Universidade Estadual de Maringá
Departamento de Informática
Curso: Informática
Disciplina: Projeto e Análise de Algoritmos
Professor: Rodrigo Calvo

Projeto e Análise de Algoritmos
Trabalho I

Integrante	RA
Douglas Mezuraro	95676
Victor Glauber Lopes Silva	68474

Arquivos de código contidos no trabalho

- ***PAA**.dpr*: Arquivo do projeto Delphi
- *I**mpl.Algorithms**.pas*: Arquivo com a implementação dos exercícios *a* e *b*
- ***Impl.Input**.pas*: Arquivo que contém a classe onde as informações do arquivo de entrada são armazenadas
- ***Form.Main**.pas*: Arquivo que é a interface gráfica utilizada para facilitar a interação do usuário
- ***Form.Main**.dfm*: Arquivo que o delphi utiliza para montar o formulário em tempo de execução
- ***Impl.Tests**.pas*: Arquivo que contém testes unitários usados para facilitar a homologação dos algoritmos desenvolvidos para o exercício *a* e *b*

Outros arquivos arquivos com extensões que não sejam *.dpr*, *.pas* e *.dfm* são gerados e utilizados pelo *Delphi* para eventuais configurações adicionais do projeto.

Informações iniciais

A implementação foi feita em *object-pascal* (*Delphi*). O saída da execução do trabalho é mostrado em forma de texto no próprio formulário. Foi implementado também uma espécie de testes unitários para facilitar a homologação dos resultados da execução dos algoritmos.

Arquivo de entrada

Os algoritmos não validam se o arquivo de entrada estão seguindo a lógica apresentada no enunciado do trabalho, então caso a entrada não obedeça às regras apresentadas a execução pode apresentar problemas.

O arquivo de entrada deve ser no formato *json* conforme o exemplo abaixo:

```
{
	"arrays" :
		[
			{ "key" : [1, 3, 7, 10, 13, 12, 11], "value" : 4 },
			{ "key" : [0, 1, 2, 3, 4, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5], "value" : 5 },
			{ "key" : [9, 7, 5, 3, 1], "value" : 0 },
			{ "key" : [0, 2, 4, 8, 16, 32, 64, 128], "value" : 7 },
			{ "key" : [9, 8, 7, 6, 5, 4, 3, 2, 1, 0], "value" : 0 } ,
			{ "key" : [0, 9, 8, 7, 6, 5, 4, 3, 2, 1], "value" : 1 } ,
			{ "key" : [0, 1, 9, 8, 7, 6, 5, 4, 3, 2], "value" : 2 } ,
			{ "key" : [0, 1, 2, 9, 8, 7, 6, 5, 4, 3], "value" : 3 } ,
			{ "key" : [0, 1, 2, 3, 9, 8, 7, 6, 5, 4], "value" : 4 } ,
			{ "key" : [0, 1, 2, 3, 4, 9, 8, 7, 6, 5], "value" : 5 } ,
			{ "key" : [0, 1, 2, 3, 4, 5, 9, 8, 7, 6], "value" : 6 } ,
			{ "key" : [0, 1, 2, 3, 4, 5, 6, 9, 8, 7], "value" : 7 } ,
			{ "key" : [0, 1, 2, 3, 4, 5, 6, 7, 9, 8], "value" : 8 } ,
			{ "key" : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], "value" : 9 }
		]
}
```

Onde *key* é o array a ser executado e *value* o índice do maior elemento do array *key*.

Instruções de execução

1. Abrir o arquivo bin/*PAA.exe*
2. Abrir o arquivo de entrada bin/input-file.*json*
3. Clicar em executar para ver a saída

Exercício A

Foi desenvolvido o algoritmo recursivo chamado *HighestElement()* que é baseado no algoritmo de *busca-binária* que também é executado com tempo de O(⌊log⌋n). A principal diferença é que o algoritmo da *busca-binária* recebe um array *A* ordenado e um elemento *k* qualquer e retorna o índice deste elemento *k* no array *A*. Já no exercício *A* o algoritmo recebe um array ordenado crescentemente até um elemento *k* qualquer e ordenado decrescentemente a partir ele e então retornar o índice deste elemento *k*.

```
class function TImpl.HighestElement(const A: TArray<Integer>; const L, R: Integer): Integer;
```

A lógica utilizada neste algoritmo é sempre partir o array ao meio e comparar se o elemento no índice encontrado é maior que o elemento anterior e maior que o próximo elemento.

No algoritmo temos quatro casos:

1. L = R: Quando o maior elemento do array ou está na primeira ou na última posição
2. A[M] > A[M+1] e A[M] > A[M-1]: quando a quebra ao meio (M) foi feita exatamente na posição do maior elemento do array
3. A[M] < A[M+1]: o elemento do meio é menor que o próximo elemento, ou seja, o maior elemento está na porção direita à posição em que foi feita a quebra
4. A[M] > A[M+1]: o elemento do meio é maior que o próximo elemento, ou seja, o maior elemento está na porção esquerda à posição em que foi feita a quebra

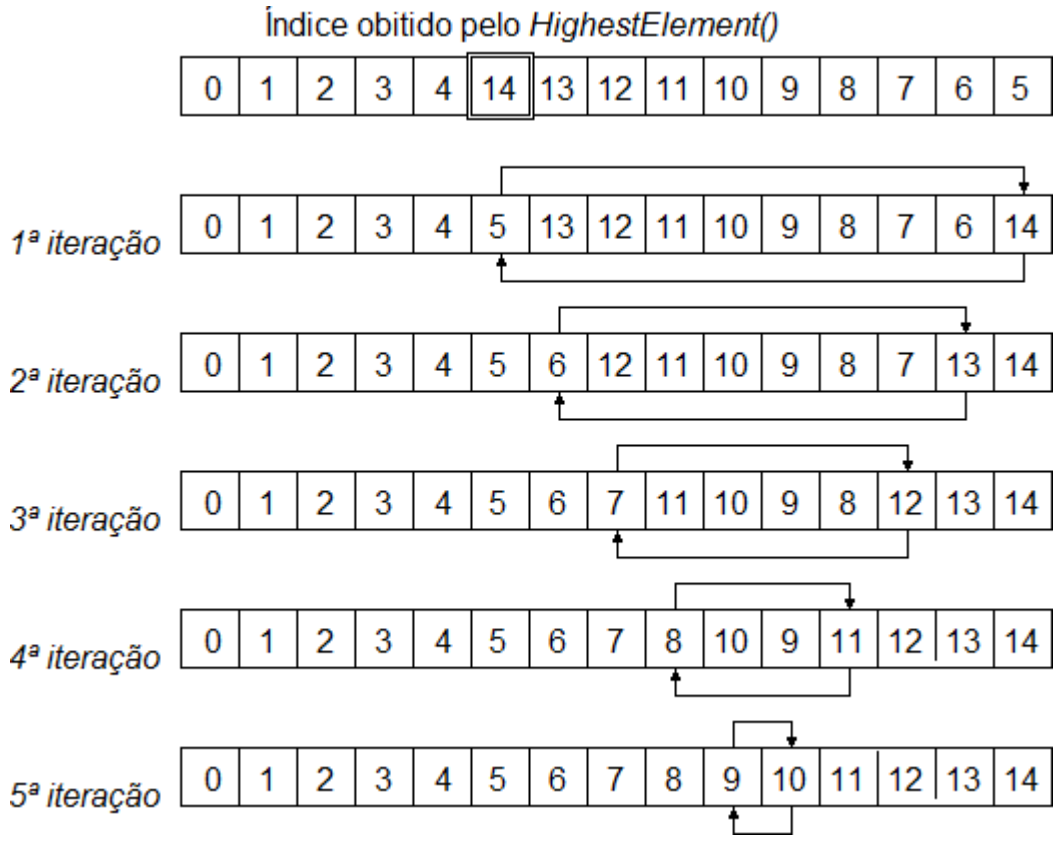
Exercício B

Foi desenvolvido o método *Sort()* que roda em tempo O(n). Sabendo que o array *A* está ordenado decrescentemente a partir de um índice *k* qualquer foi utilizado foi pensado em uma lógica que apenas inverte os valores do array a partir deste elemento *k*.

```
class function TImpl.Sort(const A: TArray<Integer>): TArray<Integer>;
```

Então, inicialmente a variável L recebe o índice do maior elemento do array *A* a partir da execução do método *HighestElement()* e a variável *R* recebe o índice do último elemento. Enquanto a diferença entre *R* e *L* for positiva é feito uma troca dos valores dos índices *R* e *L* e no fim de cada iteração é decrementada o valor das variáveis *R* e *L.*

Exemplo de chamada do método *Sort* passando como parâmetro o array [0, 1, 2, 3, 4, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5]:


Índice obitido pelo HighestElement()
0 1 2 3 4 14 13 12 11 10 9 8 7 6 5
1ª iteração
0 1 2 3 4 5 13 12 11 10 9 8 7 6 14
2ª iteração
0 1 2 3 4 5 6 12 11 10 9 8 7 13 14
3ª iteração
0 1 2 3 4 5 6 7 11 10 9 8 12 13 14
4ª iteração
0 1 2 3 4 5 6 7 8 10 9 11 12 13 14
5ª iteração
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14