Algoritmo

O Algoritmo é um esquema de resolução de um problema. Pode ser implementado com qualquer sequência de valores ou objetos que tenham uma lógica infinita (por exemplo, a língua portuguesa, a linguagem Pascal, a linguagem C, uma sequência numérica, um conjunto de objetos tais como lápis e borracha), ou seja, qualquer coisa que possa fornecer uma sequência lógica.

Podemos ilustrar um algoritmo pelo exemplo de uma receita culinária, embora muitos algoritmos sejam mais complexos. Um Algoritmo mostra passo a passo os procedimentos necessários para resolução de um problema.

Bubble Sort

Bubble Sort é o algoritmo mais simples, mas o menos eficiente. Neste algoritmo cada elemento da posição i será comparado com o elemento da posição i + 1, ou seja, um elemento da posição 2 será comparado com o elemento da posição 3. Caso o elemento da posição 2 for maior que o da posição 3, eles trocam de lugar e assim sucessivamente. Por causa dessa forma de execução, o vetor terá que ser percorrido quantas vezes que for necessária, tornando o algoritmo ineficiente para listas muito grandes.

Selection Sort

Este algoritmo é baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o segundo menor valor para a segunda posição e assim sucessivamente, até os últimos dois elementos.

Neste algoritmo de ordenação é escolhido um número a partir do primeiro, este número escolhido é comparado com os números a partir da sua direita, quando encontrado um número menor, o número escolhido ocupa a posição do menor número encontrado. Este número encontrado será o próximo número escolhido, caso não for encontrado nenhum número menor que este escolhido, ele é colocado na posição do primeiro número escolhido, e o próximo número à sua direita vai ser o escolhido para fazer as comparações. É repetido esse processo até que a lista esteja ordenada.

Insertion Sort

O Insertion Sort é um algoritmo simples e eficiente quando aplicado em pequenas listas. Neste algoritmo a lista é percorrida da esquerda para a direita, à medida que avança vai deixando os elementos mais à esquerda ordenados.

O algoritmo funciona da mesma forma que as pessoas usam para ordenar cartas em um jogo de baralho como o pôquer.

Quick Sort

O Quick Sort é o algoritmo mais eficiente na ordenação por comparação. Nele se escolhe um elemento chamado de pivô, a partir disto é organizada a lista para que todos os números anteriores a ele sejam menores que ele, e todos os números posteriores a ele sejam maiores que ele. Ao final desse processo o número pivô já está em sua posição final. Os dois grupos desordenados recursivamente sofreram o mesmo processo até que a lista esteja ordenada.

Merge Sort

Esse algoritmo divide o problema em pedaços menores, resolve cada pedaço e depois junta (merge) os resultados. O vetor será dividido em duas partes iguais, que serão cada uma dividida em duas partes, e assim até ficar um ou dois elementos cuja ordenação é trivial.

Para juntar as partes ordenadas os dois elementos de cada parte são separados e o menor deles é selecionado e retirado de sua parte. Em seguida os menores entre os restantes são comparados e assim se prossegue até juntar as partes.

Implementação

Vetores:

```
let Crescente = []
for (let i = 1; i <= 100000; ++i) {
   Crescente.push(i)
}

let Decrescente = []
for (let i = 100000; i >= 1; --i) {
   Decrescente.push(i)
}

let Random = []
function getRandom(min, max) {
   return Math.floor(Math.random() * (max - min + 1)) + min;
}
for (let i = 1; i <= 100000; ++i) {
   Random.push(getRandom(1, 100000))
}
```

```
const algoritmoSort  = function() {
    let algoritmoCI = new Date().getTime()
    algoritmoOrdenacao(algoritmoCrescente)
    let algoritmoCE = new Date().getTime()
    let algoritmoCF = algoritmoCE - algoritmoCI
    console.log(`Vetor Crescente em Algoritmo Sort levou: ${algoritmoCF}ms`)
    document.getElementById("algoritmoC").value = algoritmoCF

    let algoritmoDI = new Date().getTime()
    algoritmoOrdenacao(algoritmoDecrescente)
    let algoritmoDE = new Date().getTime()
    let algoritmoDF = algoritmoDE - algoritmoDI
    console.log(`Vetor Decrescente em Algoritmo Sort levou: ${algoritmoDF}ms`)
    document.getElementById("algoritmoD").value = algoritmoDF

    let algoritmoRI = new Date().getTime()
    algoritmoOrdenacao(algoritmoRandom)
    let algoritmoRE = new Date().getTime()
    let algoritmoRF = algoritmoRE - algoritmoRI
    console.log(`Vetor Random em algoritmo Sort levou: ${algoritmoRF}ms`)
    document.getElementById("algoritmoR").value = algoritmoRF
    alert ("Algoritmo Sort já finalizou!")
}
```

```
function selectionOrdenacao(arr) {

    let temp = 0;
    for (let i = 0; i < arr.length; ++i) {
        for (let j = i + 1; j < arr.length; ++j) {
            if (arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    return (arr);
}


function quickOrdenacao(array) {
    if (array.length <= 1) {
        return array;
    }
    let pivot = array[0];
    let left = [];
    let right = [];
```

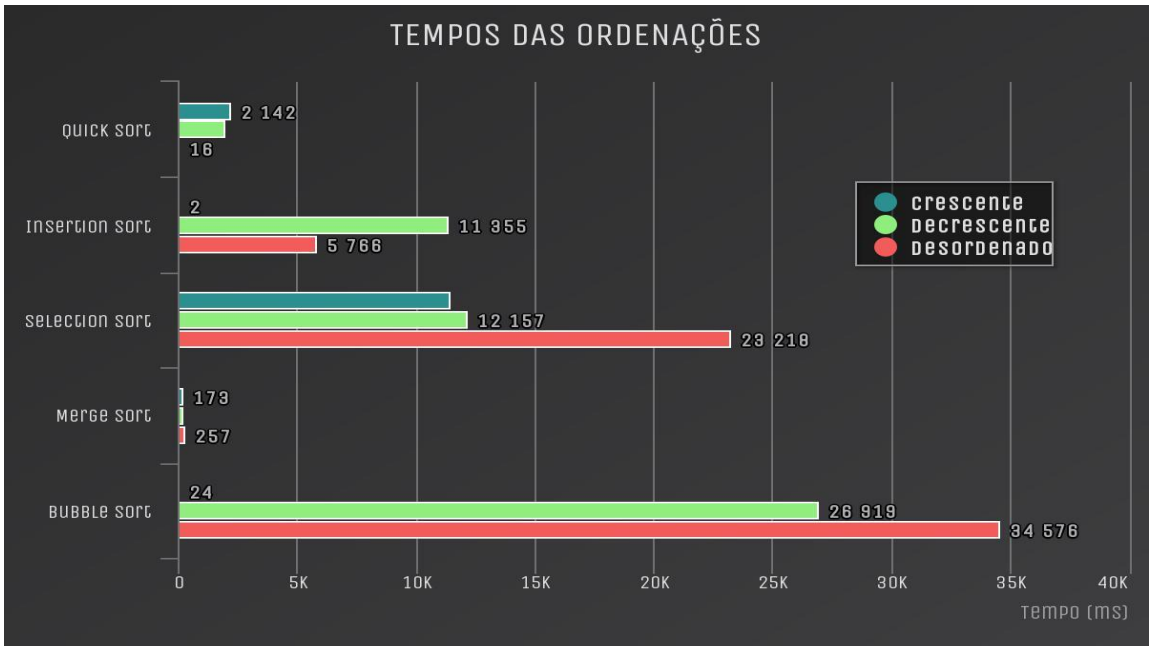
```
    for (let i = 1; i < array.length; i++) {
        array[i] < pivot ? left.push(array[i]) : right.push(array[i]);
    }
    return quickOrdenacao(left).concat(pivot, quickOrdenacao(right));
};


function mergeOrdenacao(arr) {
    if (arr.length < 2) {
        return arr;
    }
    var mid = Math.floor(arr.length / 2);
    var subLeft = mergeOrdenacao(arr.slice(0, mid));
    var subRight = mergeOrdenacao(arr.slice(mid));
    return merge(subLeft, subRight);
}
function merge(node1, node2) {
    var result = [];
    while (node1.length > 0 && node2.length > 0)
        result.push(node1[0] < node2[0] ? node1.shift() : node2.shift());
    return result.concat(node1.length ? node1 : node2);
}
```

```
function insertionOrdenacao(array) {
    var length = array.length;
    for (var i = 1, j; i < length; i++) {
        var temp = array[i];
        for (var j = i - 1; j >= 0 && array[j] > temp; j--) {
            array[j + 1] = array[j];
        }
        array[j + 1] = temp;
    }
    return array;
}

function bubbleOrdenacao(arr) {
    const sortedArray = Array.from(arr);
    let swap;
    do {
        swap = false;
        for (let i = 1; i < sortedArray.length; ++i) {
            if (sortedArray[i - 1] > sortedArray[i]) {
                [sortedArray[i], sortedArray[i - 1]] = [sortedArray[i - 1], sortedArray[i]];
                swap = true;
            }
        }
    } while (swap)
    return sortedArray;
}
```

	Quick	Merge	Selection	Bubble	Insertion
Crescente	2142ms	173ms	11366ms	24ms	2ms
Decrescente	1919ms	164ms	12157ms	26919ms	11355ms
Random	16ms	257ms	23218ms	34576ms	5766ms
	Tempo em Milisegundos (ms)				



Conclusão

Bom, foi um grande desafio fazer a conexão de tudo, acredito que obtive um resultado satisfatório (record pessoal). Consegui conciliar as matérias e algoritmo e web de uma maneira assustadora.

O algoritmo Merge, foi o que mais me impressionou, se manteve ágil nos três vetores. Por isso eu não consegui chegar a uma conclusão sobre ele. Bom, já os outros quatro, consegui entender alguns traços como por exemplo:

- Bubble Sort é um tanto quanto ineficiente se comparado aos outros.
- Selection Sort demonstrou ser um algoritmo estável, mas com um grande tempo de resposta, se comparado aos outros da lista.
- Insertion Sort é muito rápido com vetores em ordem crescente, mas o seu tempo aumenta drasticamente nos outros dois tipos de vetores.
- Quick Sort mesmo tendo o segundo maior tempo nos vetores crescentes, demonstrou ser estável e ágil nos demais vetores, alcançando a marca de segundo lugar, perdendo para o Merge que eu não consegui entender perfeitamente.

Conheci uma nova ferramenta, bibliotecas chamas Charts. São complicada de entender no início, mas bastante “ilustrativas” com a documentação das possíveis alterações que você pode fazer no seu gráfico.

Realmente esse trabalho acabou me forçou ir um pouco além, alguns estresses, principalmente por não entender muito bem JavaScript. Entretanto eu consegui absorver muita coisa da qual eu estudei para colocar esse trabalho em prática.

Conclusão

Com base nos testes realizados foram obtidas as seguintes conclusões:

Bubble sort

Para listas já ordenadas em ordem crescente é o único algoritmo que não realiza movimentações, mas em compensação é o que tem o maior tempo e o maior número de comparações. Não só em listas já ordenadas, mas em todos os casos o bubble sort se mostrou um algoritmo ineficiente.

Selection sort

Nas listas de ordem 1 e ordem 3, o selection sort foi o segundo pior algoritmo, mas se mostrou mais eficiente do que o Insertion sort em relação ao tempo e a quantidade de movimentações na lista de ordem 2.

Insertion Sort

Na lista de ordem 1, o Insertion sort se mostrou mais eficiente que todos os outros algoritmos em relação ao tempo e comparações. Na lista de ordem 2 foi menos eficiente do que o selection sort e na lista de ordem 3 a diferença de tempo entre o insertion e o selection foi pequena.

Quick Sort

O quick sort certamente é o algoritmo mais eficiente em listas totalmente desordenadas, ele se torna muito eficiente em relação aos outros no quesito de tempo. Na lista de ordem 3 e na de ordem 2 a diferença de tempo do quick sort em comparação aos outros foi absurdamente grande.