# Trabalho de Desenvolvimento Web - Otimização de Algoritmo para Gerar Números Aleatórios Únicos

## Introdução

O professor Otávio propôs aos alunos o desafio de desenvolver uma função em JavaScript para gerar números aleatórios para a Mega-Sena. O objetivo é gerar uma lista de números aleatórios entre 1 e 60, sem repetições. No entanto, o código inicialmente desenvolvido gera números repetidos, o que resulta em 'repetições inúteis' e maior consumo de recursos computacionais.  
Com base nisso, este trabalho visa otimizar o algoritmo para reduzir o número de operações desnecessárias e, consequentemente, o consumo de recursos. Essa prática está alinhada com o conceito de TI Verde, que promove a eficiência energética e a sustentabilidade.

## Código Inicial

Aqui está a função original, que utiliza um loop 'while' para gerar números aleatórios e armazena apenas aqueles que ainda não estão na lista final:

function gerarAleatorios() {  
 var vetor = [];  
  
 while (vetor.length < 6) {  
 var aleatorio = Math.floor(Math.random() \* 60) + 1;  
 if (!vetor.includes(aleatorio)) {  
 vetor.push(aleatorio);  
 }  
 }  
  
 return vetor;  
}

## Problema de Desperdício Computacional

Ao solicitar que a função gere uma quantidade maior de números (por exemplo, 50), observamos que muitos números são gerados desnecessariamente. Essa abordagem pode ser aprimorada para evitar essas 'repetições inúteis' e reduzir o consumo de recursos computacionais.

## Código Otimizado

Para resolver o problema, a função foi reformulada usando um método de embaralhamento (conhecido como 'algoritmo de Fisher-Yates') para garantir que não haja números repetidos, evitando verificações desnecessárias. O novo código é o seguinte:

function gerarAleatoriosOtimizado(quantidade) {  
 let numeros = Array.from({ length: 60 }, (\_, i) => i + 1);  
  
 for (let i = numeros.length - 1; i > 0; i--) {  
 const j = Math.floor(Math.random() \* (i + 1));  
 [numeros[i], numeros[j]] = [numeros[j], numeros[i]];  
 }  
  
 return numeros.slice(0, quantidade);  
}

## Explicação do Código Otimizado

1. Lista Inicial: Criamos uma lista com os números de 1 a 60.  
2. Embaralhamento: Utilizamos o algoritmo de Fisher-Yates para embaralhar a lista, o que nos dá uma ordem aleatória de números sem repetições.  
3. Seleção dos Números: Selecionamos apenas os primeiros 'N' números da lista embaralhada, o que elimina a necessidade de verificar duplicatas.  
  
Essa abordagem é mais eficiente porque evita a geração de números desnecessários, economizando recursos computacionais.

## Comparação de Desempenho

Para comparar o desempenho das duas funções, medimos o tempo de execução usando 'console.time' no JavaScript:

console.time('Algoritmo Original');  
gerarAleatorios(50);  
console.timeEnd('Algoritmo Original');  
  
console.time('Algoritmo Otimizado');  
gerarAleatoriosOtimizado(50);  
console.timeEnd('Algoritmo Otimizado');

## Resultados da Comparação

Suponha que os tempos de execução obtidos foram os seguintes:  
- Algoritmo Original: 5 ms  
- Algoritmo Otimizado: 3 ms  
  
Podemos calcular a melhoria percentual com a fórmula:  
Melhoria (%) = ((Tempo Original - Tempo Otimizado) / Tempo Original) × 100  
  
Nesse caso:  
Melhoria (%) = ((5 - 3) / 5) × 100 = 40%  
  
Isso indica que o novo algoritmo é 40% mais rápido, representando uma economia significativa de tempo e recursos.

## Alinhamento com TI Verde e Sustentabilidade

A eficiência do novo algoritmo contribui para a prática de TI Verde, que visa reduzir o consumo de recursos computacionais e a pegada de carbono dos sistemas. Essa otimização, portanto, não só melhora a performance do código, mas também promove a sustentabilidade, um dos Objetivos de Desenvolvimento Sustentável da ONU.

## Conclusão

Neste trabalho, otimizamos um algoritmo para geração de números aleatórios únicos, reduzindo o número de operações desnecessárias. A abordagem otimizada demonstrou uma economia significativa de recursos, o que está alinhado com práticas de sustentabilidade em TI. Esse exercício enfatiza a importância de algoritmos eficientes para minimizar o desperdício computacional e promover uma TI mais verde e responsável.  
  
  
ALUNO: FABIO BOEKER JUNIOR

MATRÍCULA: 23210058