PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e Informática
Exercícios sobre Contagem de Operações e Medição do Tempo de Execução de Algoritmos
Pesquisa Sequencial x Pesquisa Binária
Curso: Engenharia de Software Disciplina: Algoritmos e Estruturas de Dados II
Professores: Eveline Alonso Veloso e João Caram
Aluno: Arthur Curi Kramberger(729488)

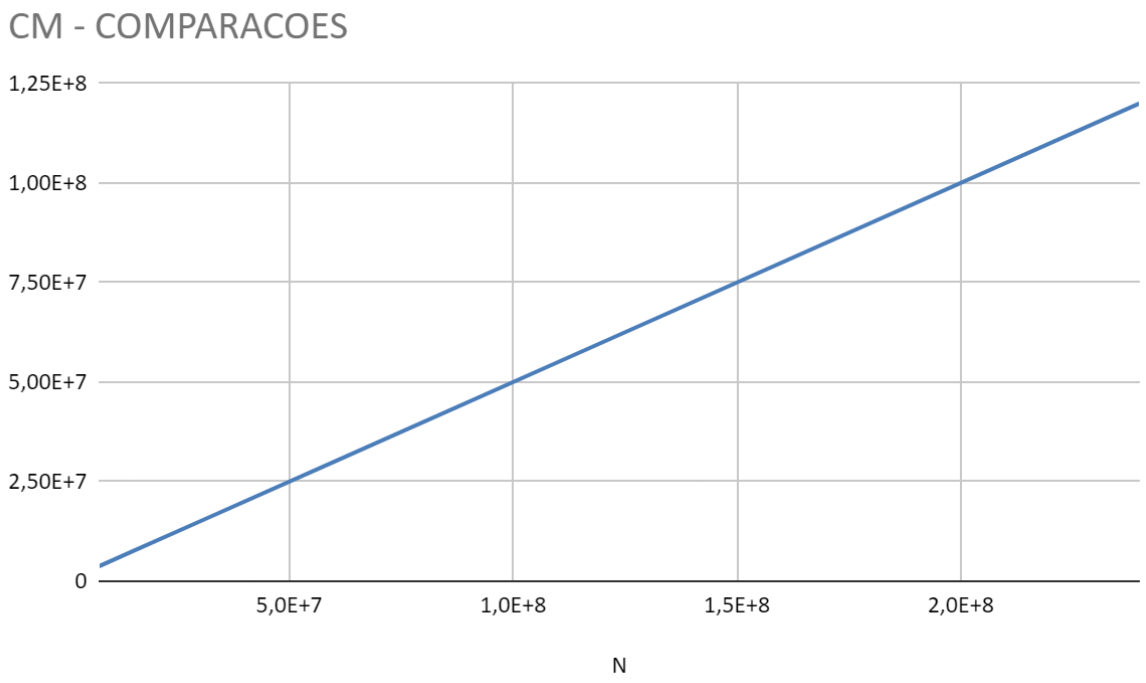
Pesquisa Sequencial:

```java
import java.util.Arrays;

public class AppSequencial {
    public static void main(String[] args) throws Exception {
        int n = 7_500_000;

        for (int i = n; i <= 2_000_000_000; i *= 2) {
            int[] vect = new int[i];

            for (int j = 0; j < i; j++) {
                vect[j] = j + 1;
            }

            medicoes(vect, vect[0], "Melhor Caso");

            medicoes(vect, vect[i / 2], "Caso Médio");

            medicoes(vect, -1, "Pior Caso");

            System.out.println();
        }
    }

    public static void medicoes(int[] vect, int valorPesquisa, String
descricao) {
        long[] tempos = new long[5];
        long[] operacoes = new long[5];

        for (int j = 0; j < 5; j++) {
            long inicio = System.nanoTime();
            int numOp = pesquisaSequencial(valorPesquisa, vect);
            long fim = System.nanoTime();

            tempos[j] = fim - inicio;
            operacoes[j] = numOp;
```

```
        }

        Arrays.sort(tempos);
        long tempoMedio = (tempos[1] + tempos[2] + tempos[3]) / 3;

        System.out.println(descricao + " | N: " + vect.length + ",
Operações: " + operacoes[0] + ", Tempo médio (ns): " + tempoMedio);
    }

    public static int pesquisaSequencial(int valor, int[] vect) {
        int cc = 0;
        for (int i = 0; i < vect.length; i++) {
            cc++;
            if (vect[i] == valor) {
                return cc;
            }
        }
        return cc;
    }
}
```

LEGENDA:
MC = Melhor Caso
CM = Caso Médio
PC = Pior Caso

N	MC - Operações	MC - Tempo médio (ns)	CM - Operações	CM - Tempo médio (ns)	PC - Operações	PC - Tempo médio (ns)
7500000	1	266	3750001	2586866	7500000	3081700
15000000	1	100	7500001	3067600	15000000	4676633
30000000	1	133	15000001	5528133	30000000	12419500
60000000	1	100	30000001	17989933	60000000	23339466
120000000	1	100	60000001	22950333	120000000	55275566
240000000	1	66	120000001	45926466	240000000	128314966



A pesquisa sequencial é simples, mas fica mais lenta à medida que o vetor cresce, já que precisa olhar cada elemento até achar o que procura ou chegar ao fim.

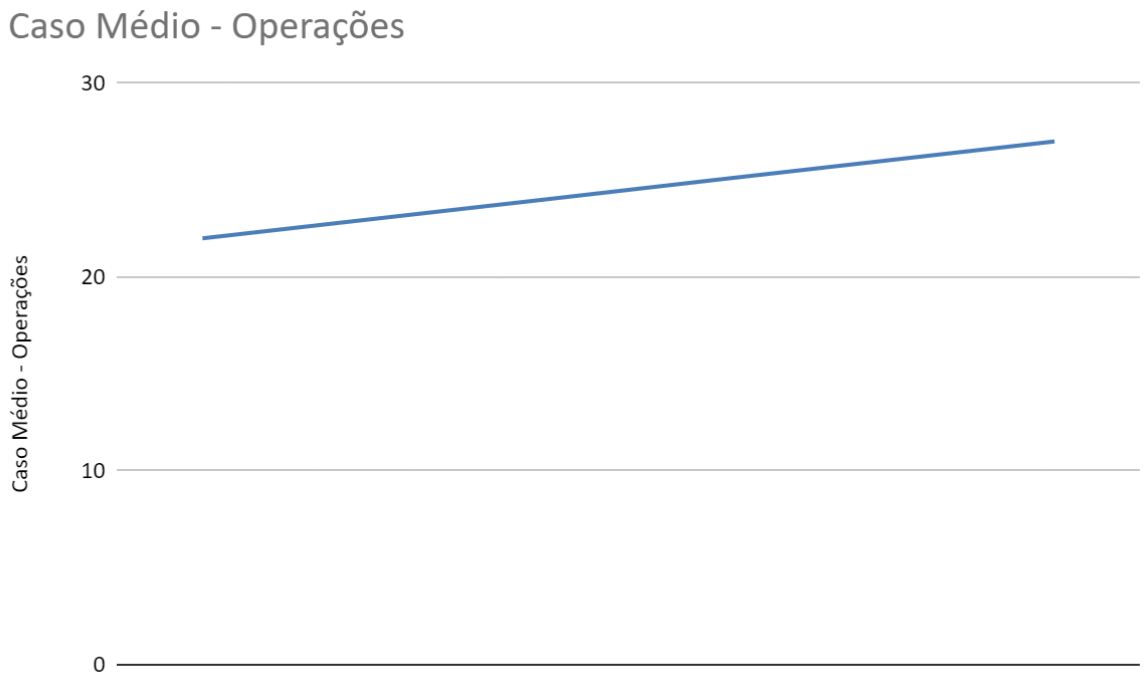
Pesquisa Binária:

```java
import java.util.Arrays;

public class AppBinario {
    public static void main(String[] args) throws Exception {
        int n = 7_500_000;

        for (int i = n; i <= 2_000_000_000; i *= 2) {
            int[] vect = new int[i];

            for (int j = 0; j < i; j++) {
                vect[j] = j + 1;
            }

            medicoes(vect, vect[0], "Melhor Caso");

            medicoes(vect, vect[i / 2], "Caso Médio");

            medicoes(vect, -1, "Pior Caso");

            System.out.println();
```

```
        }
    }

    public static void medicoes(int[] vect, int valorPesquisa, String
descricao) {
        long[] tempos = new long[5];
        long[] operacoes = new long[5];

        for (int j = 0; j < 5; j++) {
            long inicio = System.nanoTime();
            int numOp = pesquisaBinaria(valorPesquisa, vect, 0,
vect.length - 1, 0);
            long fim = System.nanoTime();

            tempos[j] = fim - inicio;
            operacoes[j] = numOp;
        }

        Arrays.sort(tempos);
        long tempoMedio = (tempos[1] + tempos[2] + tempos[3]) / 3;

        System.out.println(descricao + " | N: " + vect.length + ",
Operações: " + operacoes[0] + ", Tempo médio (ns): " + tempoMedio);
    }

    public static int pesquisaBinaria(int valor, int[] vect, int
inicio, int fim, int cc) {
        if (inicio > fim) {
            return cc;
        }

        int meio = (inicio + fim) / 2;
        cc++;

        if (vect[meio] == valor) {
            return cc;
        } else if (vect[meio] < valor) {
            return pesquisaBinaria(valor, vect, meio + 1, fim, cc);
        } else {
            return pesquisaBinaria(valor, vect, inicio, meio - 1, cc);
        }
    }
}
```

LEGENDA:
MC = Melhor Caso
CM = Caso Médio
PC = Pior Caso

N	MC - Operações	MC - Tempo médio (ns)	CM - Operações	CM - Tempo médio (ns)	PC - Operações	PC - Tempo médio (ns)
7500000	22	2400	22	2433	22	3900
15000000	23	466	23	433	23	400
30000000	24	300	24	366	24	333
60000000	25	266	25	400	25	633
120000000	26	366	26	400	26	366
240000000	27	400	27	500	27	400



A pesquisa binária é muito mais eficiente em vetores grandes, porque reduz drasticamente o número de operações necessárias para encontrar um valor ou determinar que ele não está no vetor.