

# Algoritmos e Estruturas de Dados

## Análise de Algoritmos

**Autores:**

Carlos Urbano

Catarina Reis

José Magno

Marco Ferreira

# Análise de Algoritmos

- **Objetivos**

- Comparar diferentes algoritmos que resolvem um mesmo problema (veremos um exemplo mais à frente)
- Análise da performance de um algoritmo em diferentes ambientes (diferentes máquinas)
- Análise das variações de performance com base na alteração de parâmetros do problema (tamanho do problema ou os dados iniciais)

# Análise de Algoritmos

- **Tempo de Execução – como medir?**

- Copiar um ficheiro através da internet  
2 segundos para estabelecer a ligação e 8,5kb/s para “sacar” o ficheiro
- Escrever todos os pares de pontos que se podem obter a partir de um conjunto de  $n$  números

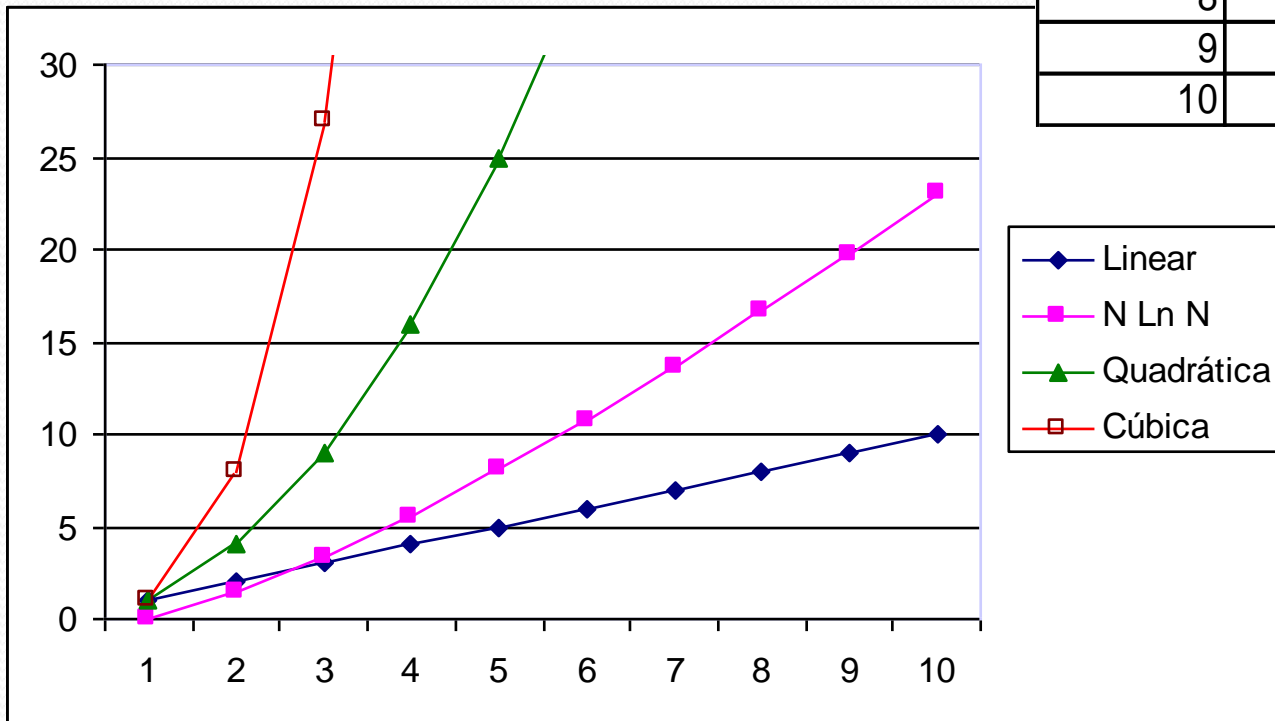
Exemplo  $n = 6$                        $\{1, 2, 3, 4, 5, 6\}$

$\{ (1,1), (1, 2), (1, 3), (1,4), (1, 5) , (1,6),$   
 $(2,1), (2, 2), (2, 3), (2,4), (2, 5) , (2,6),$   
 $(3,1), (3, 2), (3, 3), (3,4), (3, 5) , (3,6),$   
 $(4,1), (4, 2), (4, 3), (4,4), (4, 5) , (4,6),$   
 $(5,1), (5, 2), (5, 3), (5,4), (5, 5) , (5,6),$   
 $(6,1), (6, 2), (6, 3), (6,4), (6, 5) , (6,6) \}$

# Análise de Algoritmos

- Tempo de Execução

	Linear	N Ln N	Quadrática	Cúbica
1	1	0,0	1	1
2	2	1,4	4	8
3	3	3,3	9	27
4	4	5,5	16	64
5	5	8,0	25	125
6	6	10,8	36	216
7	7	13,6	49	343
8	8	16,6	64	512
9	9	19,8	81	729
10	10	23,0	100	1000



# Análise de Algoritmos

- **Tempos Médios de Execução**

Operações por segundo	Problema com tamanho $10^6$			Problema com tamanho $10^{12}$		
	Linear	N Ln N	Quadrática	Linear	N Ln N	Quadrática
$10^{24}$	$10^{-18}$	$1,4 \times 10^{-17}$	$10^{-12}$	$10^{-12}$	$1,4 \times 10^{-11}$	1
$10^{12}$	$10^{-6}$	$1,4 \times 10^{-5}$	1	1	14	$10^{12}$
$10^9$	$10^{-3}$	$1,4 \times 10^{-2}$	$10^3$	$10^3$	$1,4 \times 10^4$	$10^{15}$
$10^6$	1	14	$10^6$	$10^6$	$1,4 \times 10^7$	$10^{18}$

Operações por segundo	Problema com tamanho $10^6$			Problema com tamanho $10^{12}$		
	Linear	N Ln N	Quadrática	Linear	N Ln N	Quadrática
$10^{24}$	instantâneo	instantâneo	instantâneo	instantâneo	instantâneo	segundos
$10^{12}$	instantâneo	instantâneo	segundos	segundos	segundos	anos
$10^9$	instantâneo	instantâneo	minutos	minutos	horas	milénios
$10^6$	segundos	segundos	dias	dias	meses	milénios

# Análise de Algoritmos

- **Tempo de Execução**

- Diversas variáveis: máquina, dados iniciais, etc
- Então, como medir e comparar o tempo de execução de algoritmos?
- Utilizando uma **notação robusta e fiável**
  - Ex: **Big-Oh** (existem outras)
  - normalmente, definida através de fórmulas matemáticas
  - mas iremos simplificar a definição recorrendo a **4 regras simples e fáceis de perceber**

# Análise de Algoritmos

- **Notação Big-Oh**

- **Regra 1:** O tempo de execução de um ciclo é no máximo o tempo de execução do corpo do ciclo multiplicado pelo número de iterações do ciclo

```
for (int i = 0; i < n; i++) {  
    sum++;  
}
```

$O(N)$

- **Regra 2:** Nos ciclos encadeados, o tempo total de execução é o tempo de execução do corpo do ciclo mais interno multiplicado pelo produto dos tamanhos de todos os ciclos

```
for (int i = 0, j; i < n; i++) {  
    for (j = 0; j < n; j++) {  
        sum++;  
    }  
}
```

$O(N^2)$

# Análise de Algoritmos

- **Notação Big-Oh**

- **Regra 3:** Nas instruções consecutivas deve ser considerada a que tiver um maior tempo de execução

```
for (int i = 0; i < n; i++) {  
    a[i] = 0;  
}  
for (int i = 0, j; i < n; i++) {  
    for (j = 0; j < n; j++) {  
        sum++;  
    }  
}
```

$O(N^2)$



# Análise de Algoritmos

- **Notação Big-Oh**

- **Regra 4:** Para um bloco `if ... else` deve ser considerado o ramo com maior tempo de execução

```
if (sum == 0) {  
    for (int i = 0; i < n; i++) {  
        a[i] = 0;  
    }  
} else {  
    for (int i = 0, j; i < n; i++) {  
        for (j = 0; j < n; j++) {  
            sum++;  
        }  
    }  
}
```

$O(N^2)$

# Análise de Algoritmos

- Exercício 1: Qual a ordem de execução do seguinte bloco?

```
for (int i = 0, j; i < n; i++) {  
    for (j = 0; j < n * n; j++) {  
        sum++;  
    }  
}
```

# Análise de Algoritmos

- Exercício 1: Qual a ordem de execução do seguinte bloco?

```
for (int i = 0, j; i < n; i++) {  
    for (j = 0; j < n * n; j++) {  
        sum++;  
    }  
}
```

$O(N^3)$

# Análise de Algoritmos

- **Exercício 2:** Qual a ordem de execução e tempo efetivo do bloco seguinte?

```
for (int i = 0, j, k; i < n; i++) {  
    for (j = 0; j < i * i; j++) {  
        if (j % i == 0) {  
            for (k = 0; k < j; k++) {  
                sum++;  
            }  
        }  
    }  
}
```

# Análise de Algoritmos

- **Exercício 2:** Qual a ordem de execução e tempo efetivo do bloco seguinte?

```
for (int i = 0, j, k; i < n; i++) {  
    for (j = 0; j < i * i; j++) {  
        if (j % i == 0) {  
            for (k = 0; k < j; k++) {  
                sum++;  
            }  
        }  
    }  
}
```

$O(N^5)$

Apesar de  
o tempo  
efetivo ser  
bastante  
menor!

# Análise de Algoritmos

- **Exercício 3:** Considere um algoritmo que demora 0,5s para uma entrada de 100 elementos. Qual será o tempo se a entrada for de 500 elementos e o tempo de execução for linear (quadrático ou cúbico)?

# Análise de Algoritmos

- **Exercício 3:** Considere um algoritmo que demora 0,5s para uma entrada de 100 elementos. Qual será o tempo se a entrada for de 500 elementos e o tempo de execução for linear (quadrático ou cúbico)?
  - Linear
    - $0,5s \times 500 / 100 = 2,5s$

# Análise de Algoritmos

- **Exercício 3:** Considere um algoritmo que demora 0,5s para uma entrada de 100 elementos. Qual será o tempo se a entrada for de 500 elementos e o tempo de execução for linear (quadrático ou cúbico)?
  - Linear
    - $0,5s \times 500 / 100 = 2,5s$
  - Quadrático
    - 100 elementos  $\Rightarrow 100 \times 100$  operações = 10.000 operações
    - 500 elementos  $\Rightarrow 500 \times 500$  operações = 250.000 operações
    - Resposta:  $0,5s \times 250.000 / 10.000 = 12,5s$



# Análise de Algoritmos

- **Exercício 3:** Considere um algoritmo que demora 0,5s para uma entrada de 100 elementos. Qual será o tempo se a entrada for de 500 elementos e o tempo de execução for linear (quadrático ou cúbico)?
  - Linear
    - $0,5s \times 500 / 100 = 2,5s$
  - Quadrático
    - 100 elementos  $\Rightarrow 100 \times 100$  operações = 10.000 operações
    - 500 elementos  $\Rightarrow 500 \times 500$  operações = 250.000 operações
    - Resposta:  $0,5s \times 250.000 / 10.000 = 12,5s$
  - Cúbico
    - 100 elementos  $\Rightarrow 100 \times 100 \times 100$  operações = 1.000.000 operações
    - 500 elementos  $\Rightarrow 500 \times 500 \times 500$  operações = 125.000.000 operações
    - Resposta:  $0,5s \times 125.000.000 / 1.000.000 = 62,5s$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- Definição do problema: Dada uma sequência de números inteiros pretende-se encontrar uma subsequência com a soma (não negativa) máxima
- Exemplo:
  - Na sequência  $\{-2, 11, -4, 5, 7, -3, 13, 7, -5, 3\}$  temos a subsequência com soma máxima:
    - compreendida entre os índices 1 e 7
    - $\{11, -4, 5, 7, -3, 13, 7\}$
    - com valor 36

# Análise de Algoritmos

## Subsequência com Soma Máxima

- Classes a construir de seguida

### Estatísticas

classe *Estatistica*

classe **EstatisticaDeIteracoes**



### Análise de Complexidade

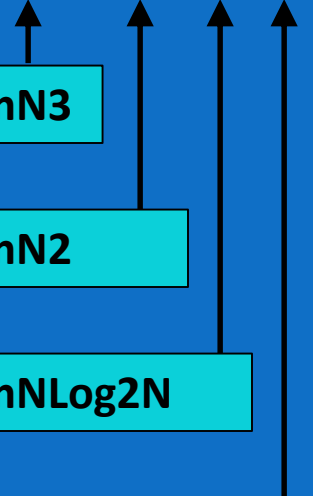
classe *SubsequenciaSomaMaxima*

classe **SubsequenciaSomaMaximaDeOrdemN3**

classe **SubsequenciaSomaMaximaDeOrdemN2**

classe **SubsequenciaSomaMaximaDeOrdemNLog2N**

classe **SubsequenciaSomaMaximaDeOrdemN**



# Análise de Algoritmos

## Subsequência com Soma Máxima

- Classes a construir de seguida

### Utilização

classe **MainTeoricaSubsequenciaSomaMaximaDeOrdemN3**

classe **MainTeoricaSubsequenciaSomaMaximaDeOrdemN2**

classe **MainTeoricaSubsequenciaSomaMaximaDeOrdemNLog2N**

classe **MainTeoricaSubsequenciaSomaMaximaDeOrdemN**

classe **MainTeoricaComparacaoSubsequenciaSomaMaxima**

# Análise de Algoritmos

## Subsequência com Soma Máxima

```
import java.util.Arrays;

public abstract class Estatistica {
    private int tamanho;
    private String[] nomesContadores;
    private long[] contadores;
    private long tempoInicial;
    private long tempoTotal;

    public Estatistica(int tamanho, String... nomesContadores) {
        this.tamanho = tamanho;
        this.nomesContadores = nomesContadores;
        this.contadores = new long[nomesContadores.length];
        iniciar();
    }

    private void iniciar() {
        tempoInicial = System.nanoTime();
        Arrays.fill(contadores, 0);
        tempoTotal = -1;
    }
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

```
public void parar() {  
    tempoTotal = System.nanoTime() - tempoInicial;  
}  
  
protected void incrementarContador(int indice) {  
    contadores[indice]++;  
}  
  
public void apresentar() {  
    System.out.println("Tempo de execução: " + getTempoTotal() + " ms");  
    for (int i = 0; i < nomesContadores.length; i++) {  
        System.out.println(nomesContadores[i] + ": " + contadores[i]);  
    }  
    System.out.println();  
}  
  
public int getTamanho() {  
    return tamanho;  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

```
public int getNumeroContadores() {  
    return contadores.length;  
}  
  
public String getNomeContador(int indice) {  
    return nomesContadores[indice];  
}  
  
public long getContador(int indice) {  
    return contadores[indice];  
}  
  
public double getTempoTotal() {  
    return (tempoTotal / 1000000d);  
}  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

```
public class EstatisticaDeIteracoes extends Estatistica {  
    public EstatisticaDeIteracoes(int tamanho) {  
        super(tamanho, "Número de Iterações");  
    }  
  
    public void incrementarIteracoes() {  
        incrementarContador(0);  
    }  
  
    public long getIteracoes() {  
        return getContador(0);  
    }  
}
```



# Análise de Algoritmos

## Subsequência com Soma Máxima

```
public class Par<T1, T2> {  
    private T1 primeiro;  
    private T2 segundo;  
  
    public Par(T1 primeiro, T2 segundo) {  
        this.primeiro = primeiro;  
        this.segundo = segundo;  
    }  
  
    public T1 getPrimeiro() {  
        return primeiro;  
    }  
  
    public void setPrimeiro(T1 primeiro) {  
        this.primeiro = primeiro;  
    }  
  
    public T2 getSegundo() {  
        return segundo;  
    }  
  
    public void setSegundo(T2 segundo) {  
        this.segundo = segundo;  
    }  
  
    public String toString() {  
        return "Primeiro: " + primeiro + " Segundo: " + segundo;  
    }  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

```
public abstract class SubsequenciaSomaMaxima {
    public abstract long executar(EstatisticaDeIteracoes estatistica,
        Par<Integer, Integer> indicesInicialEFinal, int... elementos);

    public EstatisticaDeIteracoes getEstatistica(int... elementos) {
        EstatisticaDeIteracoes estatistica =
            new EstatisticaDeIteracoes(elementos.length);
        Par<Integer, Integer> indicesInicialEFinal = new Par<>(0, 0);
        long soma = executar(estatistica, indicesInicialEFinal, elementos);
        estatistica.parar();
        System.out.println("Subsequência da soma máxima calculada com " +
            getClass().getSimpleName() + ": ");
        System.out.println("Índice inicial = " + indicesInicialEFinal.getPrimeiro()
            + " Índice final = " + indicesInicialEFinal.getSegundo()
            + " Soma = " + soma);
        VetorDeInteiros.apresentar(10, Arrays.copyOfRange(elementos,
            indicesInicialEFinal.getPrimeiro(),
            indicesInicialEFinal.getSegundo() + 1));
        estatistica.apresentar();
        return estatistica;
    }
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

- **Solução mais simples:** Consiste em calcular todas as somas possíveis para determinar aquela que tem valor máximo

```
public class SubsequenciaSomaMaximaDeOrdemN3 extends SubsequenciaSomaMaxima {
    public long executar(EstatisticaDeIteracoes estatistica,
                        Par<Integer, Integer> indicesInicialEFinal, int... elementos) {
        long somaMaxima = 0, somaAtual;
        int inicio, fim;
        for (int i = 0; i < elementos.length; i++) {
            for (int j = i; j < elementos.length; j++) {
                somaAtual = 0;
                for (int k = i; k <= j; k++) {
                    somaAtual += elementos[k];
                    if (somaAtual > somaMaxima) {
                        inicio = i;
                        fim = j;
                        somaMaxima = somaAtual;
                    }
                }
                estatistica.incrementarIteracoes();
            }
        }
        indicesInicialEFinal.setPrimeiro(inicio);
        indicesInicialEFinal.setSegundo(fim);
        return somaMaxima;
    }
}
```

Note-se que as variáveis **inicio** e **fim** apenas são necessárias para efeitos de simulação em JGrasp

# Análise de Algoritmos

## Subsequência com Soma Máxima

- **Solução mais simples:** Consiste em calcular todas as somas possíveis para determinar aquela que tem valor máximo

```
public class MainTeoricaSubsequenciaSomaMaximaDeOrdemN3 {  
  
    public MainTeoricaSubsequenciaSomaMaximaDeOrdemN3() {  
        SubsequenciaSomaMaxima subsequenciaSomaMaxima =  
            new SubsequenciaSomaMaximaDeOrdemN3();  
        subsequenciaSomaMaxima.getEstatistica(-2, 11, -4, 5, 7, -3, 13, 7, -5, 3);  
    }  
  
    public static void main(String[] args) {  
        new MainTeoricaSubsequenciaSomaMaximaDeOrdemN3();  
    }  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

***jGRASP***

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 2ª solução: O ciclo mais interior não é necessário

```
public class SubsequenciaSomaMaximaDeOrdemN2 extends SubsequenciaSomaMaxima {  
    public long executar(EstatisticaDeIteracoes estatistica,  
                        Par<Integer, Integer> indicesInicialEFinal, int... elementos) {  
        long somaMaxima = 0, somaAtual;  
        int inicio = 0, fim = 0;  
        for (int i = 0; i < elementos.length; i++) {  
            somaAtual = 0;  
            for (int j = i; j < elementos.length; j++) {  
                somaAtual += elementos[j];  
                if (somaAtual > somaMaxima) {  
                    inicio = i;  
                    fim = j;  
                    somaMaxima = somaAtual;  
                }  
            }  
            estatistica.incrementarIteracoes();  
        }  
        indicesInicialEFinal.setPrimeiro(inicio);  
        indicesInicialEFinal.setSegundo(fim);  
        return somaMaxima;  
    }  
}
```

Note-se que as variáveis inicio e fim apenas são necessárias para efeitos de simulação em JGrasp

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 2ª solução: O ciclo mais interior não é necessário

```
public class MainTeoricaSubsequenciaSomaMaximaDeOrdemN2 {  
  
    public MainTeoricaSubsequenciaSomaMaximaDeOrdemN2() {  
        SubsequenciaSomaMaxima subsequenciaSomaMaxima =  
            new SubsequenciaSomaMaximaDeOrdemN2();  
        subsequenciaSomaMaxima.getEstatistica(-2, 11, -4, 5, 7, -3, 13, 7, -5, 3);  
    }  
  
    public static void main(String[] args) {  
        new MainTeoricaSubsequenciaSomaMaximaDeOrdemN2();  
    }  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

***jGRASP***



# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução:
  - A subsequência com soma máxima está:
    - toda na metade esquerda da sequência
    - ou toda na metade direita da sequência
    - ou na junção das duas metades da sequência (isto é, o resultado da adição da soma máxima do meio para a esquerda com o da soma máxima do meio + 1 para a direita)
  - As duas primeiras são facilmente calculadas de modo recursivo
  - O terceiro caso é facilmente resolvido com dois ciclos

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Versão Recursiva – caso base

```
public class SubsequenciaSomaMaximaDeOrdemNLog2N
    extends SubsequenciaSomaMaxima {

    public long executar(EstatisticaDeIteracoes estatistica,
        Par<Integer, Integer> indicesInicialEFinal, int... elementos) {
        return executarRecursivo(estatistica, indicesInicialEFinal, elementos, 0,
            elementos.length - 1);
    }

    private long executarRecursivo(EstatisticaDeIteracoes estatistica,
        Par<Integer, Integer> indicesInicialEFinal,
        int[] elementos, int esq, int dir) {
        estatistica.incrementarIteracoes();
        long somaMaxima = 0;
        int inicio, fim;
        if (esq == dir) {
            indicesInicialEFinal.setPrimeiro(esq);
            indicesInicialEFinal.setSegundo(dir);
            inicio = fim = esq;
            if (elementos[esq] > 0) {
                return somaMaxima = elementos[esq];
            }
            return somaMaxima = 0;
        }
    }
```

Note-se que as variáveis **somaMaxima**, **inicio** e **fim** apenas são necessárias para efeitos de simulação em JGrasp

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Versão Recursiva – chamadas

...

```
int meio = (esq + dir) / 2;
Par<Integer, Integer> indicesInicialEFinalEsq = new Par<>(0,0);
Par<Integer, Integer> indicesInicialEFinalDir = new Par<>(0,0);
long somaMaximaEsq = executarRecursivo(estatistica,
                                       indicesInicialEFinalEsq,
                                       elementos, esq, meio);

long somaMaximaDir = executarRecursivo(estatistica,
                                       indicesInicialEFinalDir,
                                       elementos, meio + 1, dir);

int inicioEsq = indicesInicialEFinalEsq.getPrimeiro();
int fimEsq = indicesInicialEFinalEsq.getSegundo();
int inicioDir = indicesInicialEFinalDir.getPrimeiro();
int fimDir = indicesInicialEFinalDir.getSegundo();
```

Note-se que as variáveis **inicioEsq**, **fimEsq**, **inicioDir** e **fimDir** apenas são necessárias para efeitos de simulação em JGrasp

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Versão Recursiva – cálculo das somas do meio para a esquerda e do meio + 1 para a direita

...

```
long somaMaximaMeioEsq = 0, somaAtualMeioEsq = 0;
for (int i = meio; i >= esq; i--) {
    somaAtualMeioEsq += elementos[i];
    if (somaAtualMeioEsq > somaMaximaMeioEsq) {
        somaMaximaMeioEsq = somaAtualMeioEsq;
    }
    estatistica.incrementarIteracoes();
}

long somaMaximaMeioDir = 0, somaAtualMeioDir = 0;
for (int i = meio + 1; i <= dir; i++) {
    somaAtualMeioDir += elementos[i];
    if (somaAtualMeioDir > somaMaximaMeioDir) {
        somaMaximaMeioDir = somaAtualMeioDir;
    }
    estatistica.incrementarIteracoes();
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Versão Recursiva – cálculo e retorno da soma máxima

...

```
long somaMaximaEsqDir = somaMaximaMeioEsq + somaMaximaMeioDir;
if (somaMaximaEsq >= somaMaximaDir && somaMaximaEsq >= somaMaximaEsqDir) {
    inicio = indicesInicialEFinalEsq.getPrimeiro();
    fim = indicesInicialEFinalEsq.getSegundo();
    indicesInicialEFinal.setPrimeiro(inicio);
    indicesInicialEFinal.setSegundo(fim);
    somaMaxima = somaMaximaEsq;
    return somaMaxima;
}
if (somaMaximaDir >= somaMaximaEsqDir) {
    inicio = indicesInicialEFinalDir.getPrimeiro();
    fim = indicesInicialEFinalDir.getSegundo();
    indicesInicialEFinal.setPrimeiro(inicio);
    indicesInicialEFinal.setSegundo(fim);
    somaMaxima = somaMaximaDir;
    return somaMaxima;
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Versão Recursiva – cálculo e retorno da soma máxima

...

```
    inicio = indicesInicialEFinalEsq.getPrimeiro();  
    fim = indicesInicialEFinalDir.getSegundo();  
    indicesInicialEFinal.setPrimeiro(inicio);  
    indicesInicialEFinal.setSegundo(fim);  
    somaMaxima = somaMaximaEsqDir;  
    return somaMaxima;  
}  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Versão Recursiva

```
public class MainTeoricaSubsequenciaSomaMaximaDeOrdemNLog2N {  
  
    public MainTeoricaSubsequenciaSomaMaximaDeOrdemNLog2N() {  
        SubsequenciaSomaMaxima subsequenciaSomaMaxima =  
            new SubsequenciaSomaMaximaDeOrdemNLog2N();  
        subsequenciaSomaMaxima.getEstatistica(-2, 11, -4, 5, 7, -3, 13, 7, -5, 3);  
    }  
  
    public static void main(String[] args) {  
        new MainTeoricaSubsequenciaSomaMaximaDeOrdemNLog2N();  
    }  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

***jGRASP***



# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade
    - Considerando  $T(N)$  o tempo necessário para calcular a subsequência de soma máxima para  $N$  elementos, temos:

$$T(N) = \begin{cases} 1 & \Leftarrow N = 1 \\ 2T(N/2) + N & \Leftarrow N > 1 \end{cases}$$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$T(N) = 2T(N / 2) + N$$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$\begin{aligned}T(N) &= 2T(N / 2) + N \\&= 2[2T(N / 2^2) + N / 2] + N\end{aligned}$$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$\begin{aligned}T(N) &= 2T(N / 2) + N \\&= 2[2T(N / 2^2) + N / 2] + N \\&= 2^2 T(N / 2^2) + 2N\end{aligned}$$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$\begin{aligned}T(N) &= 2T(N / 2) + N \\&= 2[2T(N / 2^2) + N / 2] + N \\&= 2^2 T(N / 2^2) + 2N \\&= 2^2 [2T(N / 2^3) + N / 2^2] + 2N\end{aligned}$$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$\begin{aligned}T(N) &= 2T(N/2) + N \\&= 2[2T(N/2^2) + N/2] + N \\&= 2^2T(N/2^2) + 2N \\&= 2^2[2T(N/2^3) + N/2^2] + 2N \\&= 2^3T(N/2^3) + 3N\end{aligned}$$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$\begin{aligned}T(N) &= 2T(N/2) + N \\&= 2[2T(N/2^2) + N/2] + N \\&= 2^2T(N/2^2) + 2N \\&= 2^2[2T(N/2^3) + N/2^2] + 2N \\&= 2^3T(N/2^3) + 3N \\&\dots \\&= 2^kT(N/2^k) + kN\end{aligned}$$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$\begin{aligned}T(N) &= 2T(N/2) + N \\&= 2[2T(N/2^2) + N/2] + N \\&= 2^2T(N/2^2) + 2N \\&= 2^2[2T(N/2^3) + N/2^2] + 2N \\&= 2^3T(N/2^3) + 3N \\&\dots \\&= 2^kT(N/2^k) + kN\end{aligned}$$

como o ponto de paragem é  $N=1$  temos  $N/2^k=1$ , ou seja,  $N=2^k$ , ou ainda,  $k=\log_2 N$ , logo

$$T(N) = N + N \log_2 N$$



# Análise de Algoritmos

## Subsequência com Soma Máxima

- 3ª solução: Recursiva
  - Análise de complexidade

$$\begin{aligned}T(N) &= 2T(N/2) + N \\&= 2[2T(N/2^2) + N/2] + N \\&= 2^2T(N/2^2) + 2N \\&= 2^2[2T(N/2^3) + N/2^2] + 2N \\&= 2^3T(N/2^3) + 3N \\&\dots \\&= 2^kT(N/2^k) + kN\end{aligned}$$

como o ponto de paragem é  $N=1$  temos  $N/2^k=1$ , ou seja,  $N=2^k$ , ou ainda,  $k=\log_2 N$ , logo

$$T(N) = N + N \log_2 N$$

Algoritmo de ordem  
 $O(N \log_2 N)$

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 4ª solução: Algoritmo Linear (solução ótima)

```
public class SubsequenciaSomaMaximaDeOrdemN extends SubsequenciaSomaMaxima {  
    public long executar(EstatisticaDeIteracoes estatistica,  
        Par<Integer, Integer> indicesInicialEFinal, int... elementos) {  
        long somaMaxima = 0, somaAtual = 0;  
        int inicio = 0, fim = 0, inicioAtual = 0;  
        for (int j = 0; j < elementos.length; j++) {  
            somaAtual += elementos[j];  
            if (somaAtual < 0) {  
                inicioAtual = j + 1;  
                somaAtual = 0;  
            } else if (somaAtual > somaMaxima) {  
                fim = j;  
                inicio = inicioAtual;  
                somaMaxima = somaAtual;  
            }  
            indicesInicialEFinal.setPrimeiro(inicio);  
            indicesInicialEFinal.setSegundo(fim);  
            estatistica.incrementarIteracoes();  
        }  
        return somaMaxima;  
    }  
}
```

Note-se que as variáveis **inicio** e **fim** apenas são necessárias para efeitos de simulação em JGrasp

# Análise de Algoritmos

## Subsequência com Soma Máxima

- 4ª solução: Algoritmo Linear (solução ótima)

```
public class MainTeoricaSubsequenciaSomaMaximaDeOrdemN {  
  
    public MainTeoricaSubsequenciaSomaMaximaDeOrdemN() {  
        SubsequenciaSomaMaxima subsequenciaSomaMaxima =  
            new SubsequenciaSomaMaximaDeOrdemN();  
        subsequenciaSomaMaxima.getEstatistica(-2, 11, -4, 5, 7, -3, 13, 7, -5, 3);  
    }  
  
    public static void main(String[] args) {  
        new MainTeoricaSubsequenciaSomaMaximaDeOrdemN();  
    }  
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

***jGRASP***

# Análise de Algoritmos

## Subsequência com Soma Máxima

- Análise comparativa das 4 soluções

```
public class MainTeoricaComparacaoSubsequenciaSomaMaxima {
    private static final int TAMANHO = 100;
    private static final int NUMERO_EXECUCOES = 10;

    public MainTeoricaComparacaoSubsequenciaSomaMaxima() {
        VisualizadorEstatisticas v = new VisualizadorEstatisticas();
        v.adicionarEstatisticas("O(N3)",
                               getEstatisticas(new SubsequenciaSomaMaximaDeOrdemN3()));
        v.adicionarEstatisticas("O(N2)",
                               getEstatisticas(new SubsequenciaSomaMaximaDeOrdemN2()));
        v.adicionarEstatisticas("O(NLog2N)",
                               getEstatisticas(new SubsequenciaSomaMaximaDeOrdemNLog2N()));
        v.adicionarEstatisticas("O(N)",
                               getEstatisticas(new SubsequenciaSomaMaximaDeOrdemN()));

        v.visualizar();
    }
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

- Análise comparativa das 4 soluções

```
public static void main(String[] args) {
    new MainTeoricaComparacaoSubsequenciaSomaMaxima();
}

private List<Estatistica> getEstatisticas(SubsequenciaSomaMaxima algoritmo) {
    List<Estatistica> estatisticas = new ArrayList<>();
    for (int i = 1; i <= NUMERO_EXECUCOES; i++) {
        EstatisticaDeIteracoes estatistica =
            algoritmo.getEstatistica(VetorDeInteiros.
                criarAleatorioInt(TAMANHO * i, -TAMANHO * 10, TAMANHO * 10));
        estatisticas.add(estatistica);
    }
    return estatisticas;
}
}
```

# Análise de Algoritmos

## Subsequência com Soma Máxima

