Mensuração de Ordenação

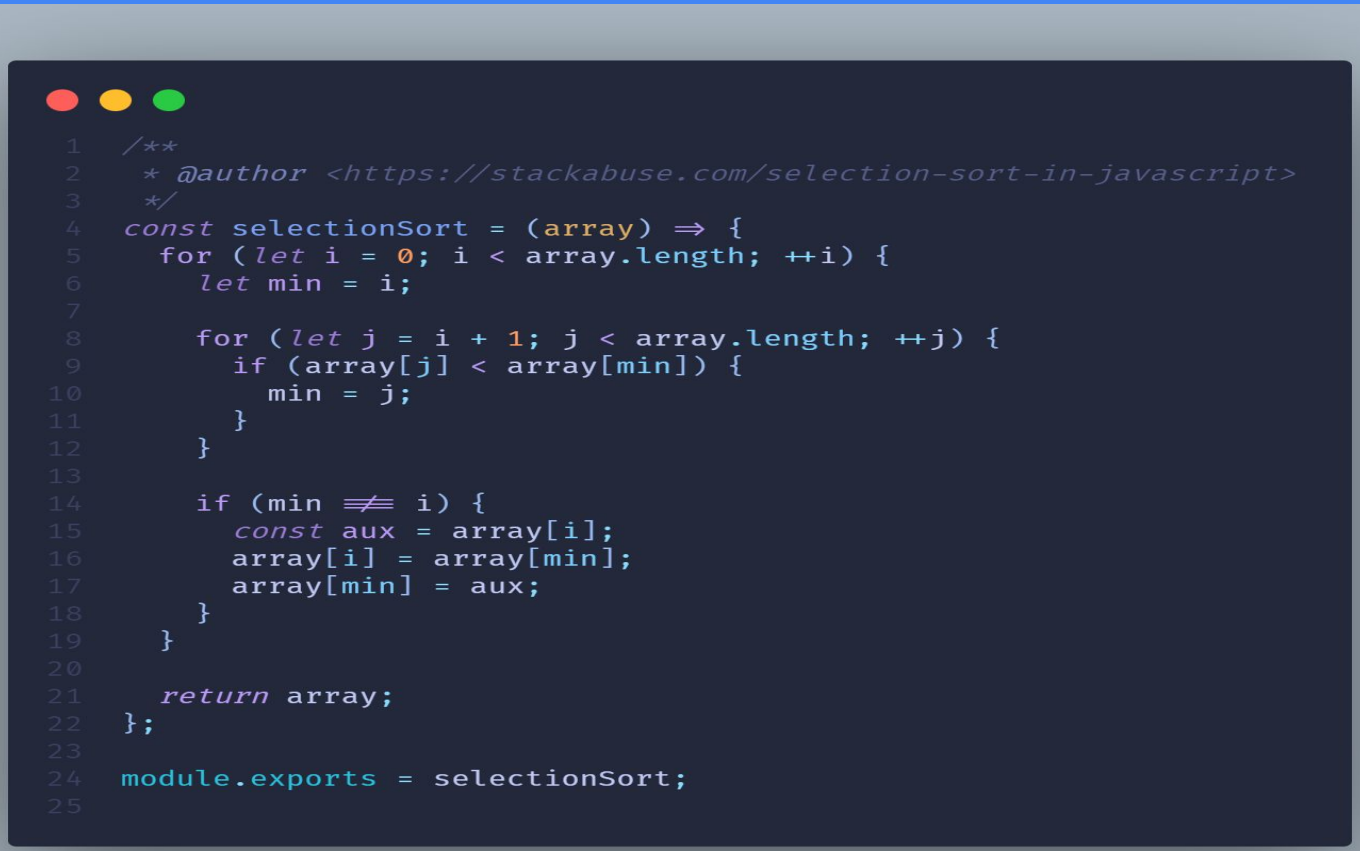
Trabalho de Estruturas de Dados

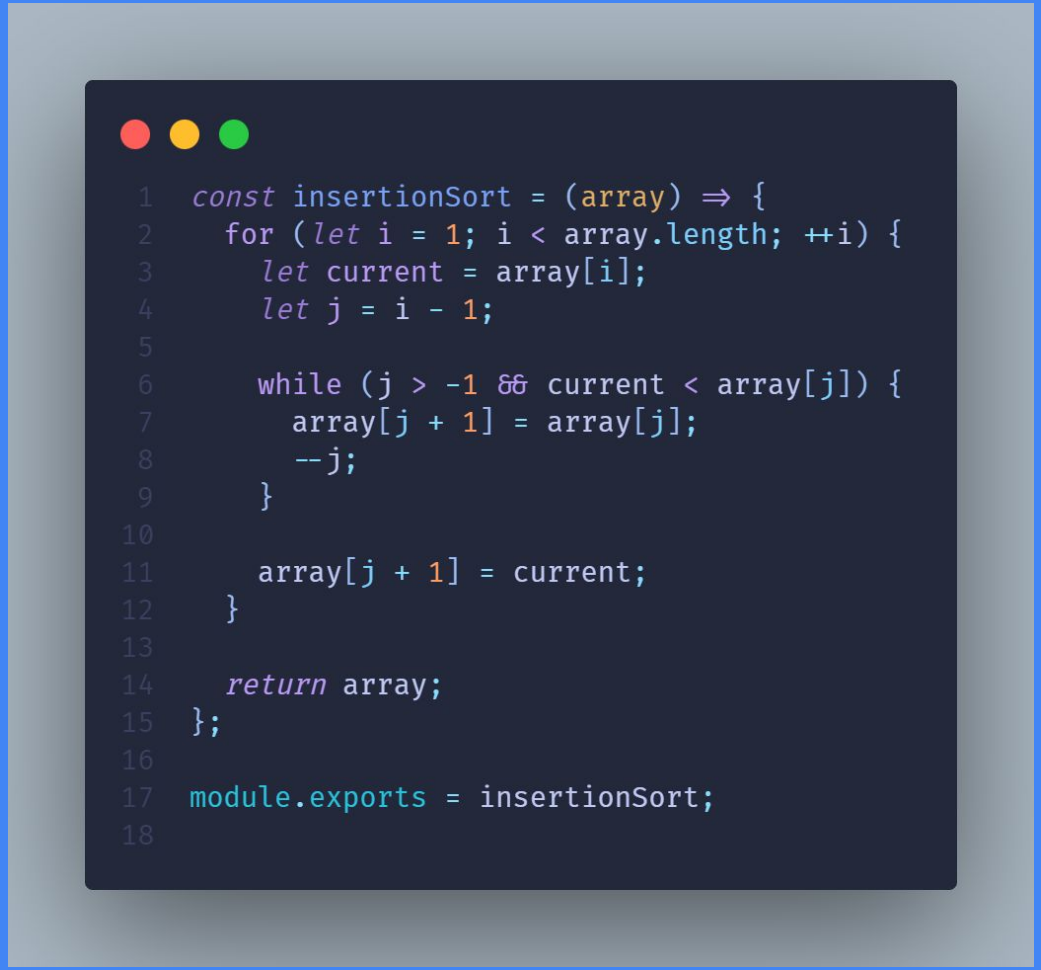
Alunos: Diego dos Santos Fernandes
Ueslei Albuquerque Garcia

```
const bubbleSort = (array) => {
  for (let i = 0; i < array.length; ++i) {
    for (let j = 0; j < array.length - 1; ++j) {
      if (array[j] > array[j + 1]) {
        const aux = array[j];
        array[j] = array[j + 1];
        array[j + 1] = aux;
      }
    }
  }

  return array;
};

module.exports = bubbleSort;
```


/**
* @author <https://stackabuse.com/selection-sort-in-javascript>
*/
const selectionSort = (array) => {
for (let i = 0; i < array.length; ++i) {
let min = i;
for (let j = i + 1; j < array.length; ++j) {
if (array[j] < array[min]) {
min = j;
}
}
if (min !== i) {
const aux = array[i];
array[i] = array[min];
array[min] = aux;
}
}
return array;
};
module.exports = selectionSort;


const insertionSort = (array) ⇒ {
for (let i = 1; i < array.length; ++i) {
let current = array[i];
let j = i - 1;
while (j > -1 && current < array[j]) {
array[j + 1] = array[j];
--j;
}
array[j + 1] = current;
}
return array;
};
module.exports = insertionSort;

```
const merge = (left, right) => {
  let array = [];

  while (left.length && right.length) {
    if (left[0] < right[0]) {
      array.push(left.shift());
    } else {
      array.push(right.shift());
    }
  }

  return [ ...array,  ...left,  ...right];
};

const mergeSort = (array) => {
  const half = array.length / 2;

  if (array.length < 2) {
    return array;
  }

  const left = array.splice(0, half);
  return merge(mergeSort(left), mergeSort(array));
};

module.exports = mergeSort;
```

```
const partition = (array, left, right) => {
  const pivot = array[Math.floor((right + left) / 2)];
  let i = left;
  let j = right;

  while (i <= j) {
    while (array[i] < pivot) {
      ++i;
    }

    while (array[j] > pivot) {
      --j;
    }

    if (i <= j) {
      const aux = array[i];
      array[i] = array[j];
      array[j] = aux;
      ++i;
      --j;
    }
  }

  return i;
};

const quickSort = (
  array,
  left = 0,
  right = array.length - 1
) => {
  if (array.length > 1) {
    const index = partition(array, left, right);

    if (left < index - 1) {
      quickSort(array, left, index - 1);
    }

    if (index < right) {
      quickSort(array, index, right);
    }
  }

  return array;
};

module.exports = quickSort;
```

```
class MaxHeap {
  constructor() {
    this._heap = [];
  }

  get heap() {
    return this._heap;
  }

  set heap(value) {
    this._heap = value;
  }

  parentIndex(index) {
    return Math.floor((index - 1) / 2);
  }

  leftChildIndex(index) {
    return 2 * index + 1;
  }

  rightChildIndex(index) {
    return 2 * index + 2;
  }

  swap(a, b) {
    const aux = this.heap[a];
    this.heap[a] = this.heap[b];
    this.heap[b] = aux;
  }

  insert(item) {
    this.heap.push(item);

    let index = this.heap.length - 1;
    let parent = this.parentIndex(index);

    while (this.heap[parent] && this.heap[parent] < this.heap[index]) {
      this.swap(parent, index);

      index = this.parentIndex(index);
      parent = this.parentIndex(index);
    }
  }

  delete() {
```

```
  delete() {
    const item = this.heap.shift();

    this.heap.unshift(this.heap.pop());

    let index = 0;
    let leftChild = this.leftChildIndex(index);
    let rightChild = this.rightChildIndex(index);

    while (
      (this.heap[leftChild] && this.heap[leftChild] > this.heap[index]) ||
      this.heap[rightChild] > this.heap[index]
    ) {
      let max = leftChild;

      if (this.heap[rightChild] && this.heap[rightChild] > this.heap[max]) {
        max = rightChild;
      }

      this.swap(max, index);

      index = max;
      leftChild = this.leftChildIndex(max);
      rightChild = this.rightChildIndex(max);
    }

    return item;
  }
}

const heapSort = (unorderedArray) => {
  const sortedArray = [];
  const heap = new MaxHeap();

  for (let i = 0; i < unorderedArray.length; ++i) {
    heap.insert(unorderedArray[i]);
  }

  for (let i = 0; i < unorderedArray.length; ++i) {
    sortedArray.push(heap.delete());
  }

  return sortedArray;
};

module.exports = heapSort;
```


const path = require('path');
const fs = require('fs');
const readArray = (fileName) => {
return fs.readFileSync(
path.join(__dirname, '..', 'data', fileName),
'utf-8'
)
.split('\n')
.map((value) => Number(value));
}
module.exports = readArray;

```
const readArray = require('./src/functions/readArray');
const bubbleSort = require('./src/functions/bubbleSort');
const selectionSort = require('./src/functions/selectionSort');
const insertionSort = require('./src/functions/insertionSort');
const mergeSort = require('./src/functions/mergeSort');
const quickSort = require('./src/functions/quickSort');
const heapSort = require('./src/functions/heapSort');

const sortingMethods = [
  bubbleSort,
  selectionSort,
  insertionSort,
  mergeSort,
  quickSort,
  heapSort
];

let entry = undefined;
let loop = undefined;

for (const sortingMethod of sortingMethods) {
  for (let i = entry || 1; i <= 4; ++i) {
    const fileName = `entrada${i}.txt`;

    for (let j = loop || 1; j <= 5; ++j) {
      const label = `${sortingMethod.name}, entry No. ${i}, loop No. ${j}`;
      const array = readArray(fileName);

      console.time(label);
      sortingMethod(array);
      console.timeEnd(label);
      console.log(new Date().toLocaleTimeString());
      console.log('-'.repeat(label.length));
    }

    loop = undefined;
  }

  entry = undefined;
}
```

```
const path = require('path');
const fs = require('fs');

const sortingMethods = {
  bubbleSort: 'Método da bolha',
  selectionSort: 'Método de ordenação por seleção',
  insertionSort: 'Método de ordenação por inserção',
  mergeSort: 'Método de ordenação por intercalação',
  quickSort: 'Método de ordenação por separação',
  heapSort: 'Método de ordenação por monte'
};

const createSheets = () => {
  let filePath = path.join(__dirname, '..', 'data', 'result.txt');
  const runTimes = fs.readFileSync(filePath, 'utf-8').split('\n');
  const header = [
    'Método de ordenação',
    ...['primeira', 'segunda', 'terceira', 'quarta', 'quinta'].map(value => `Tempo da ${value} execução (ms)`),
    'Média aritmética'
  ].map(value => `"${value}"`).join(',') + '\n';
  const map = new Map();
  runTimes.forEach(runtime => {
    const runtimeArray = runtime.split(' ');
    const sortingMethod = runtimeArray[0].slice(0, -1);
    const entry = Number(runtimeArray[3].slice(0, -1));
    const loop = Number(runtimeArray[6].slice(0, -1));
    const label = { sortingMethod, entry, loop };
    let value = 0;
    if (runtimeArray.length === 9) {
      const [milliseconds, seconds, minutes, hours] = runtimeArray[7].split(/[:\.]/).reverse().map(value => Number(value));
      if (hours !== undefined) value += hours * 60 * 60 * 1000;
      value += minutes * 60 * 1000;
      value += seconds * 1000;
      value += milliseconds;
    } else {
      value = Number(runtimeArray[7].slice(0, -2));
    }
    map.set(JSON.stringify(label), value);
  });
  const sheets = Array(4).fill(header);
  for (let i = 0; i < 4; ++i) {
    for (const rawSortingMethod of Object.keys(sortingMethods)) {
      const sortingMethod = sortingMethods[rawSortingMethod];
      let average = 0;
      sheets[i] += `"${sortingMethod}",`;
      for (let j = 1; j <= 5; ++j) {
        const label = { sortingMethod: rawSortingMethod, entry: (i + 1), loop: j };
        const value = map.get(JSON.stringify(label));
        sheets[i] += `"${value.toLocaleString()}",`;
        average += value;
      }
      average /= 5;
      sheets[i] += `"${Number(average.toFixed(3)).toLocaleString()}"\n`;
    }
    filePath = path.join(__dirname, '..', 'sheets', `results-over-entry-${(i + 1)}.csv`);
    fs.writeFileSync(filePath, sheets[i]);
  }
}

createSheets();
```


Ordenação da primeira entrada
Média aritmética dos tempos de execução
5000000
4000000
3000000
2000000
1000000
0
Bubble(Bolha)
Selection(Seleção)
Inserction(Inserção)
Merge(Intercalação)
Quick(Rápida)
Heap(Pilha)
Método de ordenação
Média aritmética


Ordenação da segunta entrada
1000000
800000
600000
400000
200000
0
Média aritmética dos tempos de execução
Bubble(Bolha)
Selection(Seleção)
Insertion(Inserção)
Merge(Intercalação)
Quick(Rápida)
Heap(Pilha)
Método de ordenação
Média aritmética
Área do gráfico


Ordenação da terceira entrada
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
Média aritmética dos tempos de execução
Bubble(Bolha)
Selection(Seleção)
Inserction(Inserção)
Merge(Intercalação)
Quick(Rápida)
Heap(Pilha)
Método de ordenação
Média aritmética


Ordenação da quarta entrada
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
Média aritmética dos tempos de execução
Bubble(Bolha)
Selection(Seleção)
Inserction(Inserção)
Merge(Intercalação)
Quick(Rápida)
Heap(Pilha)
Método de ordenação
Média aritmética