

## Trabalho Prático 0 - AEDS III

0.0.0

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	cell Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.1.2	Field Documentation . . . . .	5
3.1.2.1	max . . . . .	5
3.1.2.2	min . . . . .	5
3.1.2.3	sum . . . . .	5
<b>4</b>	<b>File Documentation</b>	<b>7</b>
4.1	src/arvore.c File Reference . . . . .	7
4.1.1	Function Documentation . . . . .	7
4.1.1.1	add() . . . . .	7
4.1.1.2	main() . . . . .	8
4.1.1.3	sub() . . . . .	8
4.2	src/lib/cell.c File Reference . . . . .	8
4.2.1	Function Documentation . . . . .	8
4.2.1.1	Cell_create() . . . . .	9
4.2.1.2	Cell_destroy() . . . . .	9
4.2.1.3	Cell_fill() . . . . .	9

4.2.1.4	Cell_print()	10
4.3	src/lib/cell.h File Reference	10
4.3.1	Typedef Documentation	10
4.3.1.1	Cell	10
4.3.2	Function Documentation	11
4.3.2.1	Cell_create()	11
4.3.2.2	Cell_destroy()	11
4.3.2.3	Cell_fill()	11
4.3.2.4	Cell_print()	12
4.4	src/lib/matrix.c File Reference	12
4.4.1	Function Documentation	12
4.4.1.1	Matrix_create()	12
4.4.1.2	Matrix_destroy()	13
4.4.1.3	Matrix_fill()	13
4.4.1.4	Matrix_print()	13
4.5	src/lib/matrix.h File Reference	14
4.5.1	Function Documentation	14
4.5.1.1	Matrix_create()	14
4.5.1.2	Matrix_destroy()	15
4.5.1.3	Matrix_fill()	15
4.5.1.4	Matrix_print()	15
4.6	src/lib/segtree.c File Reference	16
4.6.1	Function Documentation	16
4.6.1.1	SegTree_construct()	16
4.6.1.2	SegTree_create()	17
4.6.1.3	SegTree_destroy()	17
4.6.1.4	SegTree_query()	18
4.6.1.5	SegTree_rangeQuery()	18
4.6.1.6	SegTree_rangeUpdate()	19
4.6.1.7	SegTree_size()	20

---

4.6.1.8	SegTree_update()	20
4.7	src/lib/segtree.h File Reference	21
4.7.1	Function Documentation	21
4.7.1.1	SegTree_construct()	21
4.7.1.2	SegTree_create()	22
4.7.1.3	SegTree_destroy()	22
4.7.1.4	SegTree_query()	23
4.7.1.5	SegTree_rangeQuery()	23
4.7.1.6	SegTree_rangeUpdate()	24
4.7.1.7	SegTree_size()	25
4.7.1.8	SegTree_update()	25
4.8	src/matrix.c File Reference	26
4.8.1	Function Documentation	26
4.8.1.1	main()	26
<b>Index</b>		<b>27</b>



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">cell</a>	Cell structure . . . . .	<a href="#">5</a>
----------------------	--------------------------	-------------------





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">arvore.c</a>	7
src/ <a href="#">matriz.c</a>	26
src/lib/ <a href="#">cell.c</a>	8
src/lib/ <a href="#">cell.h</a>	10
src/lib/ <a href="#">matrix.c</a>	12
src/lib/ <a href="#">matrix.h</a>	14
src/lib/ <a href="#">segtree.c</a>	16
src/lib/ <a href="#">segtree.h</a>	21



## Chapter 3

# Data Structure Documentation

### 3.1 cell Struct Reference

Cell structure.

```
#include <cell.h>
```

#### Data Fields

- long long [min](#)
- long long [max](#)
- long long [sum](#)

#### 3.1.1 Detailed Description

Cell structure.

This struct stores the max, min and the sum of the interval.

#### 3.1.2 Field Documentation

##### 3.1.2.1 max

```
long long max
```

##### 3.1.2.2 min

```
long long min
```

Minimum and Maximum value of the interval

##### 3.1.2.3 sum

```
long long sum
```

Sum of the interval

The documentation for this struct was generated from the following file:

- src/lib/[cell.h](#)



## Chapter 4

# File Documentation

### 4.1 src/arvore.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib/segtree.h"
#include "lib/cell.h"
```

#### Functions

- int [add](#) (int a)
- int [sub](#) (int a)
- int [main](#) ()

*Main function.*

#### 4.1.1 Function Documentation

##### 4.1.1.1 add()

```
int add (
    int a )
```

#### 4.1.1.2 main()

```
int main ( )
```

Main function.

This function uses the Bryan solution based on a Segment Tree that stores the input array data.

#### Returns

0

#### 4.1.1.3 sub()

```
int sub (
    int a )
```

## 4.2 src/lib/cell.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "cell.h"
```

### Functions

- [Cell Cell\\_create](#) ()  
*Creates a new cell and initialize it's values with 0.*
- void [Cell\\_destroy](#) (Cell \*cell)  
*Destroys the given cell.*
- void [Cell\\_print](#) (Cell cell)  
*Prints the content of the given cell.*
- void [Cell\\_fill](#) (Cell \*cell, int \*array, int i, int j)  
*Fill the cell data with the given array and intervals.*

#### 4.2.1 Function Documentation

#### 4.2.1.1 Cell\_create()

```
Cell Cell_create ( )
```

Creates a new cell and initialize it's values with 0.

Complexity: O(1)

##### Returns

The created cell

#### 4.2.1.2 Cell\_destroy()

```
void Cell_destroy (
    Cell * cell )
```

Destroys the given cell.

Complexity: O(1)

##### Parameters

<i>cell</i>	The cell to be destroyed
-------------	--------------------------

#### 4.2.1.3 Cell\_fill()

```
void Cell_fill (
    Cell * cell,
    int * array,
    int i,
    int j )
```

Fill the cell data with the given array and intervals.

Complexity: O(size-end)

##### Parameters

	<i>cell</i>	The cell
	<i>array</i>	The array
in	<i>i</i>	Start of the interval
in	<i>j</i>	End of the interval

#### 4.2.1.4 Cell\_print()

```
void Cell_print (
    Cell cell )
```

Prints the content of the given cell.

It doesn't print a new line.

Complexity: O(1)

##### Parameters

in	cell	The cell
----	------	----------

## 4.3 src/lib/cell.h File Reference

### Data Structures

- struct `cell`  
*Cell structure.*

### Typedefs

- typedef struct `cell` `Cell`

### Functions

- `Cell Cell_create ()`  
*Creates a new cell and initialize it's values with 0.*
- `void Cell_destroy (Cell *cell)`  
*Destroys the given cell.*
- `void Cell_print (Cell cell)`  
*Prints the content of the given cell.*
- `void Cell_fill (Cell *cell, int *array, int i, int j)`  
*Fill the cell data with the given array and intervals.*

### 4.3.1 Typedef Documentation

#### 4.3.1.1 Cell

```
typedef struct cell Cell
```



## 4.3.2 Function Documentation

### 4.3.2.1 Cell\_create()

```
Cell Cell_create ( )
```

Creates a new cell and initialize it's values with 0.

Complexity: O(1)

#### Returns

The created cell

### 4.3.2.2 Cell\_destroy()

```
void Cell_destroy (
    Cell * cell )
```

Destroys the given cell.

Complexity: O(1)

#### Parameters

<i>cell</i>	The cell to be destroyed
-------------	--------------------------

### 4.3.2.3 Cell\_fill()

```
void Cell_fill (
    Cell * cell,
    int * array,
    int i,
    int j )
```

Fill the cell data with the given array and intervals.

Complexity: O(size-end)

#### Parameters

	<i>cell</i>	The cell
	<i>array</i>	The array
in	<i>i</i>	Start of the interval
in	<i>j</i>	End of the interval

#### 4.3.2.4 Cell\_print()

```
void Cell_print (
    Cell cell )
```

Prints the content of the given cell.

It doesn't print a new line.

Complexity: O(1)

##### Parameters

in	cell	The cell
----	------	----------

## 4.4 src/lib/matrix.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "cell.h"
#include "matrix.h"
```

### Functions

- [Cell \\*\\* Matrix\\_create](#) (int n)  
*Builds the matrix with the given array.*
- void [Matrix\\_fill](#) ([Cell \\*\\*matrix](#), int \*array, int n)  
*Fills the matrix with the given data.*
- void [Matrix\\_destroy](#) ([Cell \\*\\*matrix](#), int n)  
*Destroys the matrix.*
- void [Matrix\\_print](#) ([Cell \\*\\*matrix](#), int n)  
*Prints the matrix.*

#### 4.4.1 Function Documentation

##### 4.4.1.1 Matrix\_create()

```
Cell** Matrix_create (
    int n )
```

Builds the matrix with the given array.

Complexity: O(n<sup>2</sup>)

**Parameters**

in	<i>n</i>	The length of the input array
----	----------	-------------------------------

**Returns**

A pointer to the created matrix

**4.4.1.2 Matrix\_destroy()**

```
void Matrix_destroy (
    Cell ** matrix,
    int n )
```

Destroys the matrix.

Complexity:  $O(n)$

**Parameters**

	<i>matrix</i>	The matrix
in	<i>n</i>	The length of the input array

**4.4.1.3 Matrix\_fill()**

```
void Matrix_fill (
    Cell ** matrix,
    int * array,
    int n )
```

Fills the matrix with the given data.

Complexity:  $O(n^3)$ , because the maximum interval given to Cell\_Fill has size  $n$ , and is executed  $n^2$  times.

**Parameters**

	<i>matrix</i>	The matrix
	<i>array</i>	The array
in	<i>n</i>	The length of the input array

**4.4.1.4 Matrix\_print()**

```
void Matrix_print (
```

```
Cell ** matrix,
int n )
```

Prints the matrix.

Complexity:  $O(n^2)$

#### Parameters

	<i>matrix</i>	The matrix
in	<i>n</i>	The length of the input array

## 4.5 src/lib/matrix.h File Reference

```
#include "cell.h"
```

### Functions

- `Cell ** Matrix_create (int n)`  
*Builds the matrix with the given array.*
- `void Matrix_fill (Cell **matrix, int *array, int n)`  
*Fills the matrix with the given data.*
- `void Matrix_destroy (Cell **matrix, int n)`  
*Destroys the matrix.*
- `void Matrix_print (Cell **matrix, int n)`  
*Prints the matrix.*

### 4.5.1 Function Documentation

#### 4.5.1.1 Matrix\_create()

```
Cell** Matrix_create (
    int n )
```

Builds the matrix with the given array.

Complexity:  $O(n^2)$

#### Parameters

in	<i>n</i>	The length of the input array
----	----------	-------------------------------

### Returns

A pointer to the created matrix

#### 4.5.1.2 Matrix\_destroy()

```
void Matrix_destroy (
    Cell ** matrix,
    int n )
```

Destroys the matrix.

Complexity:  $O(n)$

##### Parameters

	<i>matrix</i>	The matrix
in	<i>n</i>	The length of the input array

#### 4.5.1.3 Matrix\_fill()

```
void Matrix_fill (
    Cell ** matrix,
    int * array,
    int n )
```

Fills the matrix with the given data.

Complexity:  $O(n^3)$ , because the maximum interval given to Cell\_Fill has size  $n$ , and is executed  $n^2$  times.

##### Parameters

	<i>matrix</i>	The matrix
	<i>array</i>	The array
in	<i>n</i>	The length of the input array

#### 4.5.1.4 Matrix\_print()

```
void Matrix_print (
    Cell ** matrix,
    int n )
```

Prints the matrix.

Complexity:  $O(n^2)$

## Parameters

	<i>matrix</i>	The matrix
in	<i>n</i>	The length of the input array

## 4.6 src/lib/segtree.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include "segtree.h"
#include "cell.h"
```

### Functions

- void [SegTree\\_construct](#) ([Cell](#) \*segtree, int pos, int \*array, int start, int end)  
*Recursively constructs the Segment Tree.*
- int [SegTree\\_size](#) (int n)  
*Calculates the Segment Tree's number of nodes with the given the input array size.*
- [Cell](#) \* [SegTree\\_create](#) (int \*array, int n)  
*Creates a new Segment Tree.*
- void [SegTree\\_destroy](#) ([Cell](#) \*segtree)  
*Destroys the given Segment Tree.*
- [Cell](#) [SegTree\\_rangeQuery](#) ([Cell](#) \*segtree, int pos, int start, int end, int currStart, int currEnd)  
*Perform a range query on the Segment Tree recursively.*
- [Cell](#) [SegTree\\_query](#) ([Cell](#) \*segtree, int n, int start, int end)  
*Interface for the recursive range query.*
- void [SegTree\\_rangeUpdate](#) ([Cell](#) \*segtree, int pos, int start, int end, int currStart, int currEnd, int(\*transform)(int n))  
*Recursively updates the Segment Tree.*
- void [SegTree\\_update](#) ([Cell](#) \*segtree, int n, int start, int end, int(\*transform)(int n))  
*Interface for the recursive segtree update.*

### 4.6.1 Function Documentation

#### 4.6.1.1 SegTree\_construct()

```
void SegTree_construct (
    Cell * segtree,
    int pos,
    int * array,
    int start,
    int end )
```

Recursively constructs the Segment Tree.

Complexity:

With n being the Segment Tree's number of nodes, the complexity is O(n), because we need to visit every node of the tree to calculate its data.

**Parameters**

	<i>segtree</i>	The Segment Tree array
in	<i>pos</i>	The current position on the segtree
in	<i>array</i>	The input array
in	<i>start</i>	The start of the interval (0-indexed, inclusive)
in	<i>end</i>	The end of the interval (0-indexed, inclusive)

**4.6.1.2 SegTree\_create()**

```
Cell* SegTree_create (
    int * array,
    int n )
```

Creates a new Segment Tree.

Complexity:

The complexity for creating a new Segment Tree is related to the function SegTree\_construct, which is  $O(N)$ , being  $N$  the number of nodes of the tree.

The number of nodes  $N$ , in the best case is  $2 \cdot n - 1$ , with  $n$  being the input array length. In worst case  $N$  will not pass from  $4 \cdot n - 1$ . The full explanation for this is on the SegTree\_size function description.

In all cases, the complexity for creating a new Segment Tree is  $O(n)$ .

**Parameters**

in	<i>array</i>	The input array
in	<i>n</i>	The input array length

**Returns**

A pointer to the created Segment Tree

**4.6.1.3 SegTree\_destroy()**

```
void SegTree_destroy (
    Cell * segtree )
```

Destroys the given Segment Tree.

Complexity:  $O(1)$

**Parameters**

<i>segtree</i>	The segtree
----------------	-------------

**4.6.1.4 SegTree\_query()**

```
Cell SegTree_query (
    Cell * segtree,
    int n,
    int start,
    int end )
```

Interface for the recursive range query.

Complexity: The complexity here is related to the complexity of the SegTree\_rangeQuery function, which is  $O(\log N)$ , being  $N$  the number of nodes of the tree.

The number of nodes  $N$ , in the best case is  $2 \cdot n - 1$ , with  $n$  being the input array length. In worst case  $N$  will not pass from  $4 \cdot n - 1$ . The full explanation for this is on the SegTree\_size function description.

With  $2 \cdot n - 1$  or  $4 \cdot n - 1$ , the complexity of the function is  $O(\log n)$ .

**Parameters**

	<i>segtree</i>	The segtree
in	<i>n</i>	The input array length
in	<i>start</i>	The start of the interval (1-indexed, inclusive)
in	<i>end</i>	The end of the interval (1-indexed, inclusive)

**Returns**

The results of the query

**4.6.1.5 SegTree\_rangeQuery()**

```
Cell SegTree_rangeQuery (
    Cell * segtree,
    int pos,
    int start,
    int end,
    int currStart,
    int currEnd )
```

Perform a range query on the Segment Tree recursively.

Complexity:



In the best case, the range of the query will be from 0 to (L - 1), being L the length of the input array. In this case, the complexity for retrieving data from the segtree is  $O(1)$ .

In the worst case, the range of the query will have length of 1 (query of 1 element). In this case, the algorithm will dive into the tree til reach the correspondant leaf. The complexity for retrieving the leaf data will be of  $O(H)$ , with H begin the height of the tree.

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond  $\log N$ , being N the number of nodes. So the complexity for retrieving data from a leaf is  $O(\log N)$ .

#### Parameters

	<i>segtree</i>	The segtree
in	<i>pos</i>	The position
in	<i>start</i>	The start of the interval (0-indexed, inclusive)
in	<i>end</i>	The end of the interval (0-indexed, inclusive)
in	<i>currStart</i>	The current start of the interval (0-indexed, inclusive)
in	<i>currEnd</i>	The current end of the interval (0-indexed, inclusive)

#### Returns

The results of the query

#### 4.6.1.6 SegTree\_rangeUpdate()

```
void SegTree_rangeUpdate (
    Cell * segtree,
    int pos,
    int start,
    int end,
    int currStart,
    int currEnd,
    int(*) (int n) transform )
```

Recursively updates the Segment Tree.

Complexity:

As the updates are realized on the leafs of the tree, and then the nodes that represent the intervals have to be updated, the cost on the update will be related to the amount of nodes that has to be updated plus the height of the tree. The cost to reach a leaf is  $\log H$ , being H the height of the tree. Let LF the number of leafs to be updated, that is the same as the size of the interval to be updated (end-start-1).

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond  $\log N$ , being N the number of nodes. So the complexity for retrieving data from a leaf is  $O(\log N)$ .

The complexity of a update is  $O(LF \cdot \log N)$ .

#### Parameters

	<i>segtree</i>	The segtree
in	<i>pos</i>	The current position on the segtree array
in	<i>start</i>	The start of the interval (0-indexed, inclusive)
in	<i>end</i>	The end of the interval (0-indexed, inclusive)
in	<i>currStart</i>	The current start of the interval (0-indexed, inclusive)
in	<i>currEnd</i>	The current end of the interval (0-indexed, inclusive)

#### 4.6.1.7 SegTree\_size()

```
int SegTree_size (
    int n )
```

Calculates the Segment Tree's number of nodes with the given the input array size.

The number of nodes  $n$  required for a segtree is calculated with the length of the input array. In the best case, the length  $L$  is a power of 2, so the number of nodes is  $n=2\cdot L-1$ . If  $L$  is not a power of 2, we have to find the next power of 2 after  $L$ . In this case, the number of nodes will not pass from  $n=2\cdot 2^{\lceil \log L \rceil} - 1$ , which can be simplified to  $n=4\cdot L-1$ .

Being  $n$  the input array length, let  $np$  the power of 2 that is equal or higher than  $n$ . The size of the Segment Tree array will be  $2\cdot np-1$ .

The strategy adopted here is to calculate the  $\log_2$  of  $n$  and get the ceil of the result. This way, we can power 2 for the result and get the next power of 2. Then, we use  $2\cdot (2^{\lceil \log n \rceil})-1$  to get the the size of the Segment Tree array.

Complexity: (1)

##### Parameters

in	$n$	The input array size
----	-----	----------------------

##### Returns

The size of the Segment Tree array

#### 4.6.1.8 SegTree\_update()

```
void SegTree_update (
    Cell * segtree,
    int n,
    int start,
    int end,
    int(*) (int n) transform )
```

Interface for the recursive segtree update.

Complexity: The complexity here is related to the complexity of the SegTree\_rangeUpdate function, which is  $O(L \leftrightarrow F \cdot \log N)$ , being  $N$  the number of nodes of the tree, and  $LF$  is  $end - start - 1$ .

The number of nodes  $N$ , in the best case is  $2\cdot n-1$ , with  $n$  being the input array length. In worst case  $N$  will not pass from  $4\cdot n-1$ . The full explanation for this is on the SegTree\_size function description.

With  $2\cdot n-1$  or  $4\cdot n-1$ , the complexity of the function is  $O(LF \cdot \log n)$ .

## Parameters

	<i>segtree</i>	The segtree
in	<i>n</i>	The length of the input array
in	<i>start</i>	The start of the interval (1-indexed, inclusive)
in	<i>end</i>	The end of the interval (1-indexed, inclusive)
in	<i>transform</i>	The transform function

## 4.7 src/lib/segtree.h File Reference

```
#include "cell.h"
```

## Functions

- void [SegTree\\_construct](#) ([Cell](#) \*segtree, int pos, int \*array, int start, int end)  
*Recursively constructs the Segment Tree.*
- int [SegTree\\_size](#) (int n)  
*Calculates the Segment Tree's number of nodes with the given the input array size.*
- [Cell](#) \* [SegTree\\_create](#) (int \*array, int n)  
*Creates a new Segment Tree.*
- void [SegTree\\_destroy](#) ([Cell](#) \*segtree)  
*Destroys the given Segment Tree.*
- [Cell](#) [SegTree\\_rangeQuery](#) ([Cell](#) \*segtree, int pos, int start, int end, int currStart, int currEnd)  
*Perform a range query on the Segment Tree recursively.*
- [Cell](#) [SegTree\\_query](#) ([Cell](#) \*segtree, int n, int start, int end)  
*Interface for the recursive range query.*
- void [SegTree\\_rangeUpdate](#) ([Cell](#) \*segtree, int pos, int start, int end, int currStart, int currEnd, int(\*transform)(int n))  
*Recursively updates the Segment Tree.*
- void [SegTree\\_update](#) ([Cell](#) \*segtree, int n, int start, int end, int(\*transform)(int n))  
*Interface for the recursive segtree update.*

## 4.7.1 Function Documentation

## 4.7.1.1 SegTree\_construct()

```
void SegTree_construct (
    Cell * segtree,
    int pos,
    int * array,
    int start,
    int end )
```

Recursively constructs the Segment Tree.

Complexity:

With n being the Segment Tree's number of nodes, the complexity is  $O(n)$ , because we need to visit every node of the tree to calculate its data.

**Parameters**

	<i>segtree</i>	The Segment Tree array
in	<i>pos</i>	The current position on the segtree
in	<i>array</i>	The input array
in	<i>start</i>	The start of the interval (0-indexed, inclusive)
in	<i>end</i>	The end of the interval (0-indexed, inclusive)

**4.7.1.2 SegTree\_create()**

```
Cell* SegTree_create (
    int * array,
    int n )
```

Creates a new Segment Tree.

Complexity:

The complexity for creating a new Segment Tree is related to the function SegTree\_construct, which is  $O(N)$ , being  $N$  the number of nodes of the tree.

The number of nodes  $N$ , in the best case is  $2 \cdot n - 1$ , with  $n$  being the input array length. In worst case  $N$  will not pass from  $4 \cdot n - 1$ . The full explanation for this is on the SegTree\_size function description.

In all cases, the complexity for creating a new Segment Tree is  $O(n)$ .

**Parameters**

in	<i>array</i>	The input array
in	<i>n</i>	The input array length

**Returns**

A pointer to the created Segment Tree

**4.7.1.3 SegTree\_destroy()**

```
void SegTree_destroy (
    Cell * segtree )
```

Destroys the given Segment Tree.

Complexity:  $O(1)$

## Parameters

<i>segtree</i>	The segtree
----------------	-------------

## 4.7.1.4 SegTree\_query()

```
Cell SegTree_query (
    Cell * segtree,
    int n,
    int start,
    int end )
```

Interface for the recursive range query.

Complexity: The complexity here is related to the complexity of the SegTree\_rangeQuery function, which is  $O(\log N)$ , being  $N$  the number of nodes of the tree.

The number of nodes  $N$ , in the best case is  $2 \cdot n - 1$ , with  $n$  being the input array length. In worst case  $N$  will not pass from  $4 \cdot n - 1$ . The full explanation for this is on the SegTree\_size function description.

With  $2 \cdot n - 1$  or  $4 \cdot n - 1$ , the complexity of the function is  $O(\log n)$ .

## Parameters

	<i>segtree</i>	The segtree
in	<i>n</i>	The input array length
in	<i>start</i>	The start of the interval (1-indexed, inclusive)
in	<i>end</i>	The end of the interval (1-indexed, inclusive)

## Returns

The results of the query

## 4.7.1.5 SegTree\_rangeQuery()

```
Cell SegTree_rangeQuery (
    Cell * segtree,
    int pos,
    int start,
    int end,
    int currStart,
    int currEnd )
```

Perform a range query on the Segment Tree recursively.

Complexity:

In the best case, the range of the query will be from 0 to (L - 1), being L the length of the input array. In this case, the complexity for retrieving data from the segtree is  $O(1)$ .

In the worst case, the range of the query will have length of 1 (query of 1 element). In this case, the algorithm will dive into the tree til reach the correspondant leaf. The complexity for retrieving the leaf data will be of  $O(H)$ , with H begin the height of the tree.

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond  $\log N$ , being N the number of nodes. So the complexity for retrieving data from a leaf is  $O(\log N)$ .

#### Parameters

	<i>segtree</i>	The segtree
in	<i>pos</i>	The position
in	<i>start</i>	The start of the interval (0-indexed, inclusive)
in	<i>end</i>	The end of the interval (0-indexed, inclusive)
in	<i>currStart</i>	The current start of the interval (0-indexed, inclusive)
in	<i>currEnd</i>	The current end of the interval (0-indexed, inclusive)

#### Returns

The results of the query

#### 4.7.1.6 SegTree\_rangeUpdate()

```
void SegTree_rangeUpdate (
    Cell * segtree,
    int pos,
    int start,
    int end,
    int currStart,
    int currEnd,
    int(*) (int n) transform )
```

Recursively updates the Segment Tree.

Complexity:

As the updates are realized on the leafs of the tree, and then the nodes that represent the intervals have to be updated, the cost on the update will be related to the amount of nodes that has to be updated plus the height of the tree. The cost to reach a leaf is  $\log H$ , being H the height of the tree. Let LF the number of leafs to be updated, that is the same as the size of the interval to be updated (end-start-1).

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond  $\log N$ , being N the number of nodes. So the complexity for retrieving data from a leaf is  $O(\log N)$ .

The complexity of a update is  $O(LF \cdot \log N)$ .

#### Parameters

	<i>segtree</i>	The segtree
in	<i>pos</i>	The current position on the segtree array
in	<i>start</i>	The start of the interval (0-indexed, inclusive)
in	<i>end</i>	The end of the interval (0-indexed, inclusive)
in	<i>currStart</i>	The current start of the interval (0-indexed, inclusive)
in	<i>currEnd</i>	The current end of the interval (0-indexed, inclusive)

## 4.7.1.7 SegTree\_size()

```
int SegTree_size (
    int n )
```

Calculates the Segment Tree's number of nodes with the given the input array size.

The number of nodes  $n$  required for a segtree is calculated with the length of the input array. In the best case, the length  $L$  is a power of 2, so the number of nodes is  $n=2\cdot L-1$ . If  $L$  is not a power of 2, we have to find the next power of 2 after  $L$ . In this case, the number of nodes will not pass from  $n=2\cdot 2^{\lceil \log L \rceil} - 1$ , which can be simplified to  $n=4\cdot L-1$ .

Being  $n$  the input array length, let  $np$  the power of 2 that is equal or higher than  $n$ . The size of the Segment Tree array will be  $2\cdot np-1$ .

The strategy adopted here is to calculate the  $\log_2$  of  $n$  and get the ceil of the result. This way, we can power 2 for the result and get the next power of 2. Then, we use  $2\cdot (2^{\lceil \log n \rceil})-1$  to get the the size of the Segment Tree array.

Complexity: (1)

## Parameters

in	$n$	The input array size
----	-----	----------------------

## Returns

The size of the Segment Tree array

## 4.7.1.8 SegTree\_update()

```
void SegTree_update (
    Cell * segtree,
    int n,
    int start,
    int end,
    int (*) (int n) transform )
```

Interface for the recursive segtree update.

Complexity: The complexity here is related to the complexity of the SegTree\_rangeUpdate function, which is  $O(L \leftrightarrow F \cdot \log N)$ , being  $N$  the number of nodes of the tree, and  $LF$  is  $end - start - 1$ .

The number of nodes  $N$ , in the best case is  $2\cdot n-1$ , with  $n$  being the input array length. In worst case  $N$  will not pass from  $4\cdot n-1$ . The full explanation for this is on the SegTree\_size function description.

With  $2\cdot n-1$  or  $4\cdot n-1$ , the complexity of the function is  $O(LF \cdot \log n)$ .

**Parameters**

	<i>segtree</i>	The segtree
in	<i>n</i>	The length of the input array
in	<i>start</i>	The start of the interval (1-indexed, inclusive)
in	<i>end</i>	The end of the interval (1-indexed, inclusive)
in	<i>transform</i>	The transform function

## 4.8 src/matriz.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib/matrix.h"
#include "lib/cell.h"
```

**Functions**

- int [main](#) ()  
*Main function.*

### 4.8.1 Function Documentation

#### 4.8.1.1 main()

```
int main ( )
```

Main function.

This function uses the Nubby solution based on a matrix that stores the data for each possible interval.

Complexity:  $O(n^2)$

**Returns**

0



# Index

add  
    arvore.c, 7

arvore.c  
    add, 7  
    main, 7  
    sub, 8

Cell  
    cell.h, 10

cell, 5  
    max, 5  
    min, 5  
    sum, 5

cell.c  
    Cell\_create, 8  
    Cell\_destroy, 9  
    Cell\_fill, 9  
    Cell\_print, 9

cell.h  
    Cell, 10  
    Cell\_create, 11  
    Cell\_destroy, 11  
    Cell\_fill, 11  
    Cell\_print, 12

Cell\_create  
    cell.c, 8  
    cell.h, 11

Cell\_destroy  
    cell.c, 9  
    cell.h, 11

Cell\_fill  
    cell.c, 9  
    cell.h, 11

Cell\_print  
    cell.c, 9  
    cell.h, 12

main  
    arvore.c, 7  
    matriz.c, 26

matrix.c  
    Matrix\_create, 12  
    Matrix\_destroy, 13  
    Matrix\_fill, 13  
    Matrix\_print, 13

matrix.h  
    Matrix\_create, 14  
    Matrix\_destroy, 15  
    Matrix\_fill, 15  
    Matrix\_print, 15

Matrix\_create  
    matrix.c, 12  
    matrix.h, 14

Matrix\_destroy  
    matrix.c, 13  
    matrix.h, 15

Matrix\_fill  
    matrix.c, 13  
    matrix.h, 15

Matrix\_print  
    matrix.c, 13  
    matrix.h, 15

matriz.c  
    main, 26

max  
    cell, 5

min  
    cell, 5

SegTree\_construct  
    segtree.c, 16  
    segtree.h, 21

SegTree\_create  
    segtree.c, 17  
    segtree.h, 22

SegTree\_destroy  
    segtree.c, 17  
    segtree.h, 22

SegTree\_query  
    segtree.c, 18  
    segtree.h, 23

SegTree\_rangeQuery  
    segtree.c, 18  
    segtree.h, 23

SegTree\_rangeUpdate  
    segtree.c, 19  
    segtree.h, 24

SegTree\_size  
    segtree.c, 20  
    segtree.h, 25

SegTree\_update  
    segtree.c, 20  
    segtree.h, 25

segtree.c  
    SegTree\_construct, 16  
    SegTree\_create, 17  
    SegTree\_destroy, 17  
    SegTree\_query, 18  
    SegTree\_rangeQuery, 18  
    SegTree\_rangeUpdate, 19

- SegTree\_size, [20](#)
- SegTree\_update, [20](#)
- segtree.h
  - SegTree\_construct, [21](#)
  - SegTree\_create, [22](#)
  - SegTree\_destroy, [22](#)
  - SegTree\_query, [23](#)
  - SegTree\_rangeQuery, [23](#)
  - SegTree\_rangeUpdate, [24](#)
  - SegTree\_size, [25](#)
  - SegTree\_update, [25](#)
- src/arvore.c, [7](#)
- src/lib/cell.c, [8](#)
- src/lib/cell.h, [10](#)
- src/lib/matrix.c, [12](#)
- src/lib/matrix.h, [14](#)
- src/lib/segtree.c, [16](#)
- src/lib/segtree.h, [21](#)
- src/matriz.c, [26](#)
- sub
  - arvore.c, [8](#)
- sum
  - cell, [5](#)