COM 112

Counting Sort
&
E-Counting Sort

Integer Sorting

O Counting Sort, ao contrário dos algoritmos visto em aula, não trabalha com comparações. Seu método trabalha ao redor das propriedades dos números inteiros para conseguir organizar os dados recebidos.

Definição

Supondo que cada um dos n elementos de entrada é um inteiro na faixa 1 a k, para algum inteiro k.

Counting Sort determina, para cada x, o número de elementos menores que ele e insere o elemento x diretamente em sua posição.

Algoritmo **COUNTING SORT**

```
for(int i = 0; i < A.tamanho; i++)
    C[ A[ i ]]++
```

A =	3	1	1	2	4	2	0	3
	0	1	2	3	4	5	6	7

C =	0	0	0	0	0	0
	0	1	2	3	4	5

Algoritmo **COUNTING SORT**

```
for(int i = 0; i < A.tamanho; i++)
    C[ A [ i ]]++
```

A =

3	1	1	2	4	2	0	3
0	1	2	3	4	5	6	7

C =

0	0	0	1	0	0
0	1	2	3	4	5

Algoritmo **COUNTING SORT**

```
for(int i = 0; i < A.tamanho; i++)
    C[ A[ i ]]++
```

A =	3	1	1	2	4	2	0	3
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

C =	1	2	2	2	1	0
	0	**1**	**2**	**3**	**4**	**5**

Algoritmo **COUNTING SORT**

```
for(int i = 1; i <= maior; i++)
    C[ i ] += c[ i-1]
```

A =

3	1	1	2	4	2	0	3
0	1	2	3	4	5	6	7

C =

1	3	2	2	1	0
0	1	2	3	4	5

Algoritmo **COUNTING SORT**

```
for(int i = 1; i <= maior; i++)
    C[ i ] += c[ i-1]
```

A =

3	1	1	2	4	2	0	3
0	1	2	3	4	5	6	7

C =

1	3	5	7	8	8
0	1	2	3	4	5

Algoritmo **COUNTING SORT**

```
for(int i = tam - 1; i >= 0; i - -)
    B[ C[ A[ i ] ] - 1] = A[ i ]
    C[ A[ i ] ] - -
```

B =

0	0	0	0	0	0	3	0
0	1	2	3	4	5	6	7

C =

1	3	5	7->6	8	8
0	1	2	3	4	5

A =

3	1	1	2	4	2	0	3
0	1	2	3	4	5	6	7

Algoritmo **COUNTING SORT**

```
for(int i = tam - 1; i >= 0; i - -)
    B[ C[ A[ i ] ] - 1] = A[ i ]
    C[ A[ i ] ] - -
```

B =

0	0	0	0	0	0	3	0
0	1	2	3	4	5	6	7

C =

1->0	3	5	6	8	8
0	1	2	3	4	5

A =

3	1	1	2	4	2	0	3
0	1	2	3	4	5	6	7

Algoritmo **COUNTING SORT**

```
for(int i = tam - 1; i >= 0; i - -)
    B[ C[ A[ i ] ] - 1] = A[ i ]
    C[ A[ i ] ] - -
```

B =

0	1	2	3	4	5	6	7
0	1	1	2	2	3	3	4

C =

0	1	2	3	4	5
0	3->2->1	5->4->3	6->5	8->7	8

A =

0	1	2	3	4	5	6	7
3	1	1	2	4	2	0	3

Código do Counting Sort

```
COUNTING-SORT(A,B, k)
1   seja C[0 .. k] um novo arranjo
2   for i = 0 to k
3       C[i] = 0
4   for j = 1 to A·comprimento
5       C[A[j]] = C[A[j]] + 1
6   // C[i] agora contém o número de elementos igual a i.
7   for i = 1 to k
8       C[i] = C[i] + C[i − 1]
9   // C[i] agora contém o número de elementos menores que ou iguais a i.
10  for j = A·comprimento downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Código do Counting Sort

```c
void countingSort(int *arrA, int *arrB, int max, int tam, data *comp){
    int i;
    int* count = calloc(max + 1, sizeof(int));

    for (i = 0; i < tam; i++)
    {
        count[arrA[i]]++;
        comp->copy++;
    }

    for (i = 1; i <= max; i++)
    {
        count[i] += count[i - 1];
        comp->copy++;
    }

    for (i = tam - 1; i >= 0; i--)
    {
        arrB[count[arrA[i]] - 1] = arrA[i];
        count[arrA[i]]--;
        comp->copy++;
    }
}
```

E-Counting Sort

Retirado do artigo “Implementing and Analyzing an Efficient Version of Counting Sort (E-Counting Sort)”, o algoritmo E-Counting Sort sugere uma nova versão mais eficiente do algoritmo.

Algoritmo E-**COUNTING SORT**

A =

3	1	1	2	4	2	0	3
0	1	2	3	4	5	6	7

C =

1	2	2	2	1	0
0	1	2	3	4	5

Algoritmo E-COUNTING SORT

```
 j = 0
for(int i = 0; i <= max; i++)
  while(count[i] > 0)
    arrB[j] = i
    j++
    count[i]--
```

B =

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7

C =

1	2	2	2	0	1
0	1	2	3	4	5

Algoritmo E-COUNTING SORT

```
j = 0
for(int i = 0; i <= max; i++)
  while(count[i] > 0)
    arrB[j] = i
    j++
    count[i]--
```

B =

0	1	1	0	0	0	0	0
0	1	2	3	4	5	6	7

C =

1->0	2->1->0	2	2	0	1
0	1	2	3	4	5

Algoritmo E-COUNTING SORT

```
j = 0
for(int i = 0; i <= max; i++)
  while(count[i] > 0)
    arrB[j] = i
    j++
    count[i]--
```

B =

0	1	1	2	2	0	0	0

C =

0	0	2->1->0	2	0	1
0	1	2	3	4	5

Código do E-Counting Sort

E-COUNTING SORT ALGORITHM

COUNTING_MODIFY_SORT (A, B, k)

```
1.      for i← 1 to k
2.        c[i] ←0;
3.      for i← 1 to n
4.        c[a[i]]=c[a[i]]+1;
5.      for i← 1 to k
6.        while(c[i]>0)
7.          a[j]=i;
8.          j=j+1;
9.          c[i]=c[i]-1
```

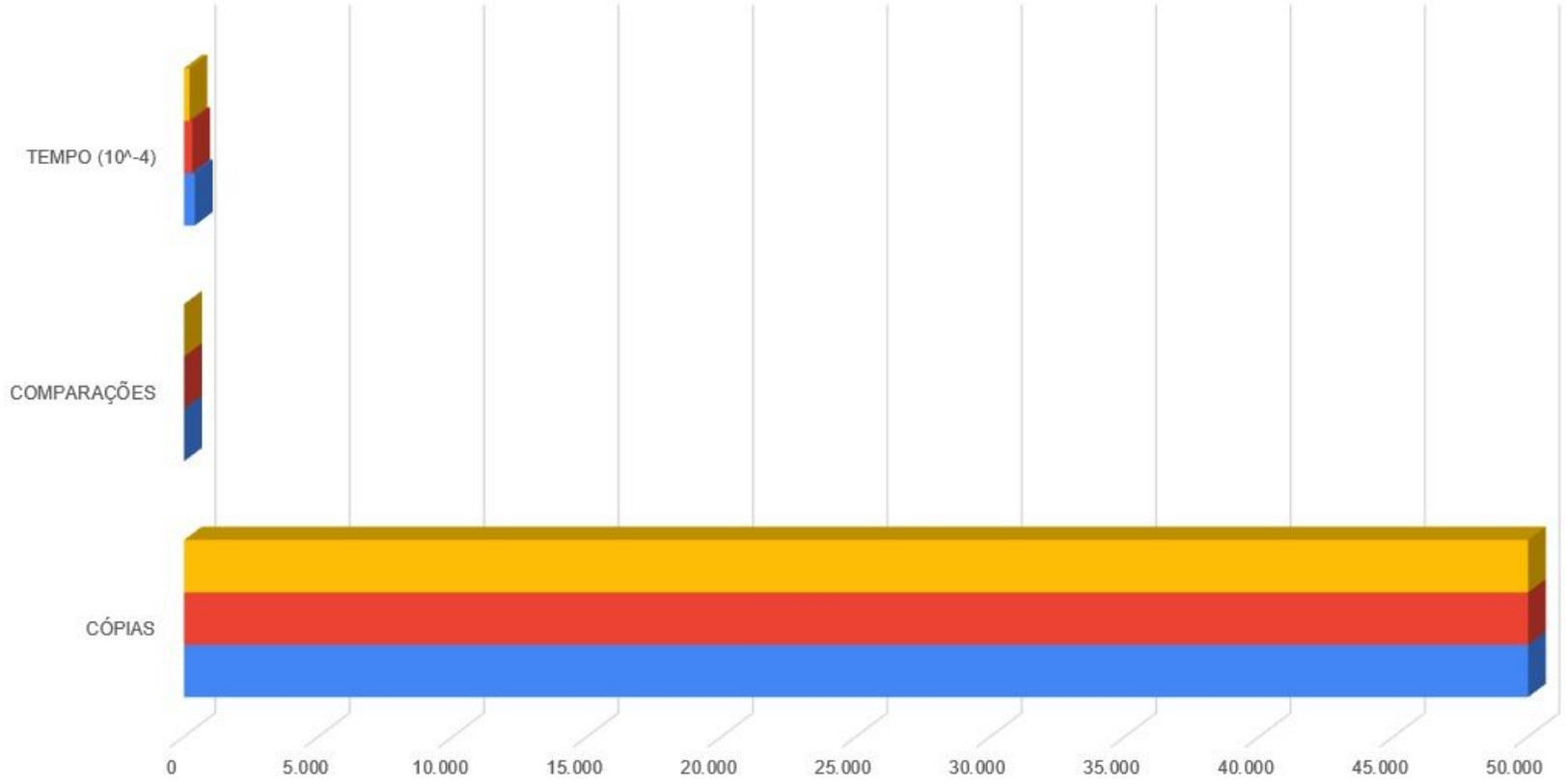
Código do E-Counting Sort

```c
void eCountingSort(int *arrA, int *arrB, int max, int tam, data *d){
    int i, j = 0;
    int* count = calloc(max + 1, sizeof(int));

    for (i = 0; i < tam; i++){
        count[arrA[i]]++;
        d->copy++;
    }

    for (i = 0; i <= max; i++){
        while(count[i] > 0){
            arrB[j] = i;
            j++;
            count[i]--;
            d->copy++;
            d->comp++;
        }
    }
}
```

Resultados

Comparações feitas entre os dois algoritmos com diferentes tamanhos de vetores desordenados, ordenados e decrescentes.

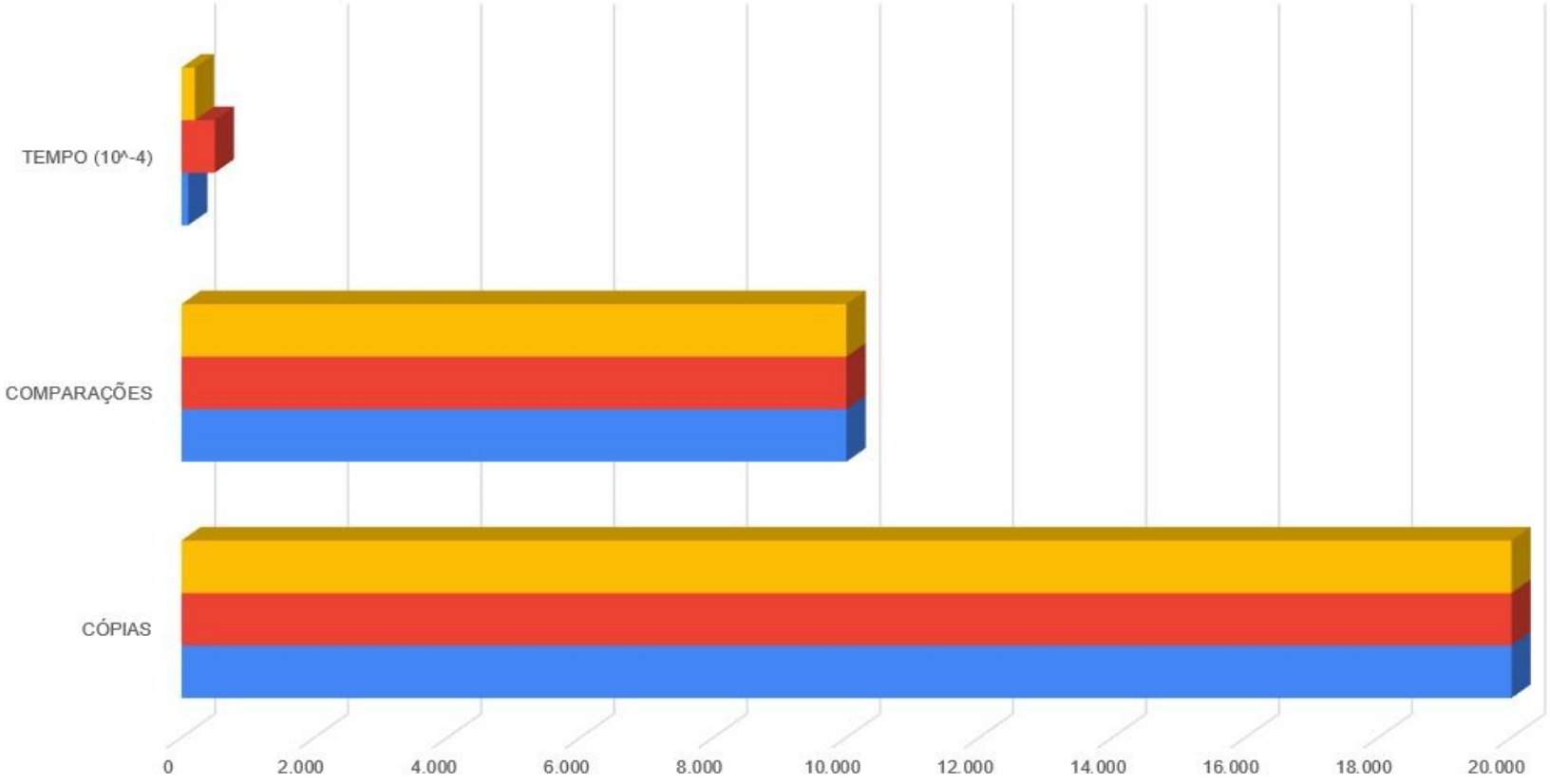
COUNTING - GRÁFICO DO TESTE COM 10K NÚMEROS



	CÓPIAS	COMPARAÇÕES	TEMPO (10^-4)
DECRESCENTE	49.995	0	200
ALEATÓRIO	49.995	0	300
ORDENADO	49.995	0	400

DECRESCENTE ALEATÓRIO ORDENADO

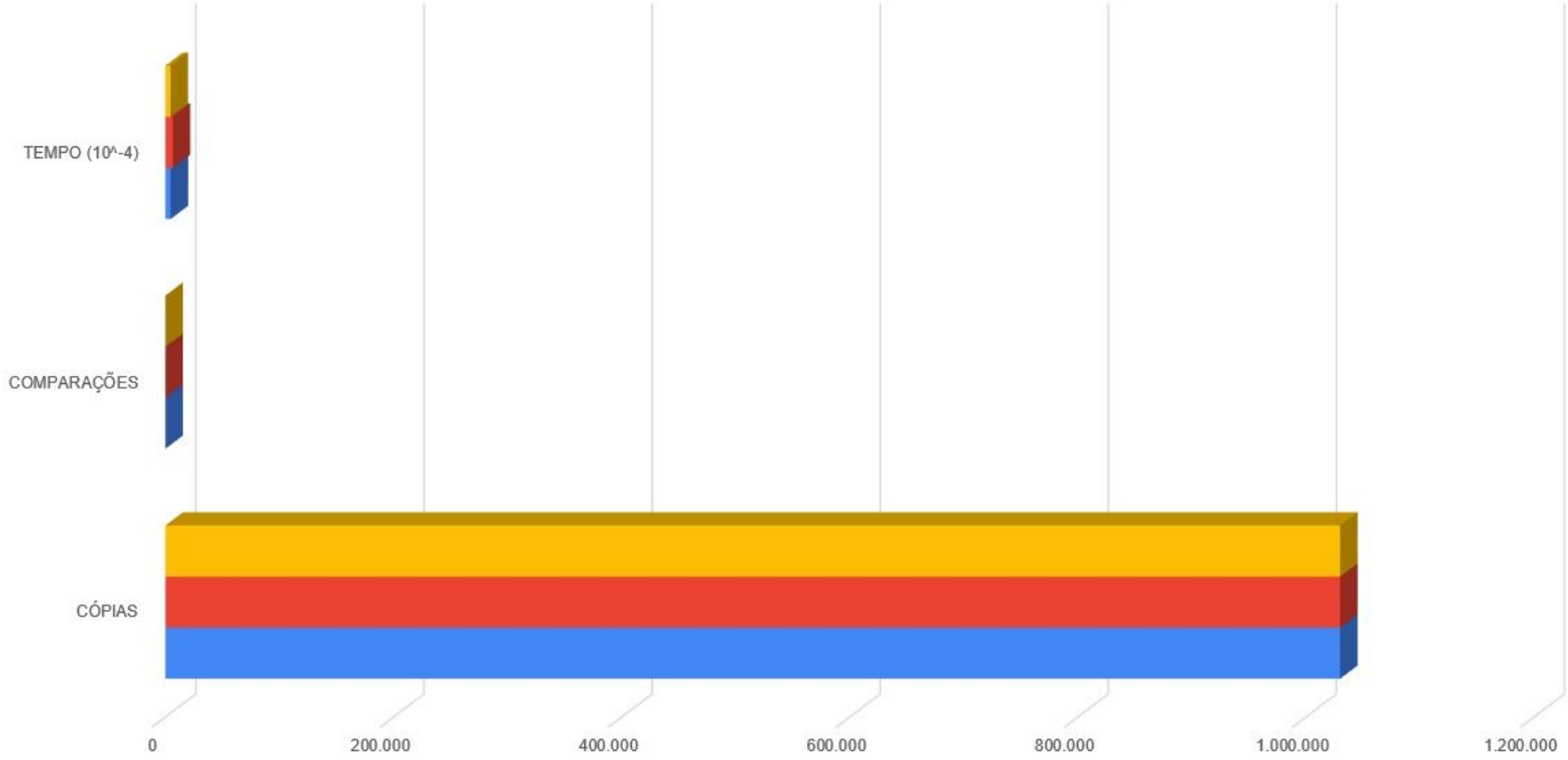
E-COUNTING - GRÁFICO DO TESTE COM 10K NÚMEROS



	CÓPIAS	COMPARAÇÕES	TEMPO (10^-4)
DECRESCENTE	20.000	10.000	200
ALEATÓRIO	20.000	10.000	500
ORDENADO	20.000	10.000	100

DECRESCENTE ALEATÓRIO ORDENADO

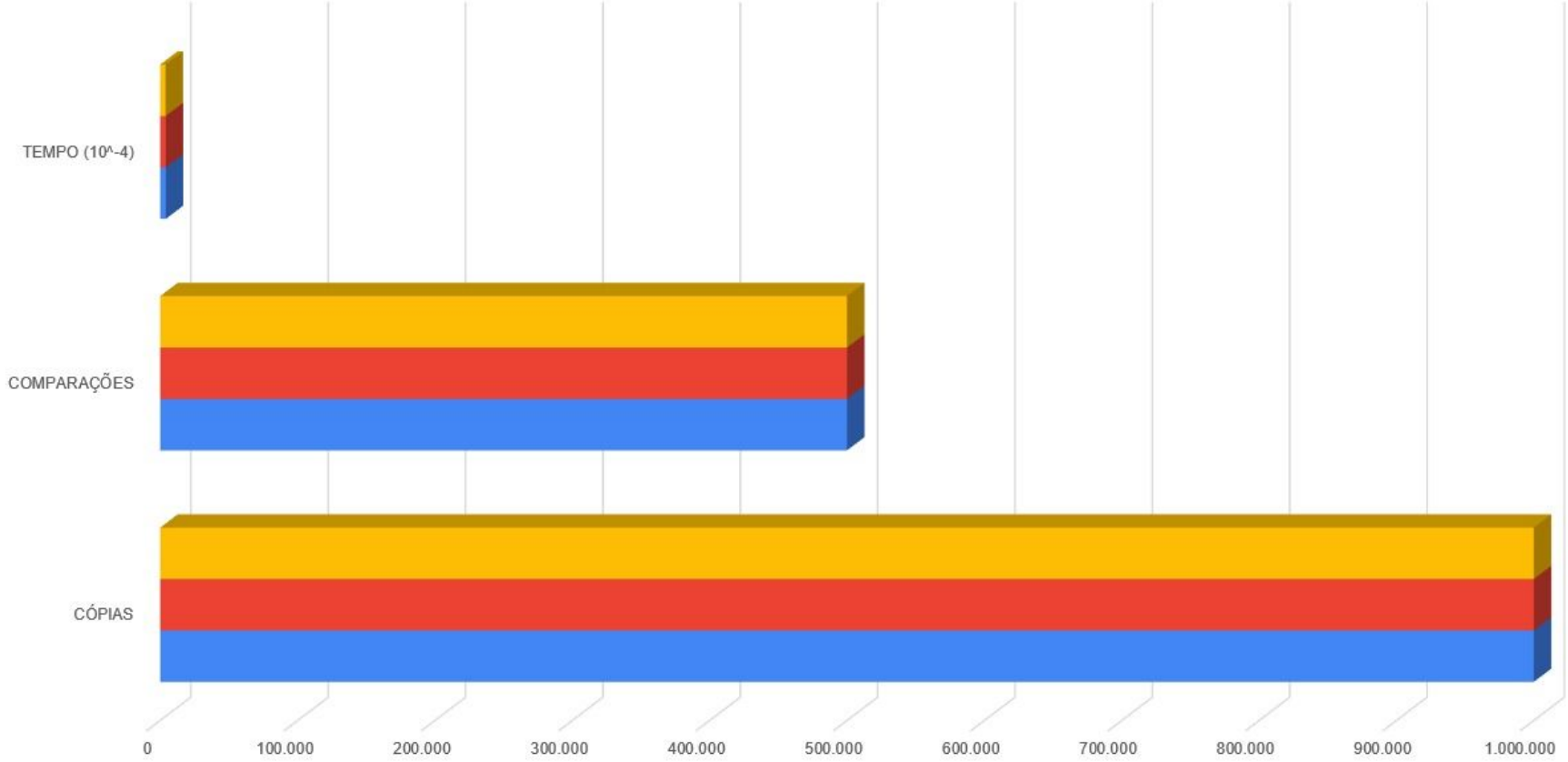
COUNTING - GRÁFICO DO TESTE COM 500K NÚMEROS



	CÓPIAS	COMPARAÇÕES	TEMPO (10^-4)
DECRESCENTE	1.029.999	0	4500
ALEATÓRIO	1.029.999	0	6500
ORDENADO	1.029.999	0	4700

DECRESCENTE ALEATÓRIO ORDENADO

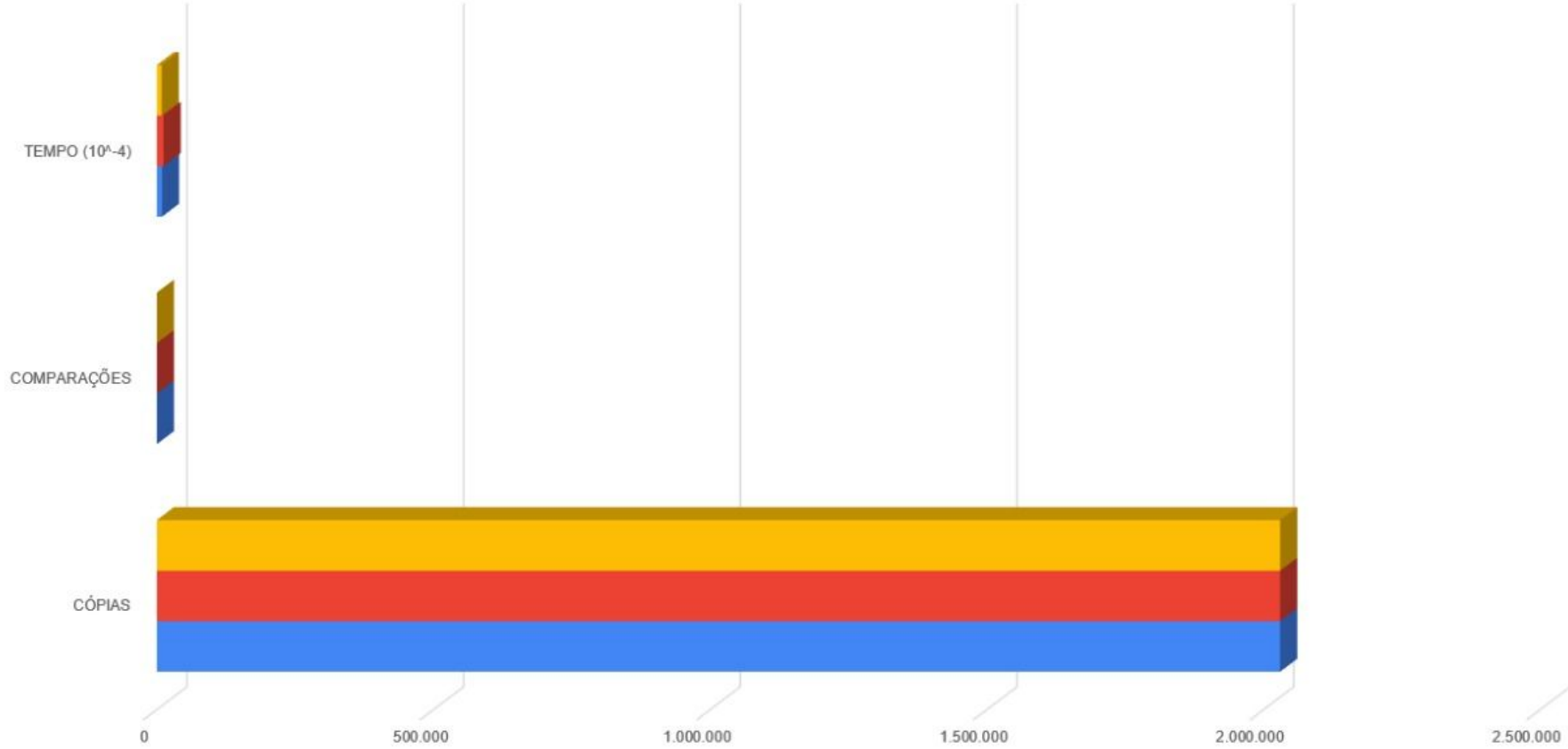
E-COUNTING - GRÁFICO DO TESTE COM 500K NÚMEROS



	CÓPIAS	COMPARAÇÕES	TEMPO (10^-4)
DECRESCENTE	1.000.000	500.000	3900
ALEATÓRIO	1.000.000	500.000	4100
ORDENADO	1.000.000	500.000	4100

DECRESCENTE ALEATÓRIO ORDENADO

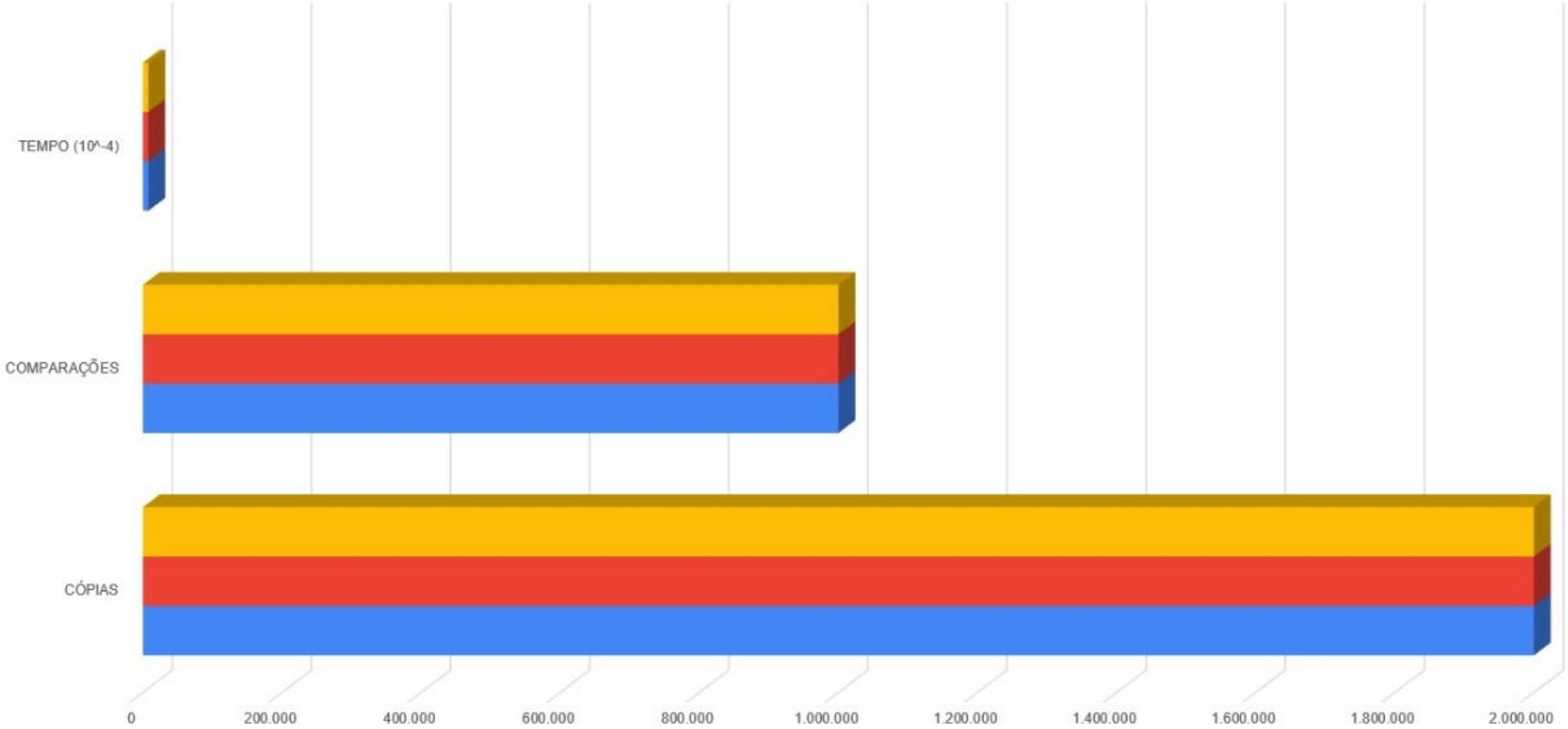
COUNTING - GRÁFICO DO TESTE COM 1M NÚMEROS



	CÓPIAS	COMPARAÇÕES	TEMPO (10^-4)
DECRESCENTE	2.029.999	0	8800
ALEATÓRIO	2.029.999	0	12100
ORDENADO	2.029.999	0	9400

DECRESCENTE ALEATÓRIO ORDENADO

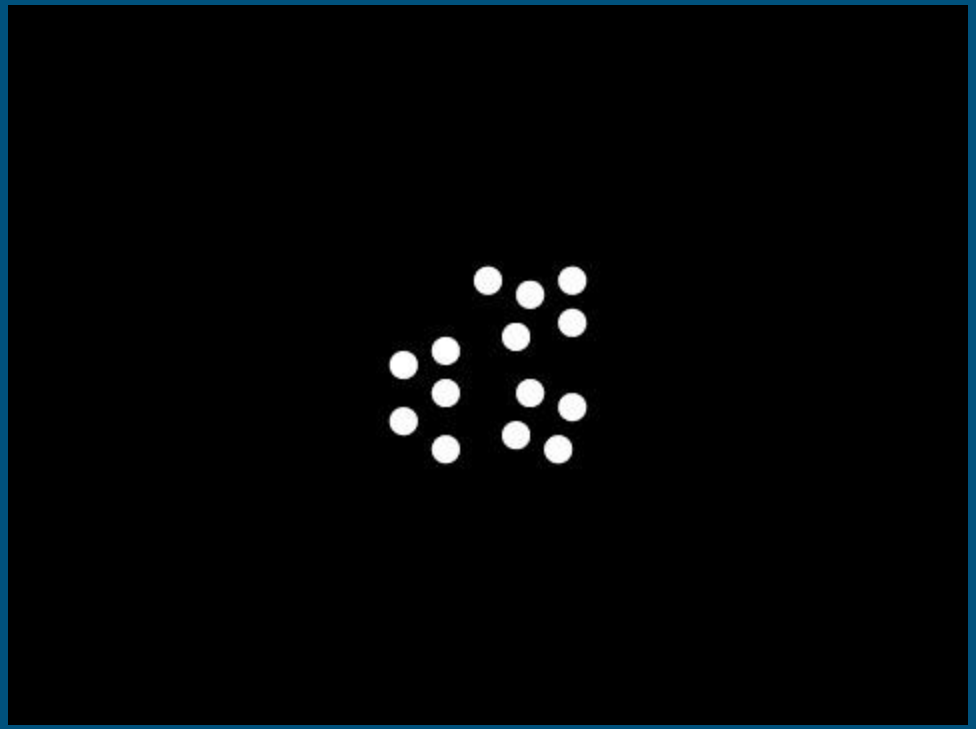
E-COUNTING - GRÁFICO DO TESTE COM 1M NÚMEROS



	CÓPIAS	COMPARAÇÕES	TEMPO (10^-4)
DECRESCENTE	2.000.000	1.000.000	7800
ALEATÓRIO	2.000.000	1.000.000	7600
ORDENADO	2.000.000	1.000.000	7900

DECRESCENTE ALEATÓRIO ORDENADO

Vídeo e documentação



https://www.overleaf.com/read/dddvvgjcjhxv