Método de organização Shell Sort

Índice de conteúdo

01

Histórico de criação

Histórico de Criação

Shell Sort é um algoritmo de ordenação eficiente, criado em 1959 por Donald Shell e publicado pela Universidade de Cincinnati. Ele é uma extensão do método InsertionSort.

Funcionamento-teoria

1 - Se estabelece um intervalo h.

2 - A coleção é dividida em subgrupos, de forma que este cada um dos elementos esteja a uma distância de h um do outro.

3 - Reorganiza o subgrupo.

4 - Diminui o h e os passos 2 a 4 se reiniciam até que h seja igual a 1.

02

Complexidade computacional

Complexidade computacional

Melhor caso: Quando o código está parcialmente ordenado, sendo necessárias:

$O(n \log_2 n)$ comparações

Pior caso: Ocorre quando os elementos estão dispostos de forma inversa a ordenação exigida. O número de comparações no pior caso também depende dos intervalos de h, e logo da equação utilizada para determinar estes h. Com a equação de Shell, seu pior caso é:

$O(n^2)$ comparações

Complexidade computacional - exemplos

OEIS	General term ($k \geq 1$)	Concrete gaps	Worst-case time complexity	Author and year of publication
	$\left\lfloor \frac{N}{2^k} \right\rfloor$	$1, 2, \ldots, \left\lfloor \frac{N}{4} \right\rfloor, \left\lfloor \frac{N}{2} \right\rfloor$	$\Theta\left(N^2\right)$ [e.g. when $N = 2^p$]	Shell, 1959[4]
	$2 \left\lfloor \frac{N}{2^{k+1}} \right\rfloor + 1$	$1, 3, \ldots, 2\left\lfloor \frac{N}{8} \right\rfloor + 1, \ 2\left\lfloor \frac{N}{4} \right\rfloor + 1$	$\Theta\left(N^{\frac{3}{2}}\right)$	Frank & Lazarus, 1960[8]
A000225	$2^k - 1$	$1, 3, 7, 15, 31, 63, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$	Hibbard, 1963[9]
A083318	$2^k + 1$, prefixed with 1	$1, 3, 5, 9, 17, 33, 65, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$	Papernov & Stasevich, 1965[10]
A003586	Successive numbers of the form $2^p 3^q$ (3-smooth numbers)	$1, 2, 3, 4, 6, 8, 9, 12, \ldots$	$\Theta\left(N \log^2 N\right)$	Pratt, 1971[1]
A003462	$\frac{3^k - 1}{2}$, not greater than $\left\lceil \frac{N}{3} \right\rceil$	$1, 4, 13, 40, 121, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$	Knuth, 1973,[3] based on Pratt, 1971[1]
A036569	$\prod_I a_q$, where $a_0 = 3$ $a_q = \min\left\{n \in \mathbb{N}: n \geq \left(\frac{5}{2}\right)^{q+1}, \forall p: 0 \leq p < q \rightarrow \gcd(a_p, n) = 1\right\}$ $I = \left\{0 \leq q < r \mid q \neq \frac{1}{2}\left(r^2 + r\right) - k\right\}$ $r = \left\lfloor \sqrt{2k + \sqrt{2k}} \right\rfloor$	$1, 3, 7, 21, 48, 112, \ldots$	$O\left(N^{1+\sqrt{\frac{8\ln(5/2)}{\ln(N)}}}\right)$	Incerpi & Sedgewick, 1985,[11] Knuth[3]
A036562	$4^k + 3 \cdot 2^{k-1} + 1$, prefixed with 1	$1, 8, 23, 77, 281, \ldots$	$O\left(N^{\frac{4}{3}}\right)$	Sedgewick, 1982[6]
A033622	$\begin{cases} 9\left(2^k - 2^{\frac{k}{2}}\right) + 1 & k \text{ even}, \\ 8 \cdot 2^k - 6 \cdot 2^{(k+1)/2} + 1 & k \text{ odd} \end{cases}$	$1, 5, 19, 41, 109, \ldots$	$O\left(N^{\frac{4}{3}}\right)$	Sedgewick, 1986[12]
	$h_k = \max\left\{\left\lfloor \frac{5h_{k-1} - 1}{11} \right\rfloor, 1\right\}, h_0 = N$	$1, \ldots, \left\lfloor \frac{5}{11}\left\lfloor \frac{5N-1}{11} \right\rfloor - \frac{1}{11} \right\rfloor, \left\lfloor \frac{5N-1}{11} \right\rfloor$	Unknown	Gonnet & Baeza-Yates, 1991[13]
A108870	$\left\lceil \frac{1}{5}\left(9 \cdot \left(\frac{9}{4}\right)^{k-1} - 4\right) \right\rceil$ (or equivalently, $\left\lceil \frac{(9/4)^k - 1}{(9/4) - 1} \right\rceil$)	$1, 4, 9, 20, 46, 103, \ldots$	Unknown	Tokuda, 1992[14] (misquote per OEIS)
A102549	Unknown (experimentally derived)	$1, 4, 10, 23, 57, 132, 301, 701$	Unknown	Ciura, 2001[15]
	$\left\lceil \frac{\gamma^k - 1}{\gamma - 1} \right\rceil, \gamma = 2.243609061420001\ldots$	$1, 4, 9, 20, 45, 102, 230, 516, \ldots$	Unknown	Lee, 2021[16]
	$\left\lfloor 4.0816 \cdot 8.5714^{\left\lfloor \frac{k}{2.2449} \right\rfloor} \right\rfloor$	$1, 4, 10, 27, 72, 187, 488, \ldots$	Unknown	Skean, Ehrenborg, Jaromczyk, 2023[17]

Complexidade computacional - principais

Sequência de Shell – $h = n/2^k$, onde n é o tamanho da coleção e k é o passo no algoritmo Shell Sort. Pior caso $O(n^2)$.

Sequência de Hibbard – $h = 2^k - 1$, onde k é o passo no algoritmo Shell Sort. Pior caso $O(n^{3/2})$.

Sequência de Knuth – $h = (3^k - 1) / 2$, onde k é o passo no algoritmo Shell Sort. Pior caso $O(n^{3/2})$.

03

Como o ShellSort funciona?

Como o ShellSort funciona?

Inicializa a função e as variáveis

```
void shellsort(struct item *vetor, int tamanho) {
int i, j, intervalo;
struct item auxiliar;
```

Roda um laço para determinar o valor inicial do intervalo

```
/* calcula o intervalo inicial. */
for(intervalo = 1; intervalo < tamanho; intervalo = 3*intervalo+1);
```

Decrementa o valor do intervalo para o próximo degrau abaixo

```
while(intervalo > 0) {
        /* atualiza o valor do intervalo. */
            intervalo = (intervalo-1)/3;
```

Laço que posiciona e atribui valores às variáveis

```
            for(i = intervalo; i < tamanho; i++) {
            auxiliar = vetor[i];
            j = i;
```

Compara se a chave na posição j - intervalo é maior que a chave na variável auxiliar

```
            /* efetua comparações entre elementos com distância do intervalo: */
            while(vetor[j – intervalo].chave > auxiliar.chave) {
                vetor[j] = vetor[j – intervalo];
                j -= intervalo;
```

Troca o valor das variáveis

```
                    vetor[j] = auxiliar;
                    }
                }
            }
```

Como o ShellSort funciona?

```
for(intervalo = 1; intervalo < tamanho; intervalo = 3*intervalo+1);
```

- Define o valor inicial do intervalo dependendo do tamanho do vetor
- O intervalo só pode assumir os valores 1, 4, 13, 40, 121, 364, 1.093, 3.280, ...
- O laço termina em si mesmo e acaba quando o intervalo assumir o valor mais próximo e menor do que o tamanho do array seguindo os valores acima.

Exemplo: tamanho do array: 15
Valor inicial do intervalo: 13

Como o ShellSort funciona?

```
while(intervalo > 0) {
        intervalo = (intervalo-1)/3;
```

- Diminui o valor do intervalo para cada ciclo efetuado enquanto o intervalo for maior que zero.
- Note que a operação (intervalo-1)/3 é o inverso da operação apresentada no slide anterior (3*intervalo+1).
- Portanto, o intervalo ainda segue a mesma sequência de valores: 1, 4, 13, 40, 121, 364, 1.093, 3.280, ...

Como o ShellSort funciona?

```
for(i = intervalo; i < tamanho; i++) {
auxiliar = vetor[i];
j = i;
```

- Laço para atribuir valor às variáveis.

- Coloca a variável i na mesma posição do intervalo. Enquanto i
  for menor que o tamanho do vetor, ou seja, ainda estiver dentro do vetor, o valor de i é incrementado.

- A variável auxiliar recebe os valores dentro da posição
  indicada por i.

- J fica na mesma posição de i.

Como o ShellSort funciona?

```
while(vetor[j – intervalo].chave > auxiliar.chave) {
vetor[j] = vetor[j – intervalo];
j -= intervalo;
```

- Faz uma verificação: executa o código enquanto o valor da chave contido na posição (j - intervalo) for maior que o valor da
  chave na posição da variável auxiliar.

- Se a verificação acima for verdadeira, os valores do vetor na posição j recebem os valores da posição j - intervalo.

- Após isso, ocorre a atribuição j-= intervalo, que nada mais é
que j = j - intervalo, o que faz com que j vá para a posição que ele estava anteriormente verificando.

Como o ShellSort funciona?

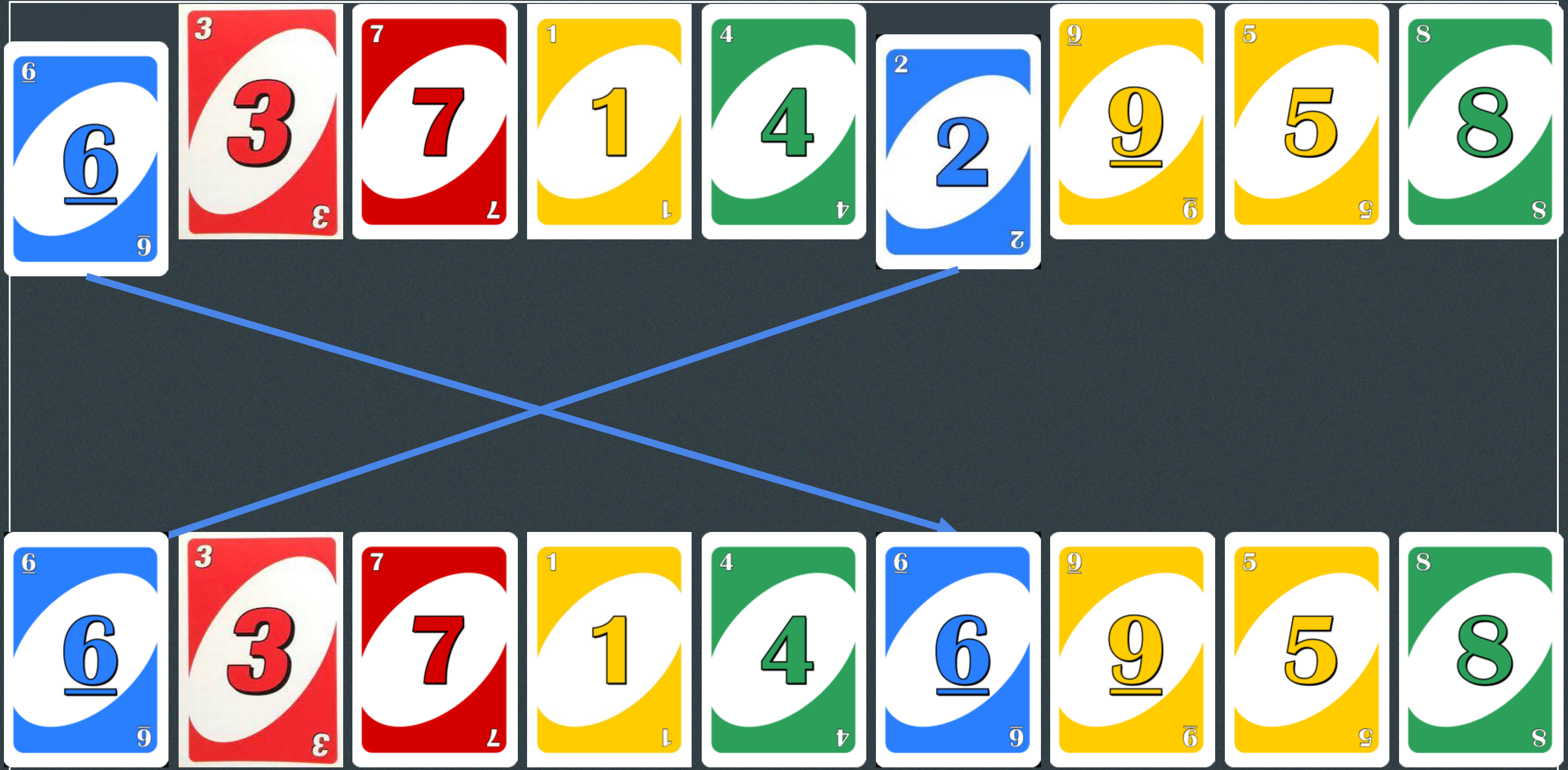
```
vetor[j] = auxiliar;
```

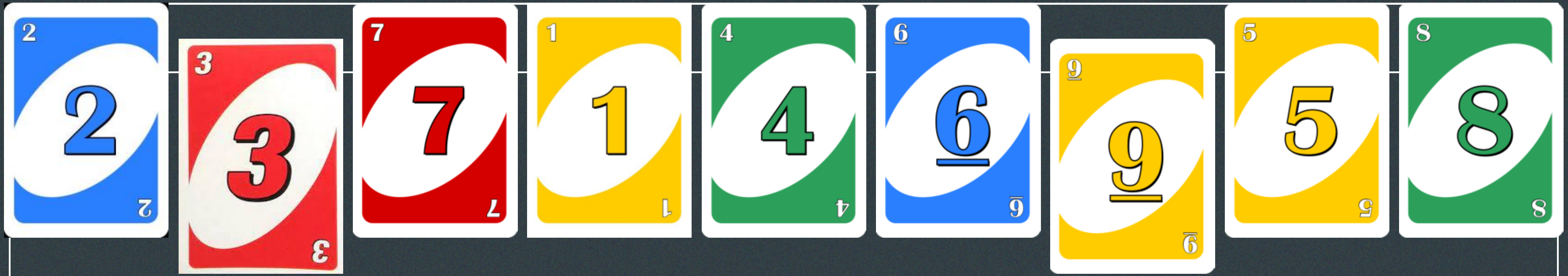
- Por último, o vetor na posição j agora recebe os valores contidos na variável auxiliar

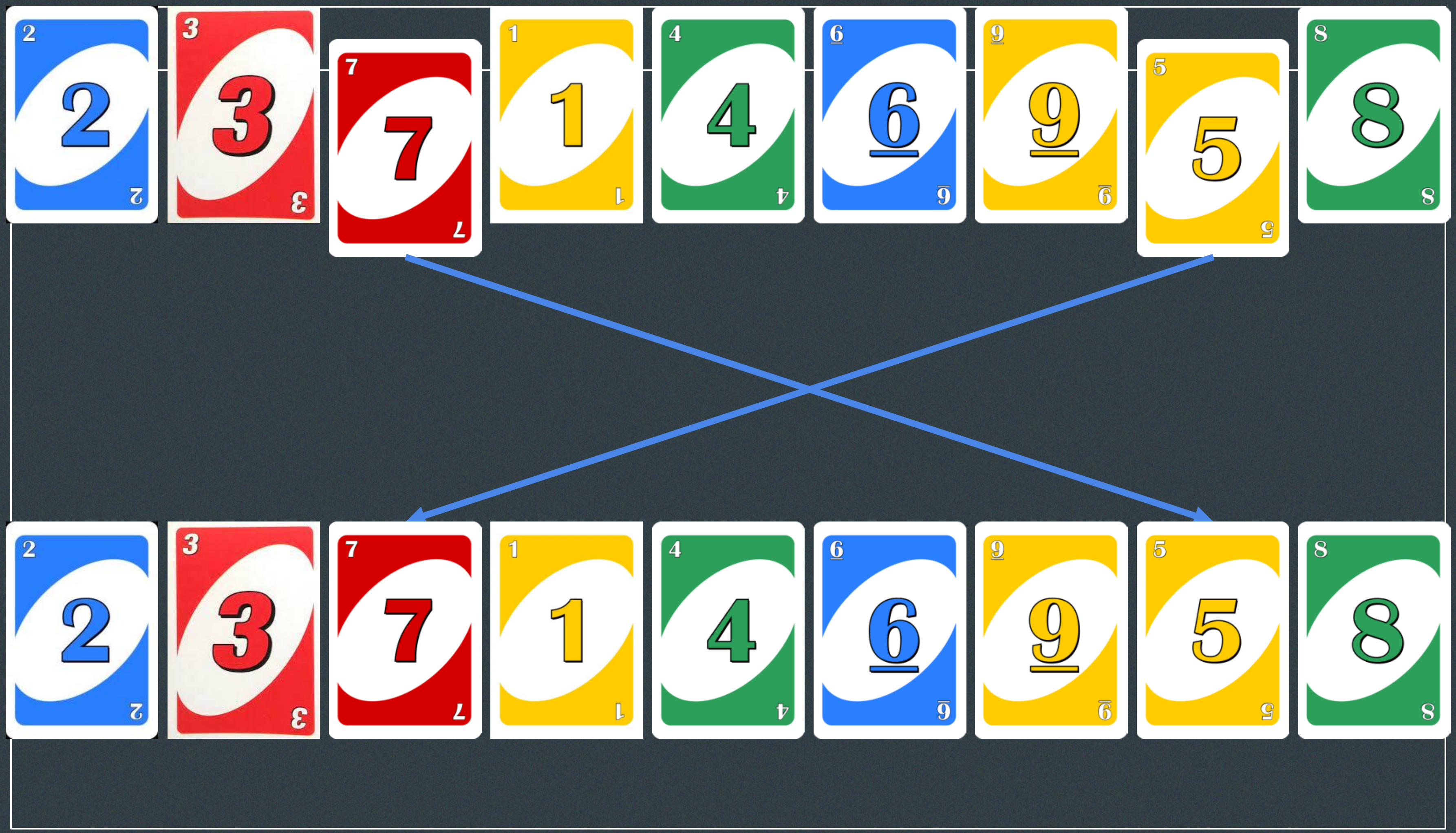
Em resumo, o código verificou se um determinado valor atrás da variável i era maior que o valor contido em i. Então, caso fosse, o valor contido na posição da variável i é passado para trás e o valor de trás é passado para frente, assumindo o valor na posição do i.

Valores do Intervalo

5,2 e 1


6 3 7 1 4 2 9 5 8
6 3 7 1 4 6 9 5 8

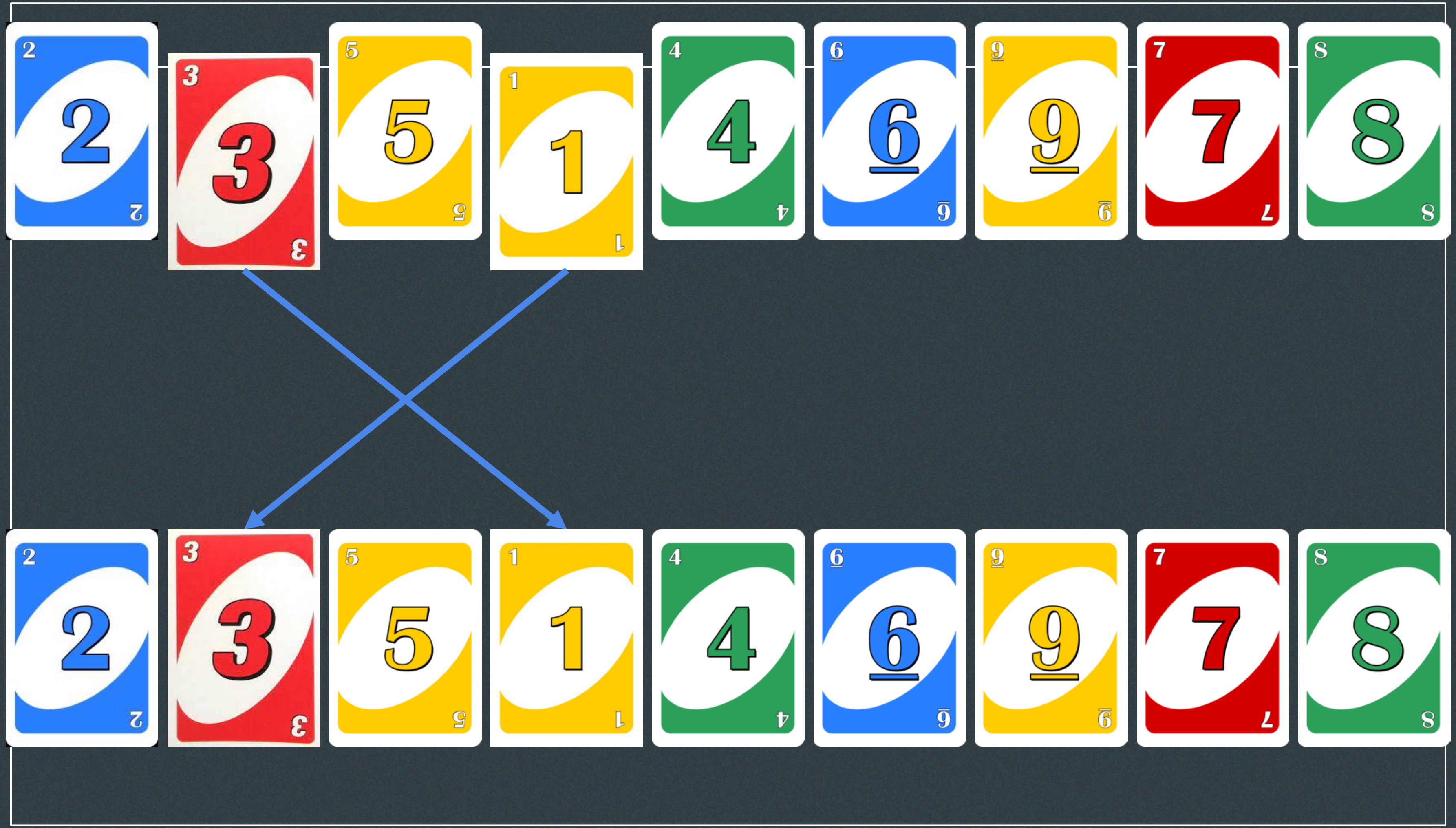

2
3
7
1
4
6
9
5
8



2
3
7
1
4
6
9
5
8
2
3
7
1
4
6
9
5
8


2
3
5
1
4
6
9
7
8


2
3
5
1
4
6
9
7
8


2 3 5 1 4 6 9 7 8
2 3 5 1 4 6 9 7 8


2 1 5 3 4 6 9 7 8
2 1 5 3 4 6 9 7 8


2
1
4
3
5
6
9
7
8


2
1
4
3
5
6
9
7
8


2
1
4
3
5
6
9
7
8


2 1 4 3 5 6 9 7 8
2 1 4 3 5 6 9 7 8


2 1 4 3 5 6 8 7 9
2 1 4 3 5 6 8 7 9

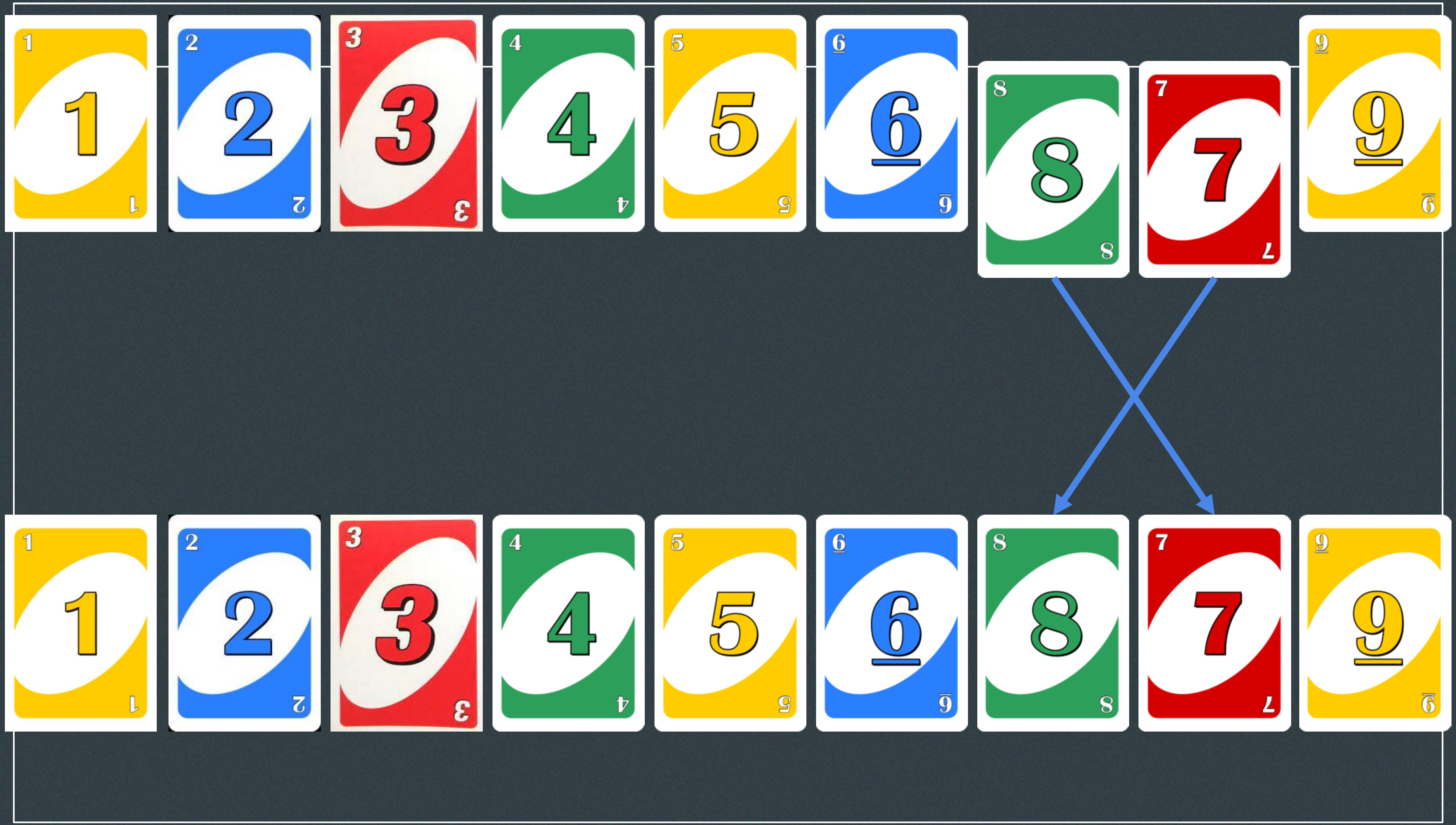

1
2
4
3
5
6
8
7
9

1
2
4
3
5
6
8
7
9
1
2
4
3
5
6
8
7
9


1
2
3
4
5
6
8
7
9


1
2
3
4
5
6
8
7
9

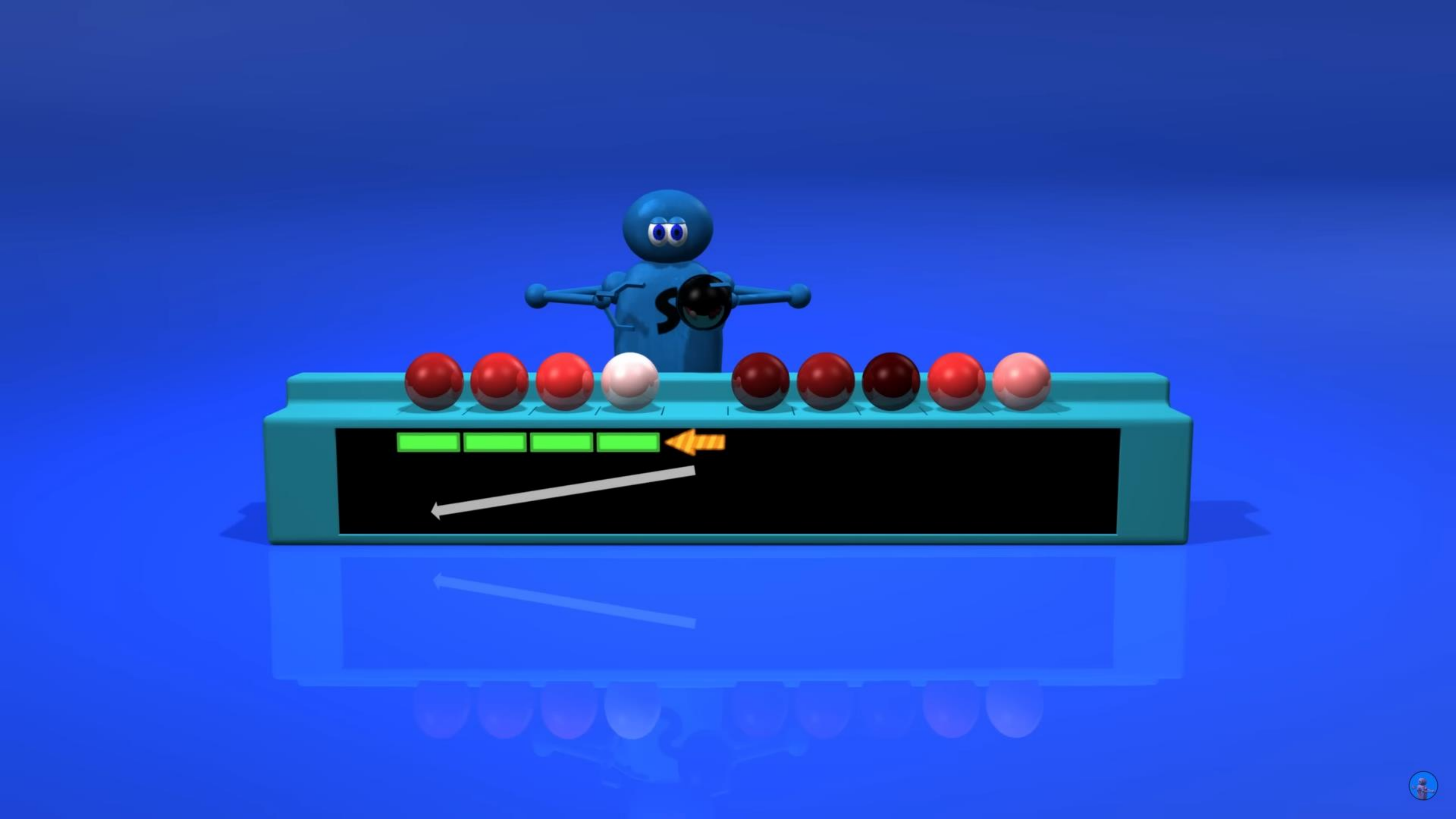

1
2
3
4
5
6
8
7
9



1 2 3 4 5 6 8 7 9
1 2 3 4 5 6 8 7 9

04

Foi aí?



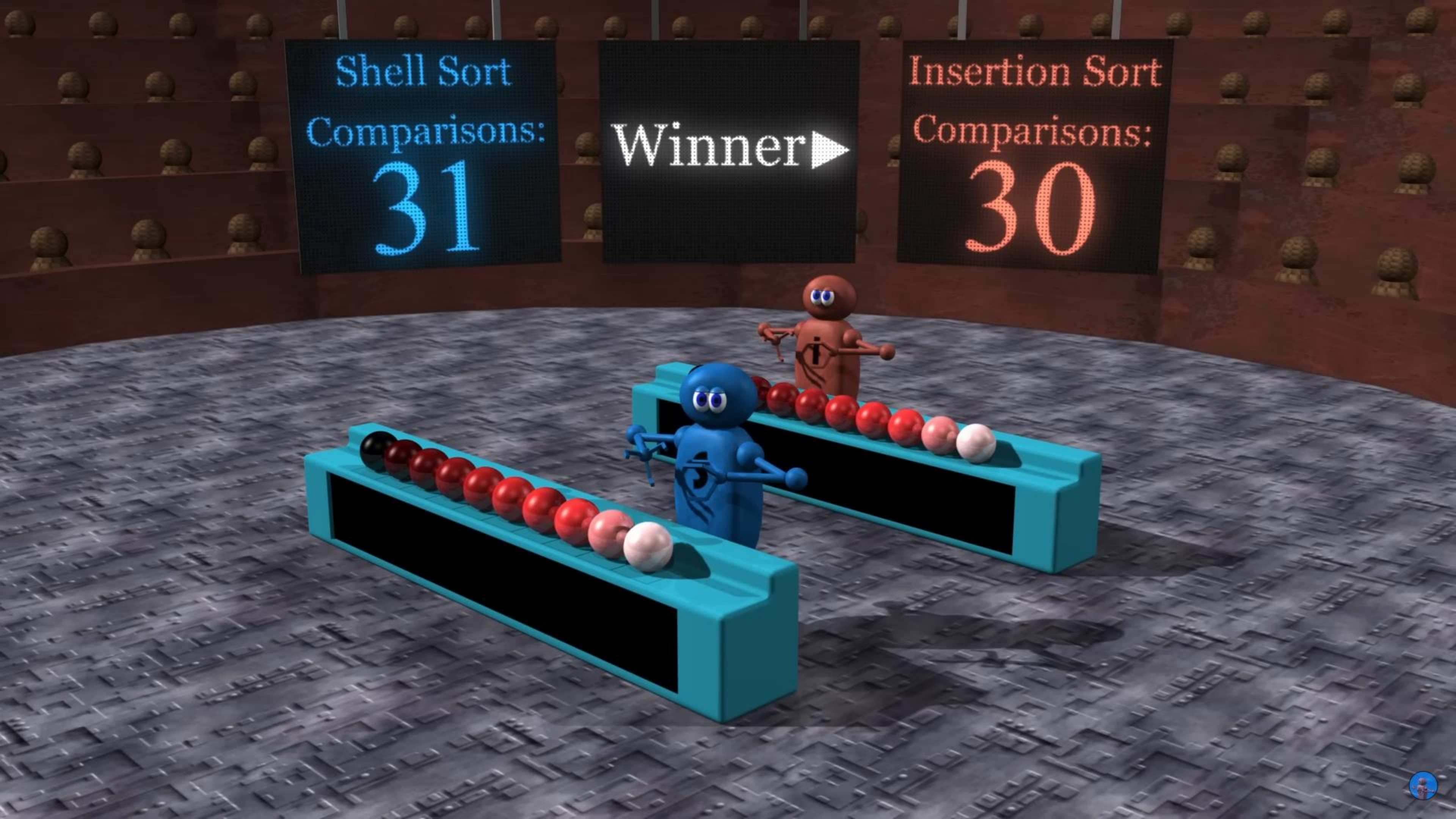
Valores do Intervalo

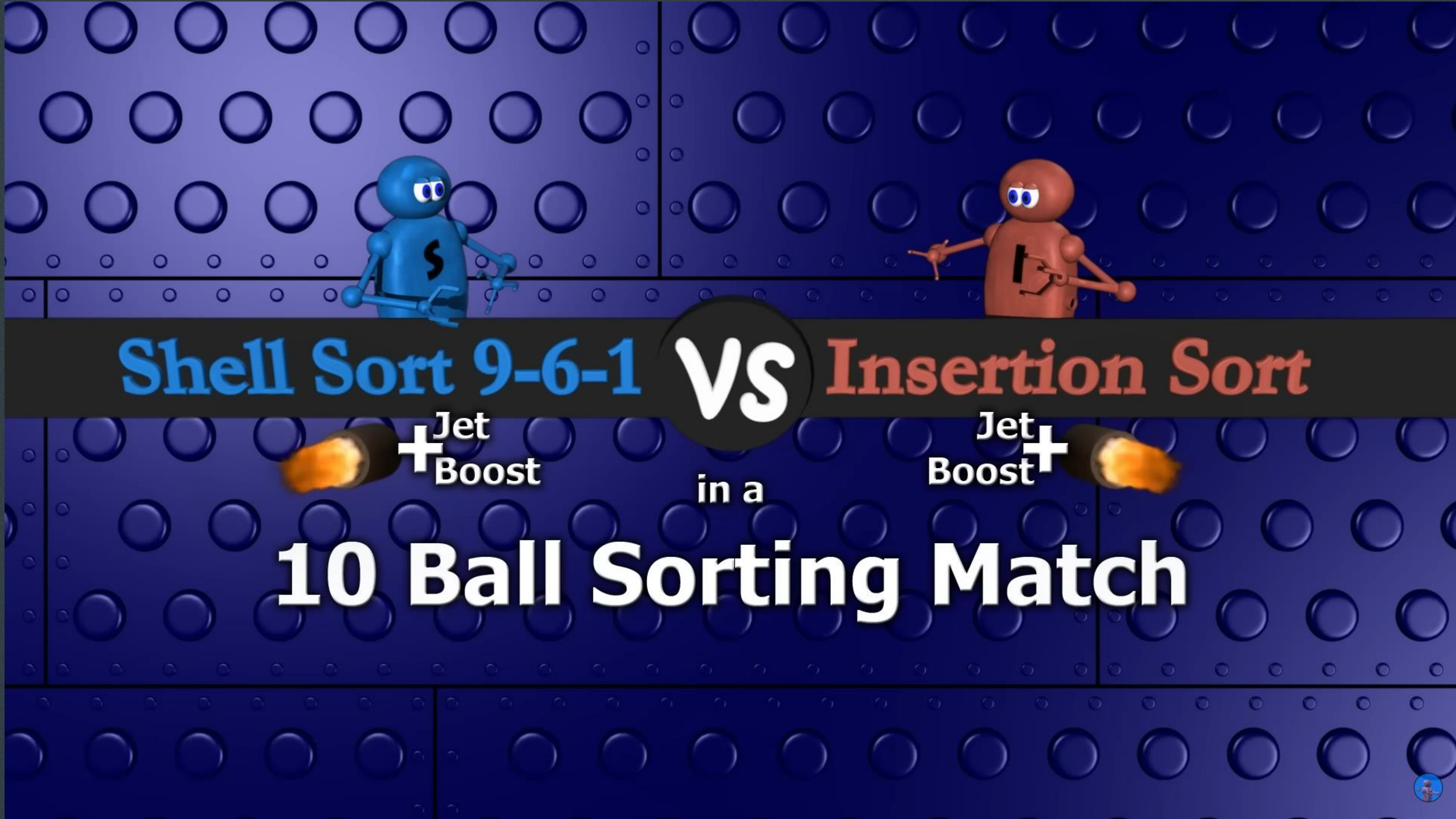
5,2 e 1

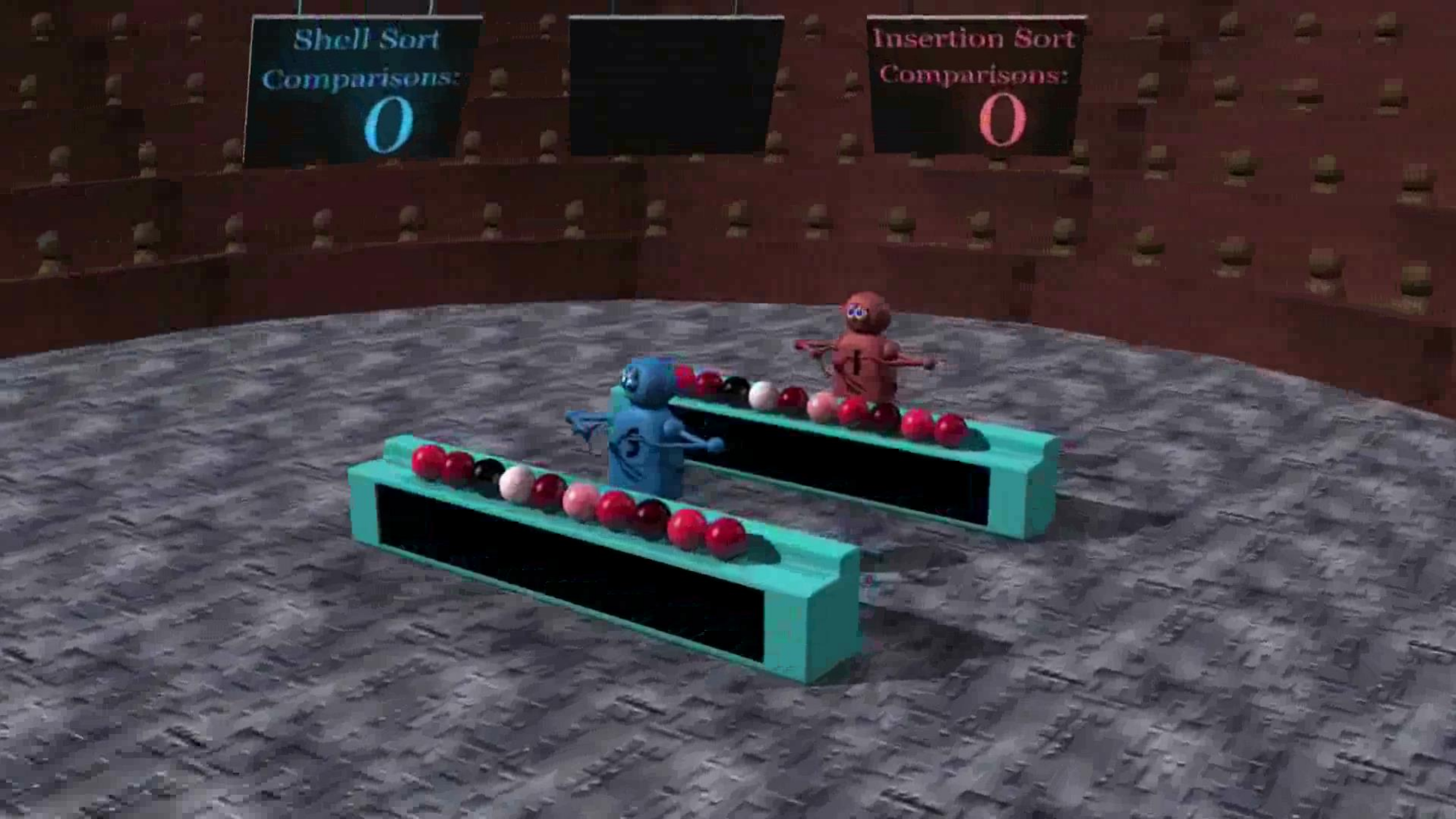
Shell Sort VS Insertion Sort

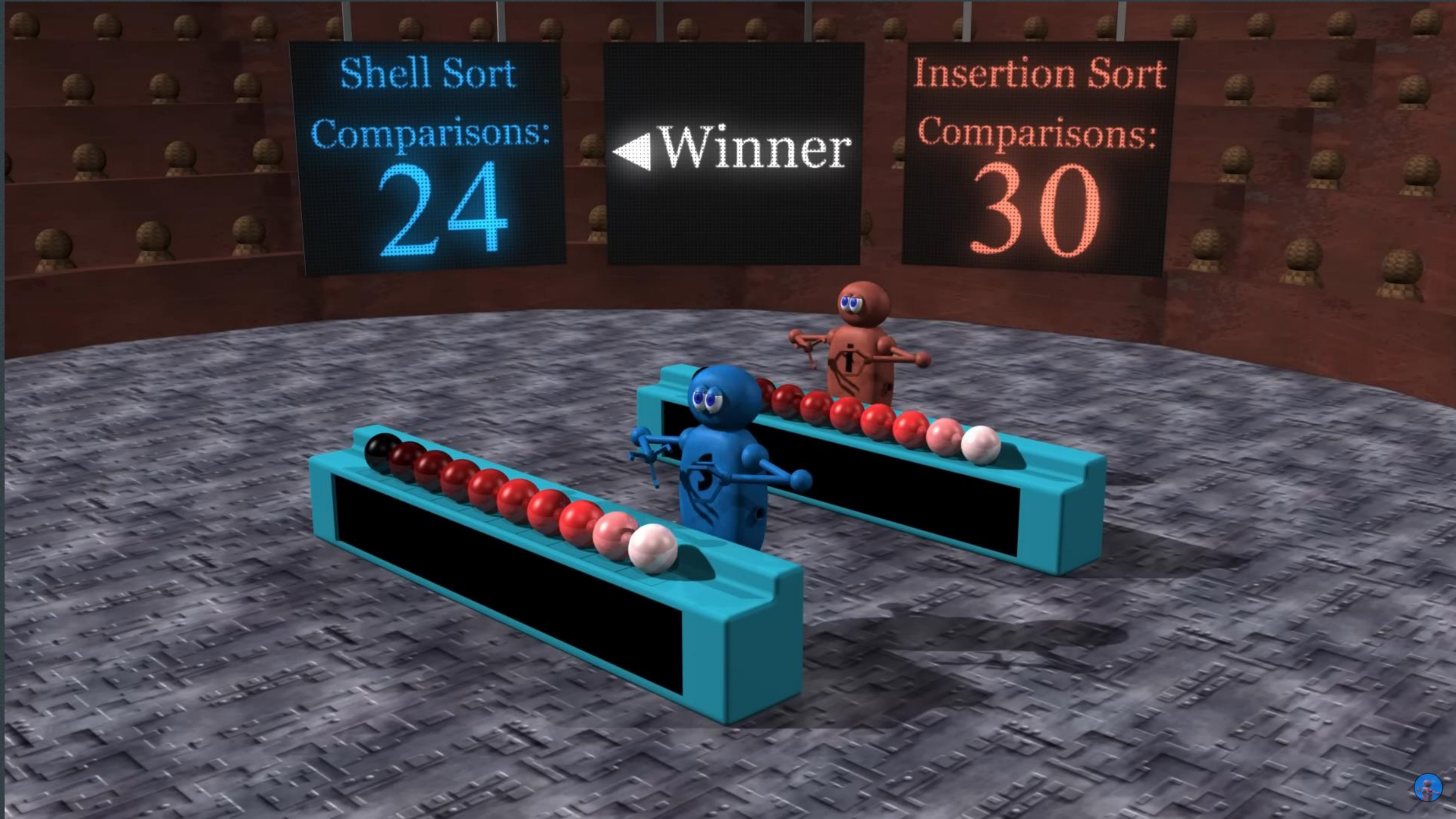
in a

10 Ball Sorting Match


Shell Sort
Comparisons:
31
Winner
Insertion Sort
Comparisons:
30


Shell Sort 9-6-1
VS
Insertion Sort
+Jet Boost
Jet Boost+
in a
10 Ball Sorting Match


Shell Sort
Comparisons:
0
Insertion Sort
Comparisons:
0


Shell Sort
Comparisons:
24
◀Winner
Insertion Sort
Comparisons:
30

Referências

http://www3.decom.ufop.br/toffolo/site_media/uploads/2013-1/bcc202/slides/16._shellsort.pdf

https://www.youtube.com/watch?v=M3bS6w1R434

https://www.youtube.com/watch?v=g06hNBhoS1k

https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao#:~:text=Selection%20Sort,-A%20ordena%C3%A7%C3%A3o%20por&text=Para%20todos%20os%20casos%20(melhor,n%C3%A3o%20%C3%A9%20um%20algoritmo%20est%C3%A1vel.

http://desenvolvendosoftware.com.br/algoritmos/ordenacao/shell-sort.html

Referências

https://en.wikipedia.org/wiki/Shellsort#Gap_sequences

https://www.baeldung.com/cs/shellsort-complexity

Obrigado!