(b) Os algoritmos recomendados para cada situação:

- Vetor ordenado e busca por um valor que não está no vetor: Busca Binária Padrão

- Vetor ordenado e busca por um valor que está no ultimo ¼ do vetor: Ternária Recursiva (Para valores grandes)

- Vetor ordenado e busca pela 3 ocorrência de um valor: Binária Recursiva

- Vetor não-ordenado e busca por um valor que não está no vetor: Binária Recursiva

- Vetor não-ordenado e busca por um valor que está no ultimo ¼ do vetor: Binária Recursiva (Para valores grandes)

- Vetor não-ordenado e busca pela 3 ocorrência de um valor: Binária Recursiva

(c) Sim, em quase todos os casos. Se continuarmos dividindo em partes chegaríamos a n vetores com um elemento, caracterizando assim uma busca sequencial.

(d) Aconteceram alguns picos de valores nas funções nas buscas Iterativas Binaria e Ternaria, no caso dois e três dos vetores ordenados.

(e) A função linear é a que mais se aproxima da maioria, Estima-se que o tempo de cada algoritmo fazendo a busca com 100 milhões de elementos ficam com: Sequencial Iterativa – 512.288, Sequencial Recursiva – Não chega até esse tamanho, Sequencial Padrão – 533.24, Binário Padrão – 0.00020627, Binário Iterativa – 0.00034979, Binário Recursiva – 0.000472, Ternário Iterativa – 0.000349, Ternaria Recursiva – 0.000338.

(f) A analise empírica é mais precisa que a matemática, o que é bom para a escolha certa do algoritmo para cada caso.

5 - REFERÊNCIAS

- http://pt.cppreference.com
- http://www.cplusplus.com/reference
- http://www.dicas-l.com.br/arquivo/usando_gnuplot_para_gerar_bons_graficos.php