# Aula passada:

Todo algoritmo de ordenação
baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso.

## Aula passada:

Todo algoritmo de ordenação
baseado em comparações faz

$\Omega(n \lg n)$

comparações no pior caso.

## Aula de hoje:

Algoritmos de ordenação lineares!

# Ordenação em tempo linear

CLRS cap 8

# Counting Sort

Recebe inteiros $n$ e $k$, e um vetor $A[1\,..\,n]$
onde cada elemento é um inteiro entre 1 e $k$.

# Counting Sort

Recebe inteiros $n$ e $k$, e um vetor $A[1\,..\,n]$
onde cada elemento é um inteiro entre 1 e $k$.

Devolve um vetor $B[1\,..\,n]$ com
os elementos de $A[1\,..\,n]$ em ordem crescente.

# Counting Sort

Recebe inteiros $n$ e $k$, e um vetor $A[1\,..\,n]$
onde cada elemento é um inteiro entre 1 e $k$.

Devolve um vetor $B[1\,..\,n]$ com
os elementos de $A[1\,..\,n]$ em ordem crescente.

```
CountingSort(A, n)
1  para i ← 1 até k faça
2      C[i] ← 0
3  para j ← 1 até n faça
4      C[A[j]] ← C[A[j]] + 1
```

# Counting Sort

Recebe inteiros $n$ e $k$, e um vetor $A[1\,..\,n]$
onde cada elemento é um inteiro entre 1 e $k$.

Devolve um vetor $B[1\,..\,n]$ com
os elementos de $A[1\,..\,n]$ em ordem crescente.

```
CountingSort(A, n)
1  para i ← 1 até k faça
2      C[i] ← 0
3  para j ← 1 até n faça
4      C[A[j]] ← C[A[j]] + 1
5  para i ← 2 até k faça
6      C[i] ← C[i] + C[i − 1]
```

# Counting Sort

Recebe inteiros $n$ e $k$, e um vetor $A[1\,..\,n]$
onde cada elemento é um inteiro entre 1 e $k$.

Devolve um vetor $B[1\,..\,n]$ com
os elementos de $A[1\,..\,n]$ em ordem crescente.

```
CountingSort(A, n)
 1  para i ← 1 até k faça
 2      C[i] ← 0
 3  para j ← 1 até n faça
 4      C[A[j]] ← C[A[j]] + 1
 5  para i ← 2 até k faça
 6      C[i] ← C[i] + C[i − 1]
 7  para j ← n decrescendo até 1 faça
 8      B[C[A[j]]] ← A[j]
 9      C[A[j]] ← C[A[j]] − 1
10  devolva B
```

# Consumo de tempo

| linha | consumo na linha |
|---|---|
| 1 | $\Theta(k)$ |
| 2 | $\mathrm{O}(k)$ |
| 3 | $\Theta(n)$ |
| 4 | $\mathrm{O}(n)$ |
| 5 | $\Theta(k)$ |
| 6 | $\mathrm{O}(k)$ |
| 7 | $\Theta(n)$ |
| 8 | $\mathrm{O}(n)$ |
| 9 | $\mathrm{O}(n)$ |
| 10 | $\Theta(1)$ |
| total | ???? |

# Consumo de tempo

| linha | consumo na linha |
|---|---|
| 1 | $\Theta(k)$ |
| 2 | $\mathrm{O}(k)$ |
| 3 | $\Theta(n)$ |
| 4 | $\mathrm{O}(n)$ |
| 5 | $\Theta(k)$ |
| 6 | $\mathrm{O}(k)$ |
| 7 | $\Theta(n)$ |
| 8 | $\mathrm{O}(n)$ |
| 9 | $\mathrm{O}(n)$ |
| 10 | $\Theta(1)$ |
| total | $\Theta(k+n)$ |

# Counting Sort

```
CountingSort(A, n)
 1  para i ← 1 até k faça
 2      C[i] ← 0
 3  para j ← 1 até n faça
 4      C[A[j]] ← C[A[j]] + 1
 5  para i ← 2 até k faça
 6      C[i] ← C[i] + C[i − 1]
 7  para j ← n decrescendo até 1 faça
 8      B[C[A[j]]] ← A[j]
 9      C[A[j]] ← C[A[j]] − 1
10  devolva B
```

Consumo de tempo: $\Theta(k + n)$

Se $k = \mathrm{O}(n)$, então o consumo de tempo é $\Theta(n)$.

# Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com $d$ dígitos
- cartões perfurados
- registros cuja chave tem vários campos

# Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com $d$ dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito $d$: mais significativo

# Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com $d$ dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito $d$: mais significativo

```
RadixSort(A, n, d)
1  para i ← 1 até d faça
2      ORDENE(A, n, i)
```

# Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com $d$ dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito $d$: mais significativo

```
RadixSort(A, n, d)
1  para i ← 1 até d faça
2      ORDENE(A, n, i)
```

ORDENE$(A, n, i)$: ordena $A[1\,..\,n]$ pelo $i$-ésimo dígito dos números em $A$ por meio de um algoritmo estável.

# Estabilidade

Um algoritmo de ordenação é estável se sempre que, inicialmente, $A[i] = A[j]$ para $i < j$, a cópia $A[i]$ termina em uma posição menor do vetor que a cópia $A[j]$.

# Estabilidade

Um algoritmo de ordenação é estável se sempre que, inicialmente, $A[i] = A[j]$ para $i < j$, a cópia $A[i]$ termina em uma posição menor do vetor que a cópia $A[j]$.

Isso só é relevante quando temos informação satélite.

# Estabilidade

Um algoritmo de ordenação é estável se sempre que, inicialmente, $A[i] = A[j]$ para $i < j$, a cópia $A[i]$ termina em uma posição menor do vetor que a cópia $A[j]$.

Isso só é relevante quando temos informação satélite.

Quais dos algoritmos que vimos são estáveis?

# Estabilidade

Um algoritmo de ordenação é estável se sempre que, inicialmente, $A[i] = A[j]$ para $i < j$, a cópia $A[i]$ termina em uma posição menor do vetor que a cópia $A[j]$.

Isso só é relevante quando temos informação satélite.

Quais dos algoritmos que vimos são estáveis?

- inserção direta? seleção direta? bubblesort?
- mergesort?
- quicksort?
- heapsort?
- countingsort?

# Consumo de tempo do Radixsort

Depende do algoritmo ORDENE.

# Consumo de tempo do Radixsort

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a $k$,
então podemos usar o CountingSort.

# Consumo de tempo do Radixsort

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a $k$,
então podemos usar o CountingSort.

Neste caso, o consumo de tempo é $\Theta(d(k+n))$.

# Consumo de tempo do Radixsort

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a $k$,
então podemos usar o CountingSort.

Neste caso, o consumo de tempo é $\Theta(d(k+n))$.

Se $d$ é limitado por uma constante (ou seja, se $d = \mathrm{O}(1)$)
e $k = \mathrm{O}(n)$, então o consumo de tempo é $\Theta(n)$.

# Bucket Sort

Recebe um inteiro $n$ e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

# Bucket Sort

Recebe um inteiro $n$ e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

| $A$ | .47 | .93 | .82 | .12 | .42 | .03 | .62 | .38 | .77 | .91 |
|---|---|---|---|---|---|---|---|---|---|---|

# Bucket Sort

Recebe um inteiro $n$ e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

| $A$ | .47 | .93 | .82 | .12 | .42 | .03 | .62 | .38 | .77 | .91 |
|---|---|---|---|---|---|---|---|---|---|---|

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

# Bucket Sort

Recebe um inteiro $n$ e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

| $A$ | .47 | .93 | .82 | .12 | .42 | .03 | .62 | .38 | .77 | .91 |
|---|---|---|---|---|---|---|---|---|---|---|

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

| $C$ | .03 | .12 | .38 | .42 | .47 | .62 | .77 | .82 | .91 | .93 |
|---|---|---|---|---|---|---|---|---|---|---|

# Exemplo

| .47 | .93 | .82 | .12 | .42 | .03 | .62 | .38 | .77 | .91 |
|---|---|---|---|---|---|---|---|---|---|

# Exemplo

| .47 | .93 | .82 | .12 | .42 | .03 | .62 | .38 | .77 | .91 |
|---|---|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| $B[0]$ : | .03 | |
| $B[1]$ : | .12 | |
| $B[2]$ : | | |
| $B[3]$ : | .38 | |
| $B[4]$ : | .47 | .42 |
| $B[5]$ : | | |
| $B[6]$ : | .62 | |
| $B[7]$ : | .77 | |
| $B[8]$ : | .82 | |
| $B[9]$ : | .93 | .91 |

# Exemplo

| .47 | .93 | .82 | .12 | .42 | .03 | .62 | .38 | .77 | .91 |
|---|---|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| $B[0]$ : | .03 | |
| $B[1]$ : | .12 | |
| $B[2]$ : | | |
| $B[3]$ : | .38 | |
| $B[4]$ : | .42 | .47 |
| $B[5]$ : | | |
| $B[6]$ : | .62 | |
| $B[7]$ : | .77 | |
| $B[8]$ : | .82 | |
| $B[9]$ : | .91 | .93 |

# Exemplo

| .47 | .93 | .82 | .12 | .42 | .03 | .62 | .38 | .77 | .91 |
|---|---|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| $B[0]$ : | .03 | |
| $B[1]$ : | .12 | |
| $B[2]$ : | | |
| $B[3]$ : | .38 | |
| $B[4]$ : | .42 | .47 |
| $B[5]$ : | | |
| $B[6]$ : | .62 | |
| $B[7]$ : | .77 | |
| $B[8]$ : | .82 | |
| $B[9]$ : | .91 | .93 |

| .03 | .12 | .38 | .42 | .47 | .62 | .77 | .82 | .91 | .93 |
|---|---|---|---|---|---|---|---|---|---|

# Bucket Sort

Recebe um inteiro $n$ e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

# Bucket Sort

Recebe um inteiro $n$ e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

```
BucketSort(A, n)
1  para i ← 0 até n − 1 faça
2      B[i] ← NIL
3  para i ← 1 até n faça
4      Insira(B[⌊n A[i]⌋], A[i])
5  para i ← 0 até n − 1 faça
6      OrdeneLista(B[i])
7  C ← Concatene(B, n)
8  devolva C
```

# Bucket Sort

```
BucketSort(A, n)
1  para i ← 0 até n − 1 faça
2      B[i] ← NIL
3  para i ← 1 até n faça
4      Insira(B[⌊n A[i]⌋], A[i])
5  para i ← 0 até n − 1 faça
6      OrdeneLista(B[i])
7  C ← Concatene(B, n)
8  devolva C
```

Insira$(p, x)$: insere $x$ na lista apontada por $p$

# Bucket Sort

```
BucketSort(A, n)
1  para i ← 0 até n − 1 faça
2      B[i] ← NIL
3  para i ← 1 até n faça
4      Insira(B[⌊n A[i]⌋], A[i])
5  para i ← 0 até n − 1 faça
6      OrdeneLista(B[i])
7  C ← Concatene(B, n)
8  devolva C
```

Insira$(p, x)$: insere $x$ na lista apontada por $p$

OrdeneLista$(p)$: ordena a lista apontada por $p$

# Bucket Sort

```
BucketSort(A, n)
1  para i ← 0 até n − 1 faça
2      B[i] ← NIL
3  para i ← 1 até n faça
4      Insira(B[⌊n A[i]⌋], A[i])
5  para i ← 0 até n − 1 faça
6      OrdeneLista(B[i])
7  C ← Concatene(B, n)
8  devolva C
```

Insira$(p, x)$: insere $x$ na lista apontada por $p$

OrdeneLista$(p)$: ordena a lista apontada por $p$

Concatene$(B, n)$: devolve a lista obtida da concatenação das listas apontadas por $B[0], \ldots, B[n-1]$.

# Consumo de tempo: caso médio

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

# Consumo de tempo: caso médio

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o OrdeneLista seja o InsertionSort.

Seja $X_i$ o número de elementos na lista $B[i]$.

# Consumo de tempo: caso médio

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o OrdeneLista seja o InsertionSort.

Seja $X_i$ o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

# Consumo de tempo: caso médio

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o OrdeneLista seja o InsertionSort.

Seja $X_i$ o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Observe que $X_i = \sum_j X_{ij}$.

# Consumo de tempo: caso médio

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o OrdeneLista seja o InsertionSort.

Seja $X_i$ o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Observe que $X_i = \sum_j X_{ij}$.

$Y_i$: número de comparações para ordenar a lista $B[i]$.

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \mathrm{E}[\sum_j {X_{ij}}^2 + \sum_j \sum_{k \neq j} X_{ij} X_{ik}] \end{aligned}$$

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \mathrm{E}[\sum_j {X_{ij}}^2] + \mathrm{E}[\sum_j \sum_{k \neq j} X_{ij} X_{ik}] \end{aligned}$$

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}] \end{aligned}$$

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$. Ademais,

$$\mathrm{E}[Y_i] \leq \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}].$$

Observe que ${X_{ij}}^2$ é uma variável aleatória binária.

# Consumo de tempo

$X_i$: número de elementos na lista $B[i]$

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{array} \right\}$$

$Y_i$: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$. Ademais,

$$\mathrm{E}[Y_i] \leq \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}].$$

Observe que ${X_{ij}}^2$ é uma variável aleatória binária.

Vamos calcular sua esperança:

$$\mathrm{E}[{X_{ij}}^2] = \Pr[{X_{ij}}^2 = 1] = \Pr[X_{ij} = 1] = \frac{1}{n}.$$

# Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro
note que $X_{ij}$ e $X_{ik}$ são variáveis aleatórias independentes.

# Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro
note que $X_{ij}$ e $X_{ik}$ são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

# Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro
note que $X_{ij}$ e $X_{ik}$ são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

Ademais, $\mathrm{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$.

# Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro note que $X_{ij}$ e $X_{ik}$ são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

Ademais, $\mathrm{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$.

Logo,

$$\begin{aligned}
\mathrm{E}[Y_i] &\leq \sum_j \frac{1}{n} + \sum_j \sum_{k \neq j} \frac{1}{n^2} \\
&= \frac{n}{n} + n(n-1)\frac{1}{n^2} \\
&= 1 + (n-1)\frac{1}{n} \\
&= 2 - \frac{1}{n}.
\end{aligned}$$

# Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

# Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que $Y$ é o número de comparações realizadas pelo BucketSort no total.

# Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que $Y$ é o número de comparações realizadas pelo BucketSort no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BucketSort.

# Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que $Y$ é o número de comparações realizadas pelo BucketSort no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BucketSort.

Mas então $\mathrm{E}[Y] = \sum_i \mathrm{E}[Y_i] \leq 2n - 1 = \mathrm{O}(n)$.

# Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que $Y$ é o número de comparações realizadas pelo BucketSort no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BucketSort.

Mas então $\mathrm{E}[Y] = \sum_i \mathrm{E}[Y_i] \leq 2n - 1 = \mathrm{O}(n)$.

O consumo de tempo esperado do BucketSort quando os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$ é $\mathrm{O}(n)$.

# Exercícios sobre cota inferior

**Exercício 7.A**
Desenhe a árvore de decisão para o SelectionSort aplicado a $A[1\,..\,3]$ com todos os elementos distintos.

**Exercício 7.B** [CLRS 8.1-1]
Qual a menor profundidade (= menor nível) que uma folha pode ter em uma árvore de decisão que descreve um algoritmo de ordenação baseado em comparações?

**Exercício 7.C** [CLRS 8.1-2]
Mostre que $\lg(n!) = \Omega(n \lg n)$ sem usar a fórmula de Stirling.
Sugestão: Calcule $\sum_{k=n/2}^{n} \lg k$. Use as técnicas de CLRS A.2.

# Exercícios

**Exercício 7.D** [CLRS 8.2-1]
Simule a execuçãao do CountingSort usando como entrada o vetor $A[1\,..\,11] = \langle 7, 1, 3, 1, 2, 4, 5, 7, 2, 4, 3 \rangle$.

**Exercício 7.E** [CLRS 8.2-2]
Mostre que o CountingSort é estável.

**Exercício 7.F** [CLRS 8.2-3]
Suponha que o **para** da linha 7 do CountingSort é substituído por

```
7    para j ← 1 até n faça
```

Mostre que o ainda funciona. O algoritmo resultante continua estável?