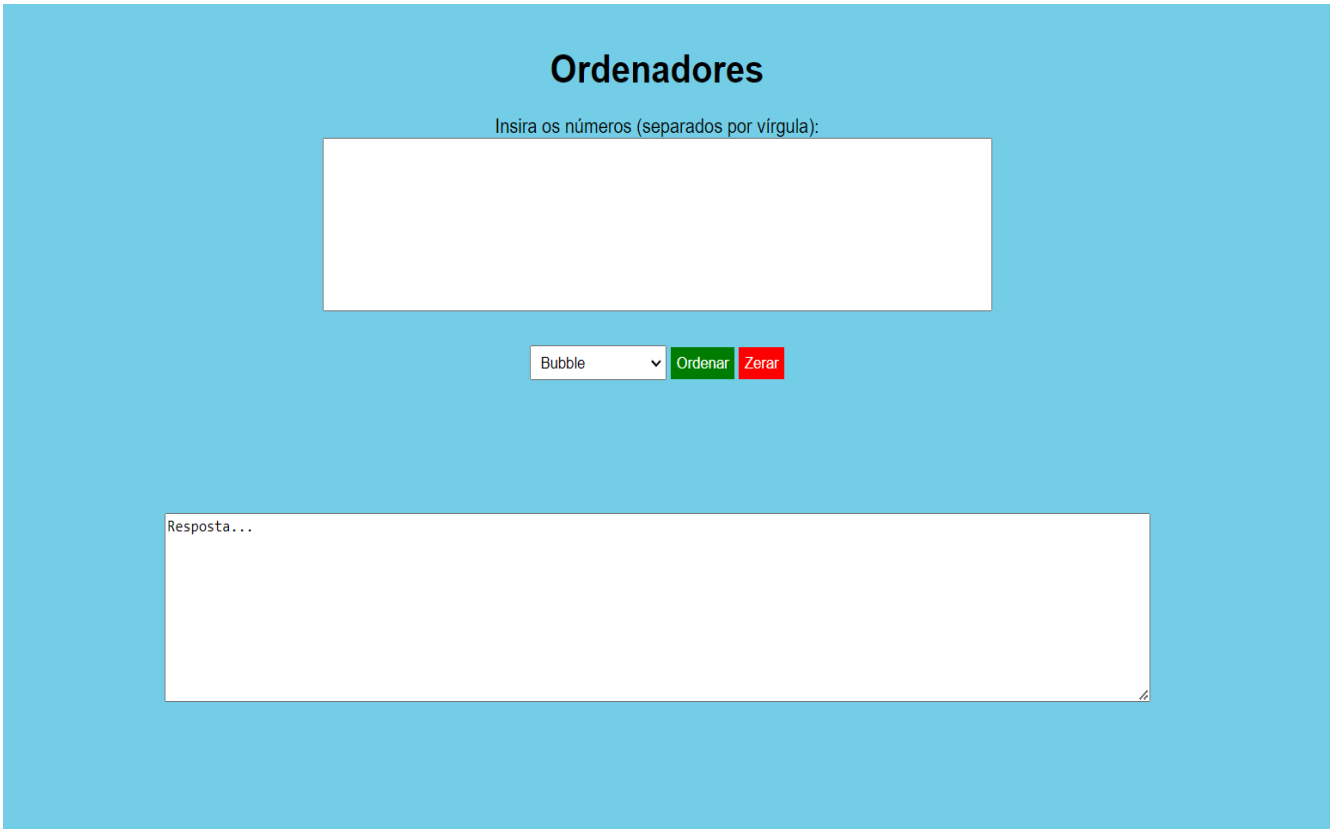
Erick Luiz D. Nossal
UNC- Universidade do Contestado, Concórdia-SC

ORDENADORES

O presente trabalho aborda a implementação de um projeto web que integra um formulário de login e uma funcionalidade de ordenação de dados, utilizando tecnologias web padrão, como HTML, CSS e JavaScript. Esta abordagem visa proporcionar aos usuários uma experiência intuitiva e eficaz ao interagir com a aplicação. Ao longo deste trabalho, exploraremos detalhadamente a construção e a implementação da funcionalidade de ordenação de dados, que permite aos usuários inserir uma lista de números e escolher entre diversos algoritmos de ordenação para organizar esses dados de maneira eficiente.



1.0 Formulário de Login (login.html)

O formulário de login permite que os usuários acessem a página de ordenação após a autenticação bem-sucedida. Abaixo está o código HTML e JavaScript responsável por esta funcionalidade.

1.1 Funcionalidade de Login

A função login() verifica se os campos de usuário e senha estão preenchidos. Se estiverem, o usuário é redirecionado para a página de ordenação (ordenadores.html). Caso contrário, uma mensagem de erro é exibida.

1.2 Verificação de Login

A função checkLogin() é executada quando a página é carregada e verifica se o usuário está autenticado. Se não estiver, o usuário é redirecionado de volta para a página de login.

2.0 Página de Ordenação (ordenadores.html)

A página de ordenação permite que o usuário insira números e escolha um algoritmo de ordenação para ordená-los.

2.1 Seleção do Algoritmo de Ordenação
A função sort() é chamada quando o botão "Ordenar" é clicado. Ela extrai os números inseridos pelo usuário, seleciona o algoritmo de ordenação escolhido a partir do menu suspenso e exibe o tempo decorrido e a lista ordenada.

2.2 Implementação dos Algoritmos de Ordenação
Os algoritmos de ordenação devem ser implementados nas respectivas funções de ordenação, como "bubbleSort", "selectionSort", "mergeSort", entre outros, dentro do bloco de switch. Cada algoritmo recebe o array de números como entrada e ordena-os de acordo com sua lógica específica.

Ordenadores utilizados

Glowworm sort: O Glowworm Sort é um algoritmo de ordenação bio-inspirado, baseado no comportamento dos vaga-lumes (glowworms) em relação à luminescência. Cada elemento da lista a ser ordenada é representado por um vaga-lume que tem uma luminescência associada ao valor do elemento. Em uma analogia simples, quanto maior o valor do elemento, maior a luminescência do correspondente vaga-lume.

Slow Sort: É um algoritmo de ordenação que foi introduzido por Robert Sedgewick como um exemplo de um algoritmo de ordenação ineficiente. Seu principal objetivo não é ser usado na prática, mas sim como uma curiosidade teórica para ilustrar princípios e técnicas de ordenação de uma maneira inusitada.

BogoSort: Também conhecido como Permutation Sort ou StupidSort, é um algoritmo de ordenação ineficiente e intencionalmente absurdo. Ele é mais uma curiosidade teórica do que um algoritmo prático para qualquer propósito de ordenação. A ideia básica do BogoSort é gerar aleatoriamente permutações do conjunto a ser ordenado até que encontre uma que esteja ordenada.

Binary Tree: A árvore binária (ou "binary tree", em inglês) é uma estrutura de dados fundamental na ciência da computação. Ela é uma estrutura hierárquica que consiste em nós, onde cada nó tem no máximo dois filhos, geralmente referidos como filho esquerdo e filho direito. As árvores binárias são usadas em diversas aplicações, incluindo a organização de dados, algoritmos de busca, algoritmos de ordenação e expressões matemáticas.

Stooge Sort: O StoogeSort é um algoritmo de ordenação recursivo e ineficiente, conhecido mais como uma curiosidade teórica do que uma ferramenta prática. Foi nomeado em referência aos comediantes "The Three Stooges" devido ao seu comportamento aparentemente cômico e ineficiente.
A ideia básica desse ordenador utiliza a troca de elementos e a divisão recursiva.

Pigeonhole Sort: O Pigeonhole Sort é um algoritmo de ordenação que é eficiente para certos tipos de dados. O algoritmo se baseia no princípio do "Pigeonhole", que afirma que se você tem mais "pombos" do que "buracos", pelo menos um buraco terá mais de um pombo. O Pigeonhole Sort funciona distribuindo os elementos em "buracos" correspondentes ao seu valor, e depois recolhendo esses elementos na ordem.

Cycle Sort: Cycle Sort é um algoritmo de ordenação in-place (que não usa espaço extra de armazenamento) e estável, conhecido por sua eficiência em termos de número de escritas, o que o torna útil quando a operação de escrita é cara. Este algoritmo é particularmente eficiente para ordenar listas com um número mínimo de trocas, funcionando bem para listas pequenas ou quando é importante minimizar o número de trocas de posição.

Bomb Sort: O Bomb Sort não é um algoritmo de ordenação amplamente conhecido ou documentado na literatura clássica de ciência da computação. É possível que você esteja se referindo a um algoritmo específico usado em um contexto particular ou a um nome coloquial para um algoritmo existente. Se houver um contexto ou uma descrição específica do Bomb Sort que você conhece, eu poderia fornecer uma explicação detalhada baseada nisso.

Gnome Sort: O Gnome Sort ordena uma lista movendo-se de um extremo ao outro e trocando elementos adjacentes que estão fora de ordem, semelhante ao Insertion Sort, mas sem usar uma estrutura de dados auxiliar como uma pilha. Se um par de elementos está fora de ordem, ele os troca e move-se um passo para trás; se estão em ordem, ele avança.

Cocktail Shaker: O Cocktail Shaker Sort, também conhecido como Bidirectional Bubble Sort ou Cocktail Sort, é uma variação do Bubble Sort que melhora o desempenho ao ordenar uma lista. Ele faz isso permitindo que os elementos "bolhassem" tanto para cima quanto para baixo da lista, em vez de apenas uma direção como no Bubble Sort tradicional. Essa técnica visa reduzir o número de iterações necessárias em certos casos, especialmente quando há muitos elementos desordenados perto do início ou do final da lista.

Counting Sort: é um algoritmo de ordenação eficiente e não-comparativo, adequado para ordenar coleções de números inteiros onde o intervalo dos valores (ou a diferença entre o valor máximo e mínimo) é razoavelmente pequeno. Ele funciona contando o número de ocorrências de cada elemento único na lista de entrada e usando essas contagens para colocar os elementos na ordem correta.

Bucket Sort: O Bucket Sort é eficiente quando o conjunto de dados é distribuído uniformemente entre os baldes e quando os baldes são ordenados de forma eficiente. É adequado para ordenar números reais em um intervalo fixo conhecido, mas pode ser impraticável quando os dados não estão uniformemente distribuídos ou quando o intervalo dos dados é muito grande.

Radix Sort: O Radix Sort é um algoritmo de ordenação não comparativo que classifica os elementos processando os dígitos individuais dos números. Ele classifica os números com base em cada dígito significativo, começando pelo dígito menos significativo até o dígito mais significativo. Radix Sort é eficiente para ordenar inteiros e outros tipos de dados que podem ser representados em forma de dígitos, como strings de caracteres alfanuméricos.

Heap Sort: O Heap Sort é um algoritmo de ordenação baseado na estrutura de dados heap, que é uma árvore binária completa que satisfaz a propriedade do heap: no caso de um max-heap, cada nó é maior ou igual aos seus filhos, e no caso de um min-heap, cada nó é menor ou igual aos seus filhos. O Heap Sort constrói um heap a partir dos dados de entrada e então usa a propriedade do heap para extrair os elementos na ordem correta.

Shell Sort: É uma generalização do Insertion Sort que melhora a eficiência ao permitir trocas de elementos que estão longe um do outro. O algoritmo começa ordenando pares de elementos distantes entre si e reduz progressivamente a distância entre os elementos a serem comparados. O Shell Sort é eficiente para listas de tamanho médio e é mais simples de implementar do que alguns algoritmos mais complexos.

Selection Sort: O Selection Sort é um algoritmo de ordenação simples e intuitivo. Ele divide a lista em duas partes: a sublista de elementos já ordenados e a sublista de elementos não ordenados. Inicialmente, a sublista ordenada está vazia e a sublista não ordenada contém todos os elementos. O algoritmo então seleciona o menor (ou maior, dependendo da ordem desejada) elemento da sublista não ordenada, troca-o com o primeiro elemento não ordenado, e move a fronteira entre as duas sublistas uma posição à direita.

Insertion Sort: O Insertion Sort é outro algoritmo de ordenação simples e eficiente para listas pequenas. Ele funciona de maneira semelhante ao modo como você organizaria cartas em suas mãos. O algoritmo constrói a lista ordenada uma entrada por vez, removendo um elemento da lista de entrada e encontrando a posição correta na sub lista ordenada.

Merge Sort: O Merge Sort é um algoritmo de ordenação eficiente baseado no paradigma de divisão e conquista. Ele divide a lista em sublistas menores, ordena essas sublistas e, em seguida, as mescla para formar uma lista ordenada final.

Bubble Sort: O Bubble Sort é um algoritmo de ordenação simples, mas ineficiente, que compara repetidamente pares adjacentes de elementos e os troca se estiverem fora de ordem. Esse processo continua até que toda a lista esteja ordenada.

TABELA COMPARATIVA ENTRE OS ORDENADORES

Foi utilizado como parâmetro 1000 números, 5000 números e 10.000 números.
Os resultados obtidos nos testes foram os seguintes:

ORDENADOR	TEMPO COM 1000	TEMPO COM 5000	TEMPO COM 10000
glowworm	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 1.90 milissegundos	Tempo decorrido: 2.00 milissegundos
slow	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 1.00 milissegundos	Tempo decorrido: 2.30 milissegundos
bogo	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 0.70 milissegundos	Tempo decorrido: 2.30 milissegundos
binary-tree	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 1.20 milissegundos	Tempo decorrido: 1.70 milissegundos
stooge	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 0.90 milissegundos	Tempo decorrido: 2.10 milissegundos
pigeonhole	Tempo decorrido: 0.20 milissegundos	Tempo decorrido: 2.00 milissegundos	Tempo decorrido: 2.60 milissegundos
cycle	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 0.90 milissegundos	Tempo decorrido: 1.80 milissegundos
comb	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 1.20 milissegundos	Tempo decorrido: 2.00 milissegundos
gnome	Tempo decorrido: 0.10 milissegundos	Tempo decorrido: 0.90 milissegundos	Tempo decorrido: 2.70 milissegundos
cocktail	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 0.80 milissegundos	Tempo decorrido: 1.80 milissegundos
counting	Tempo decorrido: 0.20 milissegundos	Tempo decorrido: 1.00 milissegundos	Tempo decorrido: 1.90 milissegundos
bucket	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 0.90 milissegundos	Tempo decorrido: 2.10 milissegundos
radix	Tempo decorrido: 0.40 milissegundos	Tempo decorrido: 1.00 milissegundos	Tempo decorrido: 2.50 milissegundos
heap	Tempo decorrido: 0.20 milissegundos	Tempo decorrido: 0.90 milissegundos	Tempo decorrido: 1.80 milissegundos
shell	Tempo decorrido: 0.20 milissegundos	Tempo decorrido: 0.80 milissegundos	Tempo decorrido: 2.10 milissegundos
selection	Tempo decorrido: 0.40 milissegundos	Tempo decorrido: 0.90 milissegundos	Tempo decorrido: 2.40 milissegundos
insertion	Tempo decorrido: 0.20 milissegundos	Tempo decorrido: 1.10 milissegundos	Tempo decorrido: 2.00 milissegundos
merge	Tempo decorrido: 0.20 milissegundos	Tempo decorrido: 1.20 milissegundos	Tempo decorrido: 2.50 milissegundos
bubble	Tempo decorrido: 0.30 milissegundos	Tempo decorrido: 0.90 milissegundos	Tempo decorrido: 1.80 milissegundos

Após os testes realizados, nota-se que nem todos os ordenadores que são mais rápidos com 1000 números serão os mais rápidos com 5000 números, e dependendo da ocasião que necessitamos será possível utilizar vários ordenadores na ocasião.

Parte dos códigos utilizados na aplicação

```html
<h1>Ordenadores</h1>

<div class="sort-buttons">
    <label for="inputNumbers">Insira os números (separados por vírgula):</label><br>
    <input type="text" id="inputNumbers"><br><br>

    <select id="sortAlgorithm">
        <option value="glowworm">Glowworm</option>
        <option value="slow">Slow</option>
        <option value="bogo">Bogo</option>
        <option value="binary-tree">Binary Tree</option>
        <option value="stooge">Stooge</option>
        <option value="pigeonhole">Pigeonhole</option>
        <option value="cycle">Cycle</option>
        <option value="comb">Comb</option>
        <option value="gnome">Gnome</option>
        <option value="cocktail">Cocktail Shaker</option>
        <option value="counting">Counting</option>
        <option value="bucket">Bucket</option>
        <option value="radix">Radix</option>
        <option value="heap">Heap</option>
        <option value="shell">Shell</option>
        <option value="selection">Selection</option>
        <option value="insertion">Insertion</option>
        <option value="merge">Merge</option>
        <option value="bubble">Bubble</option>
    </select>
    <button class="sort-button" onclick="sort()">Ordenar</button>
    <button class="reset-button" onclick="resetInput()">Zerar</button> <!-- Botão pra zerar os ordenadores -->
</div>
```

Nesse exemplo, é listado para selecionar todos os ordenadores que poderá ser selecionado pelo usuário antes de ordenar os números.

```javascript
    function heapSort(array) {
        // Implementação do Heap Sort
        document.getElementById("output").innerHTML = "Heap Sort: " + array.join(", ");
    }

    function shellSort(array) {
        // Implementação do Shell Sort
        document.getElementById("output").innerHTML = "Shell Sort: " + array.join(", ");
    }

    function selectionSort(array) {
        // Implementação do Selection Sort
        document.getElementById("output").innerHTML = "Selection Sort: " + array.join(", ");
    }

    function insertionSort(array) {
        // Implementação do Insertion Sort
        document.getElementById("output").innerHTML = "Insertion Sort: " + array.join(", ");
    }

    function mergeSort(array) {
        // Implementação do Merge Sort
        document.getElementById("output").innerHTML = "Merge Sort: " + array.join(", ");
    }

    function bubbleSort(array) {
        // Implementação do Bubble Sort
        document.getElementById("output").innerHTML = "Bubble Sort: " + array.join(", ");
    }
</script>
```

Parte do Script utilizado para funcionar em cada ordenador selecionado pelo usuário.

Conclusão

Neste artigo, exploramos a funcionalidade de ordenação em uma página HTML, que permite que o usuário insira uma lista de números e os ordene usando diferentes algoritmos de ordenação disponíveis. Esta implementação demonstra a combinação de HTML, CSS e JavaScript para criar uma experiência interativa na web.
Mesmo alguns ordenadores sendo mais rápidos que outros, podemos notar que todos têm uma função específica e suas devidas funcionalidades.

Referências Bibliográficas

- https://www.atrainformatica.com.br/2023/07/20/historia-e-evolucao-dos-computadores/
- https://resumos.soescola.com/glossario/ordenador-o-que-e-significado/
- https://docente.ifrn.edu.br/carlossilva/disciplinas/2012.1/informatica-basica-licqui1n/aula-01-introducao-a-ib
- https://www.sorocaba.unesp.br/Home/Graduacao/EngenhariadeControleeAutomacao/marcio/transparencia_aula-1-icc_2016-introd_software.pdf