Algoritmos e Estruturas de Dados II

Algoritmos de Ordenação

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

Observações

- As palavras com a fonte `Courier` indicam as palavras-reservadas da linguagem de programação.

Referências

- **Algorithms in a Nutshell**. George T. Heineman, Gary Pollice, Stanley Selkow. O'Reilly Media, 2009.
- **Projetos de Algoritmos – com implementações em Pascal e C**. Nivio Ziviani. 2ª edição. Thomson, 2005.

Algoritmos de Ordenação

Introdução

Introdução

- Algoritmos de ordenação organizam os elementos de uma lista em uma ordem (ascendente ou descendente).

Introdução

- Algoritmos de ordenação organizam os elementos de uma lista em uma ordem (ascendente ou descendente).
- Ordenação é uma das mais importantes categorias de algoritmos da Computação.

Introdução

- Algoritmos de ordenação organizam os elementos de uma lista em uma ordem (ascendente ou descendente).
- Ordenação é uma das mais importantes categorias de algoritmos da Computação.
- Frequentemente são usados para diminuir a complexidade de um problema.

Introdução

- Algoritmos de ordenação organizam os elementos de uma lista em uma ordem (ascendente ou descendente).
- Ordenação é uma das mais importantes categorias de algoritmos da Computação.
- Frequentemente são usados para diminuir a complexidade de um problema.
- Aplicam-se em problemas de banco de dados ou de recuperação de informação.

Classificação dos Algoritmos

Classificação dos Algoritmos

- Número de comparações

Classificação dos Algoritmos

- Número de comparações
- Número de trocas

Classificação dos Algoritmos

- Número de comparações
- Número de trocas
- Localização dos dados

Classificação dos Algoritmos

- Número de comparações
- Número de trocas
- Localização dos dados
  - Ordenação interna

Classificação dos Algoritmos

- Número de comparações
- Número de trocas
- Localização dos dados
  - Ordenação interna
    - Todos os dados estão em memória principal (RAM).

Classificação dos Algoritmos

- Número de comparações
- Número de trocas
- Localização dos dados
  - Ordenação interna
    - Todos os dados estão em memória principal (RAM).
  - Ordenação externa

Classificação dos Algoritmos

- Número de comparações
- Número de trocas
- Localização dos dados
  - Ordenação interna
    - Todos os dados estão em memória principal (RAM).
  - Ordenação externa
    - Memória principal não cabe todos os dados.

Classificação dos Algoritmos

- Número de comparações
- Número de trocas
- Localização dos dados
  - Ordenação interna
    - Todos os dados estão em memória principal (RAM).
  - Ordenação externa
    - Memória principal não cabe todos os dados.
    - Dados armazenados em memória secundária (disco).

Classificação dos Algoritmos

- Número de comparações
- Número de trocas
- Localização dos dados
  - Ordenação interna
    - Todos os dados estão em memória principal (RAM).
  - Ordenação externa
    - Memória principal não cabe todos os dados.
    - Dados armazenados em memória secundária (disco).
- Recursão

Classificação dos Algoritmos

Classificação dos Algoritmos

- Uso de memória

Classificação dos Algoritmos

- Uso de memória
  - *In place*: ordena sem usar memória adicional ou usando uma quantidade constante de memória adicional.

Classificação dos Algoritmos

- Uso de memória
  - *In place*: ordena sem usar memória adicional ou usando uma quantidade constante de memória adicional.
  - Alguns métodos precisam duplicar os dados.

Classificação dos Algoritmos

Classificação dos Algoritmos

- Adaptabilidade

Classificação dos Algoritmos

- Adaptabilidade
  - Um método é adaptável quando a sequência de operações realizadas depende da entrada.

Classificação dos Algoritmos

- Adaptabilidade
  - Um método é adaptável quando a sequência de operações realizadas depende da entrada.
  - Um método que sempre realiza as mesmas operações, independente da entrada, é não adaptável.

Classificação dos Algoritmos

Classificação dos Algoritmos

- Estabilidade
  - Método é estável se a ordem relativa dos registros com a mesma chave não se altera após a ordenação.

Classificação dos Algoritmos

- Estabilidade
  - Método é estável se a ordem relativa dos registros com a mesma chave não se altera após a ordenação.

Adams	A
Black	B
Brown	D
Jackson	B
Jones	D
Smith	A
Thompson	D
Washington	B
White	C
Wilson	C

Adams	A
Smith	A
Washington	B
Jackson	B
Black	B
White	C
Wilson	C
Thompson	D
Brown	D
Jones	D

Adams	A
Smith	A
Black	B
Jackson	B
Washington	B
White	C
Wilson	C
Brown	D
Jones	D
Thompson	D

Algoritmo da Bolha

Algoritmo da Bolha

- Também conhecido como *Bubble Sort*.

Algoritmo da Bolha

- Também conhecido como *Bubble Sort*.
- É o método mais simples de ordenação.

Algoritmo da Bolha

- Também conhecido como *Bubble Sort*.
- É o método mais simples de ordenação.
- Algoritmo

Algoritmo da Bolha

- Também conhecido como *Bubble Sort*.
- É o método mais simples de ordenação.
- Algoritmo
  - Compara elementos adjacentes.

Algoritmo da Bolha

- Também conhecido como *Bubble Sort*.
- É o método mais simples de ordenação.
- Algoritmo
  - Compara elementos adjacentes.
  - Os elementos são trocados, caso não estejam ordenados.

Algoritmo da Bolha

- Também conhecido como *Bubble Sort*.
- É o método mais simples de ordenação.
- Algoritmo
  - Compara elementos adjacentes.
  - Os elementos são trocados, caso não estejam ordenados.
  - Processo é repetido até a lista ficar ordenada.

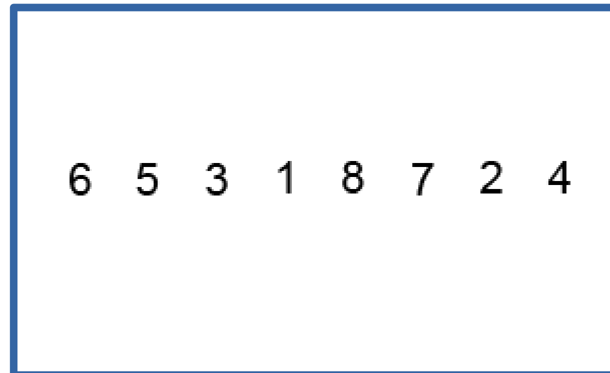
Algoritmo da Bolha

Algoritmo da Bolha

- Exemplo:

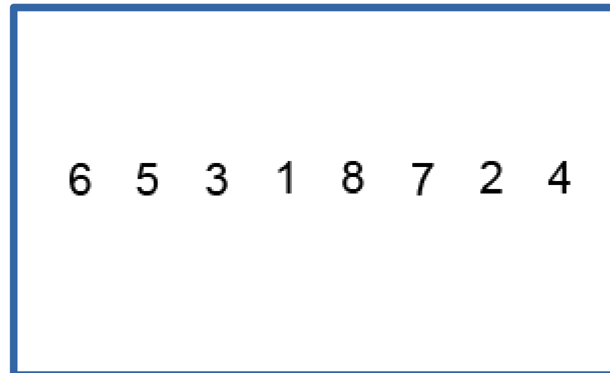
Algoritmo da Bolha

- Exemplo:



Algoritmo da Bolha

- Exemplo:



- O algoritmo tem complexidade $O(n^2)$.

Algoritmo da Bolha

Algoritmo da Bolha

- Pontos positivos

Algoritmo da Bolha

- Pontos positivos
  - Algoritmo simples.

Algoritmo da Bolha

- Pontos positivos
  - Algoritmo simples.
  - Algoritmo estável.

Algoritmo da Bolha

- Pontos positivos
  - Algoritmo simples.
  - Algoritmo estável.
- Pontos negativos

Algoritmo da Bolha

- Pontos positivos
  - Algoritmo simples.
  - Algoritmo estável.
- Pontos negativos
  - Não adaptável.

Algoritmo da Bolha

- Pontos positivos
  - Algoritmo simples.
  - Algoritmo estável.
- Pontos negativos
  - Não adaptável.
  - Muitas trocas de elementos durante a ordenação.

Algoritmo de Seleção

Algoritmo de Seleção

- Também conhecido como *Selection Sort*.

Algoritmo de Seleção

- Também conhecido como *Selection Sort*.
- É um algoritmo *in place*.

Algoritmo de Seleção

- Também conhecido como *Selection Sort*.
- É um algoritmo *in place*.
- Algoritmo:

Algoritmo de Seleção

- Também conhecido como *Selection Sort*.
- É um algoritmo *in place*.
- Algoritmo:
  - Achar o maior (ou menor) valor da lista.

Algoritmo de Seleção

- Também conhecido como *Selection Sort*.
- É um algoritmo *in place*.
- Algoritmo:
  - Achar o maior (ou menor) valor da lista.
  - Trocar esse elemento na posição atual.

Algoritmo de Seleção

- Também conhecido como *Selection Sort*.
- É um algoritmo *in place*.
- Algoritmo:
  - Achar o maior (ou menor) valor da lista.
  - Trocar esse elemento na posição atual.
  - Repetir o processo até a lista ficar ordenada.

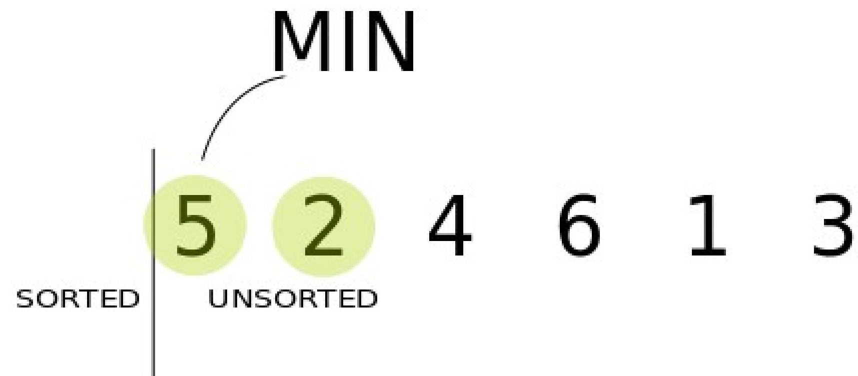
Algoritmo de Seleção

Algoritmo de Seleção

- Exemplo

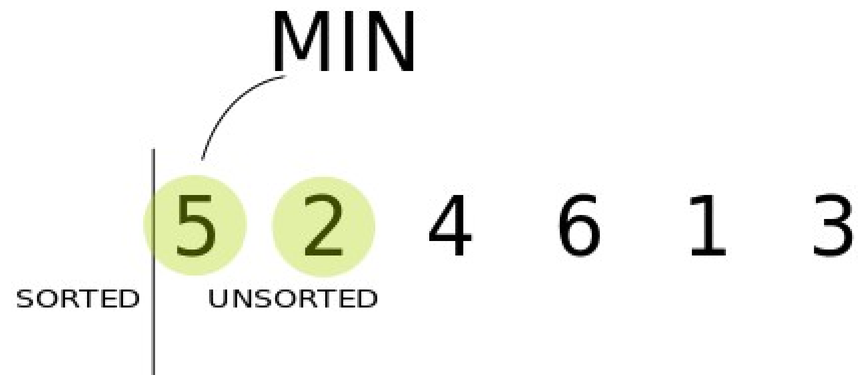
Algoritmo de Seleção

- Exemplo



Algoritmo de Seleção

- Exemplo



- O algoritmo tem complexidade $O(n^2)$.

Algoritmo de Seleção

Algoritmo de Seleção

- Pontos positivos

Algoritmo de Seleção

- Pontos positivos
  - Custo linear no tamanho da entrada para o número de movimentos de elementos.

Algoritmo de Seleção

- Pontos positivos
  - Custo linear no tamanho da entrada para o número de movimentos de elementos.
- Pontos negativos

Algoritmo de Seleção

- Pontos positivos
  - Custo linear no tamanho da entrada para o número de movimentos de elementos.
- Pontos negativos
  - Não adaptável.

Algoritmo de Seleção

- Pontos positivos
  - Custo linear no tamanho da entrada para o número de movimentos de elementos.
- Pontos negativos
  - Não adaptável.
  - Não estável.

Algoritmo de Inserção

Algoritmo de Inserção

- A cada iteração, o algoritmo remove um elemento da lista de entrada e insere esse elemento na posição correta.

Algoritmo de Inserção

Algoritmo de Inserção

- Exemplo

Algoritmo de Inserção

- Exemplo

6 5 3 1 8 7 2 4

Algoritmo de Inserção

- Exemplo

6 5 3 1 8 7 2 4

- O algoritmo tem complexidade $O(n^2)$.

Algoritmo de Inserção

Algoritmo de Inserção

- Características:

Algoritmo de Inserção

- Características:
  - Simples implementação.

Algoritmo de Inserção

- Características:
  - Simples implementação.
  - A ordenação é realizada *in place*. Requer apenas O(1) de espaço extra de memória.

Algoritmo de Inserção

- Características:
  - Simples implementação.
  - A ordenação é realizada *in place*. Requer apenas O(1) de espaço extra de memória.
  - Estável.

Algoritmo de Inserção

- Características:
  - Simples implementação.
  - A ordenação é realizada *in place*. Requer apenas O(1) de espaço extra de memória.
  - Estável.
  - Na prática, é mais eficiente do que o método Bolha e Seleção.

Algoritmo de Inserção

- Características:
  - Simples implementação.
  - A ordenação é realizada *in place*. Requer apenas O(1) de espaço extra de memória.
  - Estável.
  - Na prática, é mais eficiente do que o método Bolha e Seleção.
  - Algoritmo usado quando a entrada está praticamente ordenada ou quando a entrada é pequena.

Algoritmo de Inserção

Algoritmo de Inserção

- Pontos positivos

Algoritmo de Inserção

- Pontos positivos
  - Adequado para ordenar vetores pequenos.

Algoritmo de Inserção

- Pontos positivos
  - Adequado para ordenar vetores pequenos.
  - É o método a ser utilizado quando a lista está quase ordenada.

Algoritmo de Inserção

- Pontos positivos
  - Adequado para ordenar vetores pequenos.
  - É o método a ser utilizado quando a lista está quase ordenada.
  - É um bom método quando se deseja adicionar poucos elementos a uma lista ordenada (custo linear).

Algoritmo de Inserção

- Pontos positivos
  - Adequado para ordenar vetores pequenos.
  - É o método a ser utilizado quando a lista está quase ordenada.
  - É um bom método quando se deseja adicionar poucos elementos a uma lista ordenada (custo linear).
  - É estável.

Algoritmo de Inserção

- Pontos positivos
  - Adequado para ordenar vetores pequenos.
  - É o método a ser utilizado quando a lista está quase ordenada.
  - É um bom método quando se deseja adicionar poucos elementos a uma lista ordenada (custo linear).
  - É estável.
- Pontos negativos

Algoritmo de Inserção

- Pontos positivos
  - Adequado para ordenar vetores pequenos.
  - É o método a ser utilizado quando a lista está quase ordenada.
  - É um bom método quando se deseja adicionar poucos elementos a uma lista ordenada (custo linear).
  - É estável.
- Pontos negativos
  - Número de comparações tem crescimento quadrático.

Algoritmo de Inserção

- Pontos positivos
  - Adequado para ordenar vetores pequenos.
  - É o método a ser utilizado quando a lista está quase ordenada.
  - É um bom método quando se deseja adicionar poucos elementos a uma lista ordenada (custo linear).
  - É estável.
- Pontos negativos
  - Número de comparações tem crescimento quadrático.
  - Alto custo de movimentação de elementos na lista.

Mergesort

Mergesort

- Estratégia de divisão e conquista

Mergesort

- Estratégia de divisão e conquista
- Algoritmo baseado em dois passos:

Mergesort

- Estratégia de divisão e conquista
- Algoritmo baseado em dois passos:
  - Divide a lista de entrada em N sublistas, onde cada sublista tem 1 elemento não ordenado e N é o número de elementos da lista.

Mergesort

- Estratégia de divisão e conquista
- Algoritmo baseado em dois passos:
  - Divide a lista de entrada em N sublistas, onde cada sublista tem 1 elemento não ordenado e N é o número de elementos da lista.
  - Repetidamente junta (*merge*) as sublistas para ir produzindo novas sublistas ordenadas até que todos os elementos sejam agrupados (*merge*) em uma única lista ordenada.

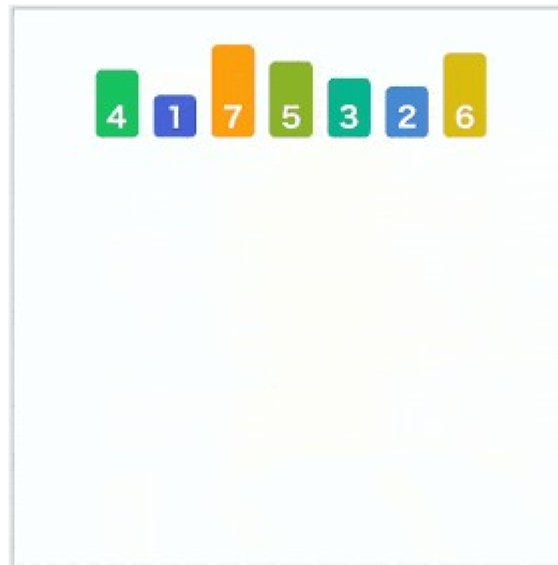
Mergesort

Mergesort

- Exemplo

Mergesort

- Exemplo



Quicksort

Quicksort

- É outro algoritmo baseado na estratégia de divisão e conquista.

Quicksort

- É outro algoritmo baseado na estratégia de divisão e conquista.
- Apesar dele ser um pouco mais complicado, na maioria das implementações ele é mais eficiente do que o Mergesort e raramente alcança o pior caso de $O(n^2)$.

Quicksort

Quicksort

- Algoritmo

Quicksort

- Algoritmo
  - Seleciona-se um elemento da lista chamado de *pivot*.

Quicksort

- Algoritmo
  - Seleciona-se um elemento da lista chamado de *pivot*.
  - Executa-se a operação de partição

Quicksort

- Algoritmo
  - Seleciona-se um elemento da lista chamado de *pivot*.
  - Executa-se a operação de partição
    - Os elementos menores que o *pivot* ficarão a sua esquerda.

Quicksort

- Algoritmo
  - Seleciona-se um elemento da lista chamado de *pivot*.
  - Executa-se a operação de partição
    - Os elementos menores que o *pivot* ficarão a sua esquerda.
    - Os elementos maiores que o *pivot* ficarão a sua direita.

Quicksort

- Algoritmo
  - Seleciona-se um elemento da lista chamado de *pivot*.
  - Executa-se a operação de partição
    - Os elementos menores que o *pivot* ficarão a sua esquerda.
    - Os elementos maiores que o *pivot* ficarão a sua direita.
  - De modo recursivo, aplica-se os passos anteriores separadamente para cada sublista onde o último *pivot* é o elemento central.

Quicksort

- Exemplo

Quicksort

- Exemplo

6 5 3 1 8 7 2 4

ShellSort

ShellSort

- Proposto por Donald Shell em 1959.

ShellSort

- Proposto por Donald Shell em 1959.
- É uma extensão do algoritmo de ordenação por inserção.

ShellSort

- Proposto por Donald Shell em 1959.
- É uma extensão do algoritmo de ordenação por inserção.
- Ordenação por inserção só troca elementos adjacentes para determinar o ponto de inserção.

ShellSort

- Proposto por Donald Shell em 1959.
- É uma extensão do algoritmo de ordenação por inserção.
- Ordenação por inserção só troca elementos adjacentes para determinar o ponto de inserção.
- O método ShellSort permite fazer trocas de elementos distantes.

ShellSort

ShellSort

- Algoritmo

ShellSort

- Algoritmo
  - Os conjuntos de elementos separados de `h` posições são ordenados.

ShellSort

- Algoritmo
  - Os conjuntos de elementos separados de `h` posições são ordenados.
    - O elemento na posição `x` é comparado (e trocado) com o elemento da posição `x−h`.

ShellSort

- Algoritmo
  - Os conjuntos de elementos separados de `h` posições são ordenados.
    - O elemento na posição `x` é comparado (e trocado) com o elemento da posição `x−h`.
    - A lista resultante é composta de `h` segmentos ordenados e entrelaçados.

ShellSort

- Algoritmo
  - Os conjuntos de elementos separados de `h` posições são ordenados.
    - O elemento na posição `x` é comparado (e trocado) com o elemento da posição `x−h`.
    - A lista resultante é composta de `h` segmentos ordenados e entrelaçados.
  - A lista é dita estar h-ordenada.

ShellSort

- Algoritmo
  - Os conjuntos de elementos separados de `h` posições são ordenados.
    - O elemento na posição `x` é comparado (e trocado) com o elemento da posição `x−h`.
    - A lista resultante é composta de `h` segmentos ordenados e entrelaçados.
  - A lista é dita estar h-ordenada.
  - Quando `h = 1` o algoritmo é equivalente ao método de inserção.

ShellSort

- Exemplo:

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a0	a11	a12
Entrada	*62*	*83*	*18*	*53*	*07*	*17*	*95*	*86*	*47*	*69*	*25*	*28*
H = 5	17	28	18	47	07	25	83	86	53	69	62	95
H = 3	17	07	18	47	28	25	69	62	53	83	86	95
H = 1	07	17	18	25	28	47	53	62	69	83	86	95

ShellSort

ShellSort

- Escolha da distância de salto (h)

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).
  - Forte impacto no desempenho do algoritmo.

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).
  - Forte impacto no desempenho do algoritmo.
  - Sequência para h:

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).
  - Forte impacto no desempenho do algoritmo.
  - Sequência para h:
    - h(s) = 1, para s = 1

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).
  - Forte impacto no desempenho do algoritmo.
  - Sequência para h:
    - $h(s) = 1$, para $s = 1$
    - $h(s) = 3h(s-1) + 1$, para $s > 1$

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).
  - Forte impacto no desempenho do algoritmo.
  - Sequência para h:
    - $h(s) = 1$, para $s = 1$
    - $h(s) = 3h(s-1) + 1$, para $s > 1$
    - A sequência corresponde a 1, 4, 13, 40, 121, 364, 1.093…

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).
  - Forte impacto no desempenho do algoritmo.
  - Sequência para h:
    - $h(s) = 1$, para $s = 1$
    - $h(s) = 3h(s-1) + 1$, para $s > 1$
    - A sequência corresponde a 1, 4, 13, 40, 121, 364, 1.093…
    - Knuth mostrou experimentalmente que essa sequência é difícil de ser batida por mais de 20% de eficiência.

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta (h = 1 é a ordenação por inserção).
  - Forte impacto no desempenho do algoritmo.
  - Sequência para h:
    - $h(s) = 1$, para $s = 1$
    - $h(s) = 3h(s-1) + 1$, para $s > 1$
    - A sequência corresponde a 1, 4, 13, 40, 121, 364, 1.093…
    - Knuth mostrou experimentalmente que essa sequência é difícil de ser batida por mais de 20% de eficiência.
    - Outras escolhas são possíveis.

ShellSort

ShellSort

- Escolha da distância de salto (h)

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta.

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta.
    - h = 1 é ordenação por inserção.

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta.
    - h = 1 é ordenação por inserção.
  - Forte impacto no desempenho do algoritmo.

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta.
    - h = 1 é ordenação por inserção.
  - Forte impacto no desempenho do algoritmo.
  - Exemplo de sequência ruim: 1, 2, 4, 8, 16…

ShellSort

- Escolha da distância de salto (h)
  - Qualquer sequência terminando com h = 1 garante a ordenação correta.
    - h = 1 é ordenação por inserção.
  - Forte impacto no desempenho do algoritmo.
  - Exemplo de sequência ruim: 1, 2, 4, 8, 16…
    - Não compara elementos em posições pares com elementos em posições ímpares até a última iteração.

ShellSort

ShellSort

- A complexidade do algoritmo ainda não é conhecida.

ShellSort

- A complexidade do algoritmo ainda não é conhecida.
- Ninguém ainda foi capaz de encontrar uma fórmula fechada para sua função de complexidade.

ShellSort

- A complexidade do algoritmo ainda não é conhecida.
- Ninguém ainda foi capaz de encontrar uma fórmula fechada para sua função de complexidade.
- A sua análise ainda contém alguns problemas matemáticos difíceis. Exemplo: escolher a sequência de incrementos.

ShellSort

- A complexidade do algoritmo ainda não é conhecida.
- Ninguém ainda foi capaz de encontrar uma fórmula fechada para sua função de complexidade.
- A sua análise ainda contém alguns problemas matemáticos difíceis. Exemplo: escolher a sequência de incrementos.
- O que se sabe é que cada incremento não deve ser múltiplo do anterior.

ShellSort

ShellSort

- Conjecturas referentes ao número de comparações para a sequência de Knuth:

ShellSort

- Conjecturas referentes ao número de comparações para a sequência de Knuth:
  - Conjectura 1: $C(n) = O(n^{1,25})$

ShellSort

- Conjecturas referentes ao número de comparações para a sequência de Knuth:
  - Conjectura 1: $C(n) = O(n^{1,25})$
  - Conjectura 2: $C(n) = O(n (\ln n)^2)$

ShellSort

ShellSort

- Pontos positivos

ShellSort

- Pontos positivos
  - É uma boa opção para arquivos de tamanhos moderados.

ShellSort

- Pontos positivos
  - É uma boa opção para arquivos de tamanhos moderados.
  - Implementação simples e requer uma quantidade de código pequena.

ShellSort

- Pontos positivos
  - É uma boa opção para arquivos de tamanhos moderados.
  - Implementação simples e requer uma quantidade de código pequena.
- Pontos negativos

ShellSort

- Pontos positivos
  - É uma boa opção para arquivos de tamanhos moderados.
  - Implementação simples e requer uma quantidade de código pequena.
- Pontos negativos
  - O tempo de execução do algoritmo é sensível à ordem inicial do arquivo.

ShellSort

- Pontos positivos
  - É uma boa opção para arquivos de tamanhos moderados.
  - Implementação simples e requer uma quantidade de código pequena.
- Pontos negativos
  - O tempo de execução do algoritmo é sensível à ordem inicial do arquivo.
  - O método não é estável.

Counting Sort

Counting Sort

- É um algoritmo de ordenação que não necessita da operação de comparação entre os elementos.

Counting Sort

- É um algoritmo de ordenação que não necessita da operação de comparação entre os elementos.
- Por isso, é considerado como algoritmo de ordenação inteira (*integer sorting algorithm*).

Counting Sort

- É um algoritmo de ordenação que não necessita da operação de comparação entre os elementos.
- Por isso, é considerado como algoritmo de ordenação inteira (*integer sorting algorithm*).
- Pode ser usado para ordenar uma lista de elementos cujos valores estão num intervalo pré-definido `[0,k)`.

Counting Sort

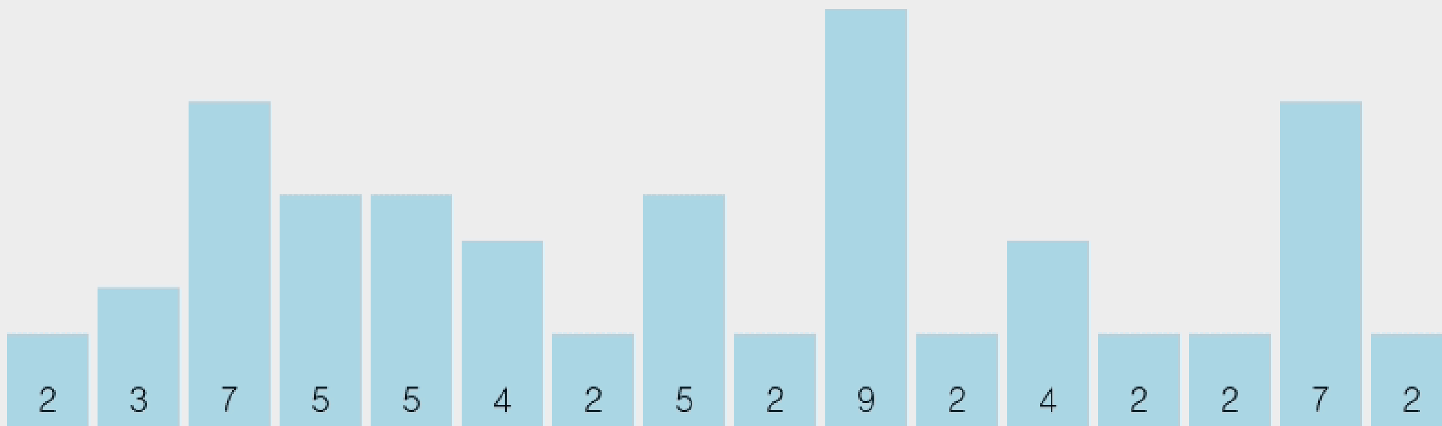
- É um algoritmo de ordenação que não necessita da operação de comparação entre os elementos.
- Por isso, é considerado como algoritmo de ordenação inteira (*integer sorting algorithm*).
- Pode ser usado para ordenar uma lista de elementos cujos valores estão num intervalo pré-definido `[0,k)`.
- É um algoritmo de ordenação estável.

Counting Sort

- Exemplo

Counting Sort

- Exemplo



Counting Sort

Counting Sort

- Algoritmo

Counting Sort

- Algoritmo
  - Ordenar uma lista `A` de tamanho `N`.

Counting Sort

- Algoritmo
  - Ordenar uma lista `A` de tamanho `N`.
  - Passos:

Counting Sort

- Algoritmo
  - Ordenar uma lista `A` de tamanho `N`.
  - Passos:
    - Inicializa-se uma lista `Aux[]` com 0s, onde o tamanho dessa lista deve ser `>= max (A[])`.

Counting Sort

- Algoritmo
  - Ordenar uma lista `A` de tamanho `N`.
  - Passos:
    - Inicializa-se uma lista `Aux[]` com 0s, onde o tamanho dessa lista deve ser `>= max (A[])`.
    - Percorra `A` e armazene a quantidade de ocorrências de cada elemento de `A` em uma posição apropriada de `Aux`.

Counting Sort

- Algoritmo
  - Ordenar uma lista `A` de tamanho `N`.
  - Passos:
    - Inicializa-se uma lista `Aux[]` com 0s, onde o tamanho dessa lista deve ser `>= max (A[])`.
    - Percorra `A` e armazene a quantidade de ocorrências de cada elemento de `A` em uma posição apropriada de `Aux`.
    - Inicializa-se `sortedA[]` vazio.

Counting Sort

- Algoritmo
  - Ordenar uma lista `A` de tamanho `N`.
  - Passos:
    - Inicializa-se uma lista `Aux[]` com 0s, onde o tamanho dessa lista deve ser `>= max (A[])`.
    - Percorra `A` e armazene a quantidade de ocorrências de cada elemento de `A` em uma posição apropriada de `Aux`.
    - Inicializa-se `sortedA[]` vazio.
    - Percorra a lista `Aux[ ]` e copie `i` em `sortedA` para `Aux[i]` número de vezes, onde `0 <= i <= max(A[])`.

Counting Sort

- Algoritmo
  - Ordenar uma lista `A` de tamanho `N`.
  - Passos:
    - Inicializa-se uma lista `Aux[]` com 0s, onde o tamanho dessa lista deve ser `>= max (A[])`.
    - Percorra `A` e armazene a quantidade de ocorrências de cada elemento de `A` em uma posição apropriada de `Aux`.
    - Inicializa-se `sortedA[]` vazio.
    - Percorra a lista `Aux[ ]` e copie `i` em `sortedA` para `Aux[i]` número de vezes, onde `0 <= i <= max(A[])`.
  - A lista `A[ ]` somente pode ser ordenada por esse algoritmo se o valor máximo de `A` é menor do que o tamanho máximo de `Aux[]`.

Counting Sort

Counting Sort

- Complexidade

Counting Sort

- Complexidade
  - A lista `A[]` é percorrida no tempo de O(N).

Counting Sort

- Complexidade
  - A lista `A[]` é percorrida no tempo de O(N).
  - A lista `Aux[]` é percorrida no tempo de O(k).

Counting Sort

- Complexidade
  - A lista `A[]` é percorrida no tempo de O(N).
  - A lista `Aux[]` é percorrida no tempo de O(k).
  - Portanto, o tempo médio é O(N + k).

Bucket Sort

Bucket Sort

- Também conhecido como *bin sort*.

Bucket Sort

- Também conhecido como *bin sort*.
- Supõe que os elementos da entrada estão distribuídos.

Bucket Sort

- Também conhecido como *bin sort*.
- Supõe que os elementos da entrada estão distribuídos.
- A ideia é dividir os elementos em segmentos de mesmo tamanho (*buckets*).

Bucket Sort

- Também conhecido como *bin sort*.
- Supõe que os elementos da entrada estão distribuídos.
- A ideia é dividir os elementos em segmentos de mesmo tamanho (*buckets*).
- Espera-se que o número de elementos seja o mesmo em todos os *buckets* (segmentos).

Bucket Sort

- Também conhecido como *bin sort*.
- Supõe que os elementos da entrada estão distribuídos.
- A ideia é dividir os elementos em segmentos de mesmo tamanho (*buckets*).
- Espera-se que o número de elementos seja o mesmo em todos os *buckets* (segmentos).
- Em seguida, os elementos são ordenados por um método qualquer.

Bucket Sort

- Também conhecido como *bin sort*.
- Supõe que os elementos da entrada estão distribuídos.
- A ideia é dividir os elementos em segmentos de mesmo tamanho (*buckets*).
- Espera-se que o número de elementos seja o mesmo em todos os *buckets* (segmentos).
- Em seguida, os elementos são ordenados por um método qualquer.
- Depois os elementos são concatenados.

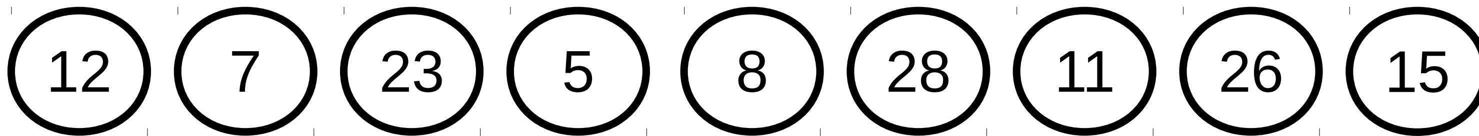
Bucket Sort

Bucket Sort

- Exemplo:

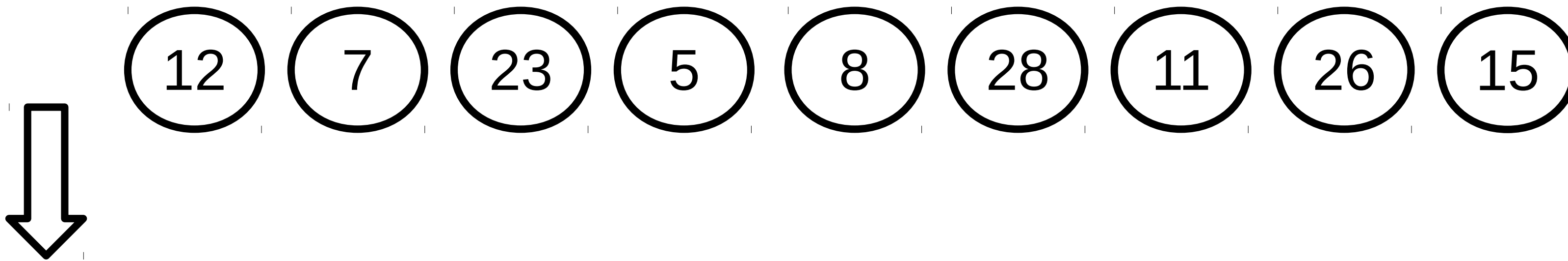
Bucket Sort

- Exemplo:



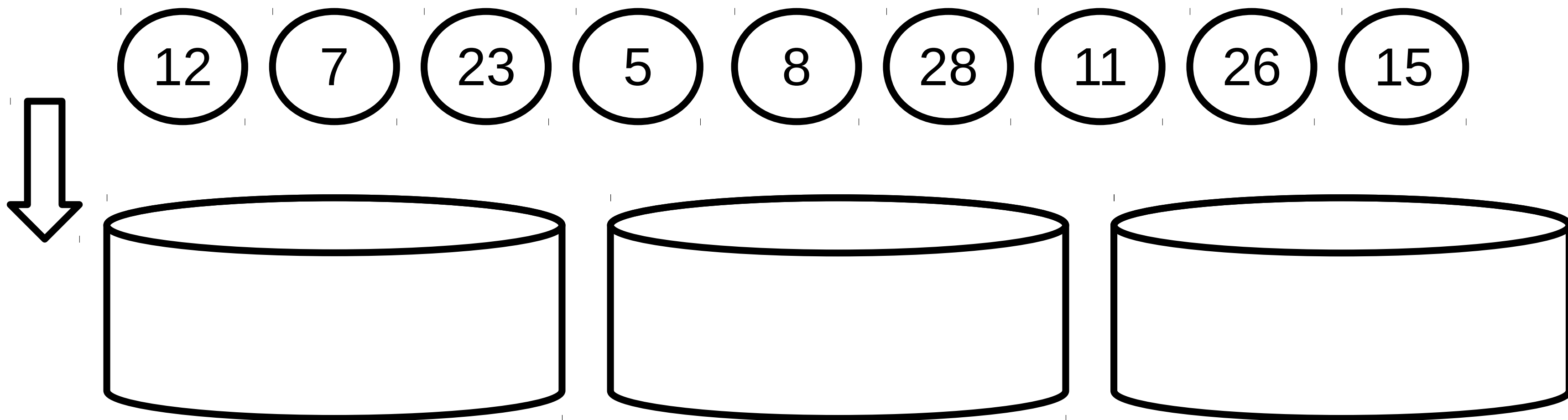
Bucket Sort

- Exemplo:



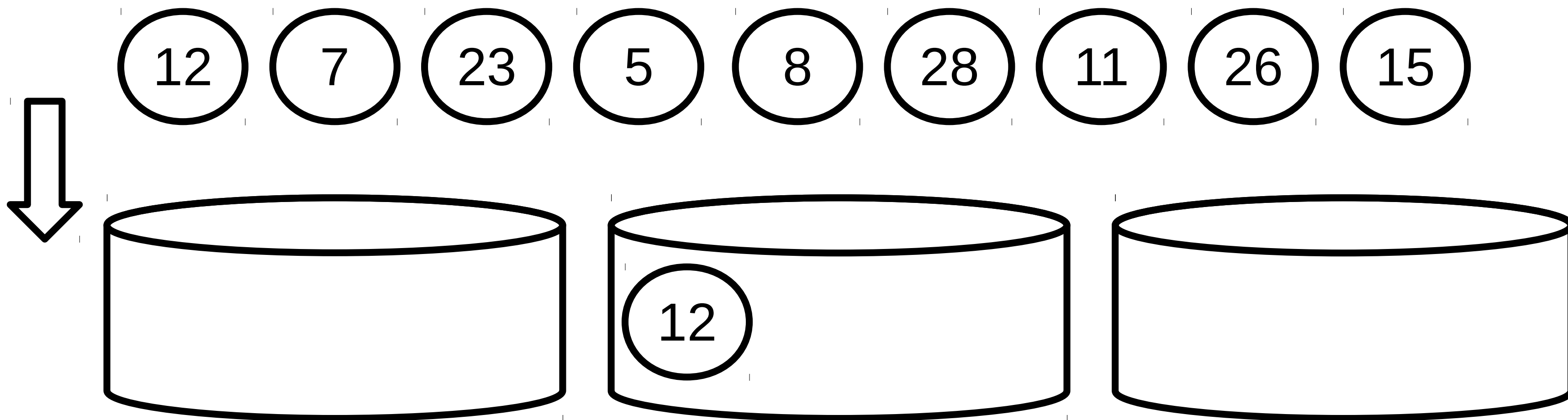
Bucket Sort

- Exemplo:



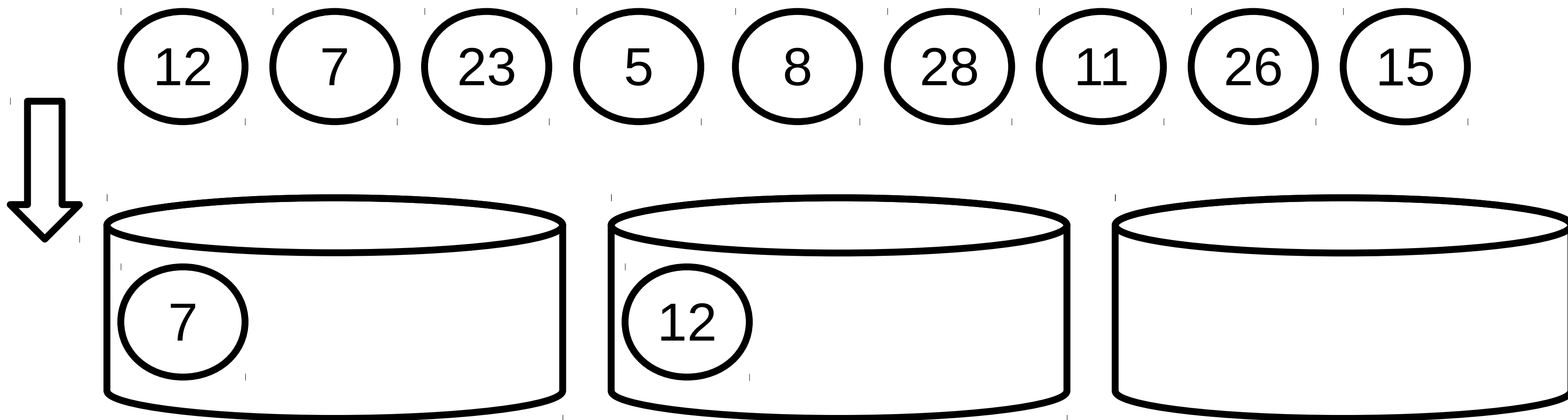
Bucket Sort

- Exemplo:



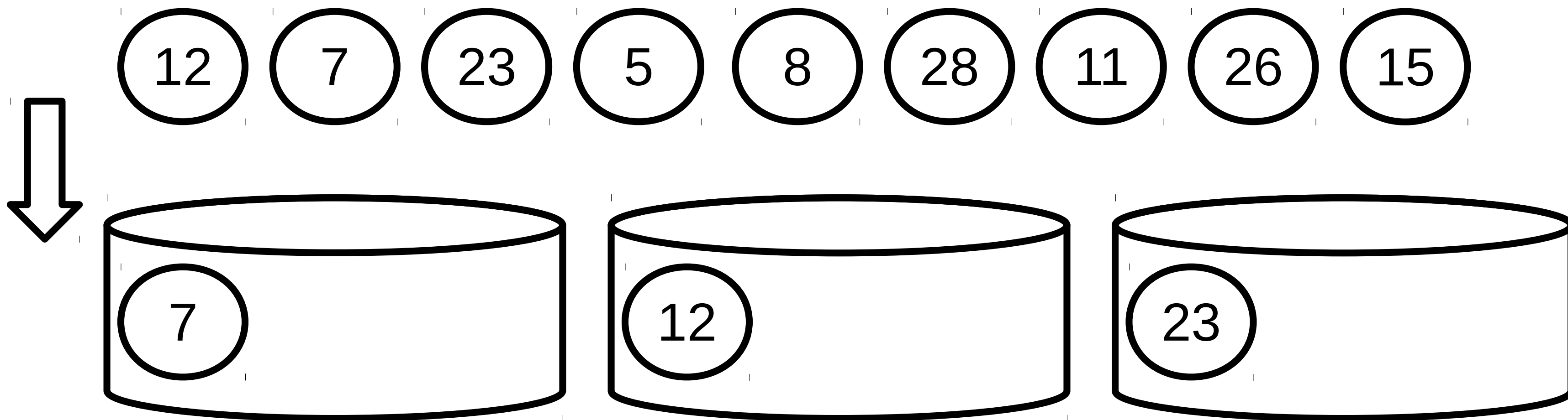
Bucket Sort

- Exemplo:



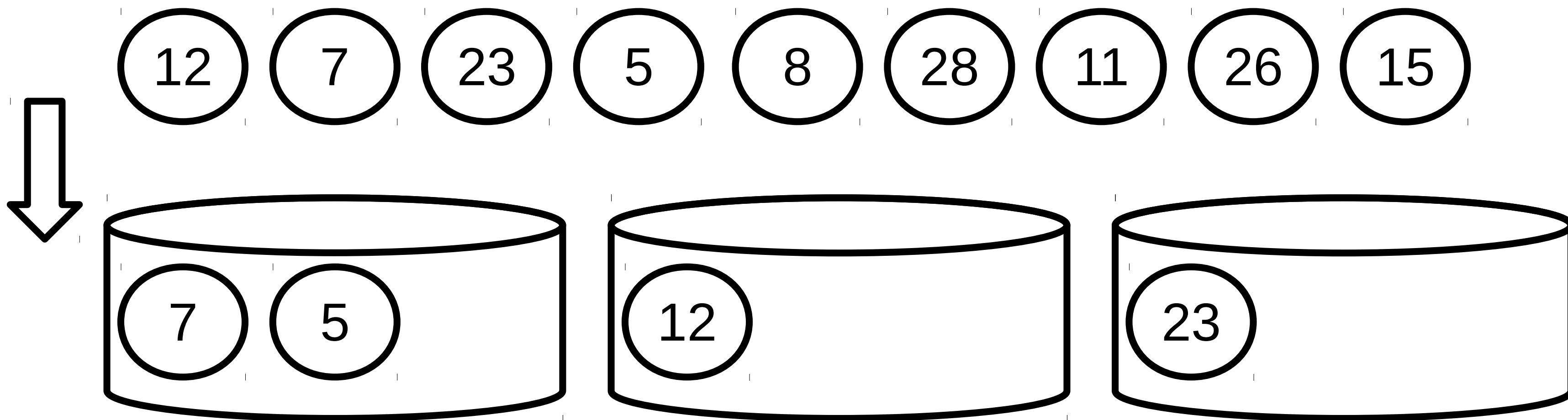
Bucket Sort

- Exemplo:



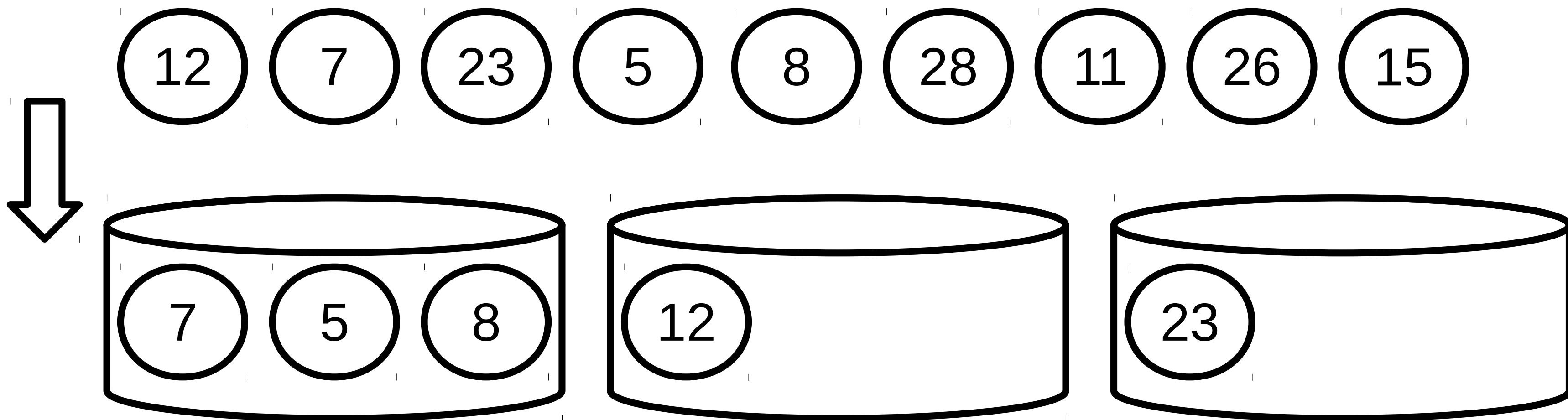
Bucket Sort

- Exemplo:



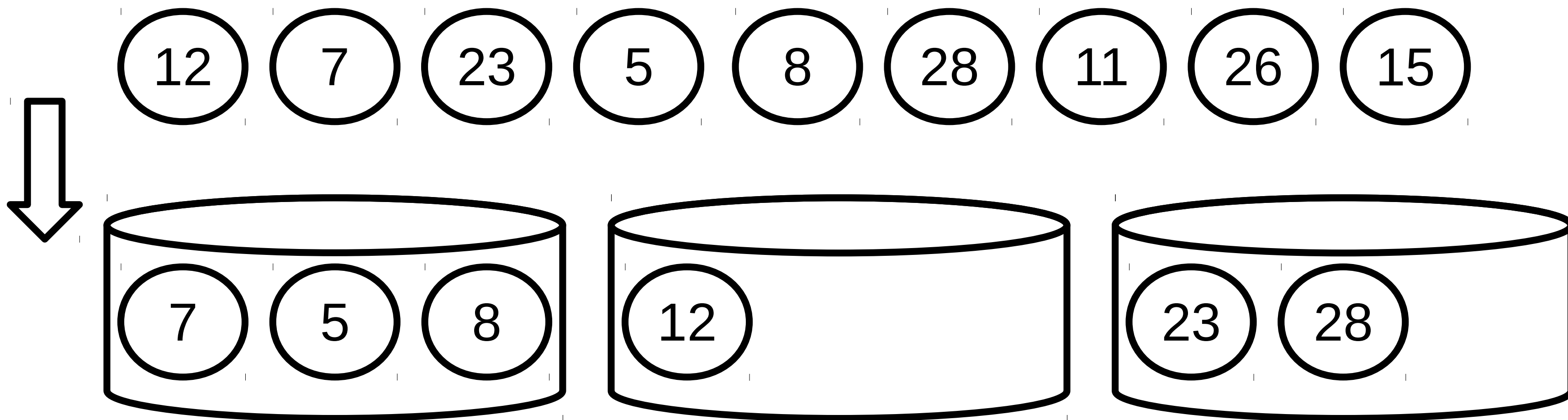
Bucket Sort

- Exemplo:



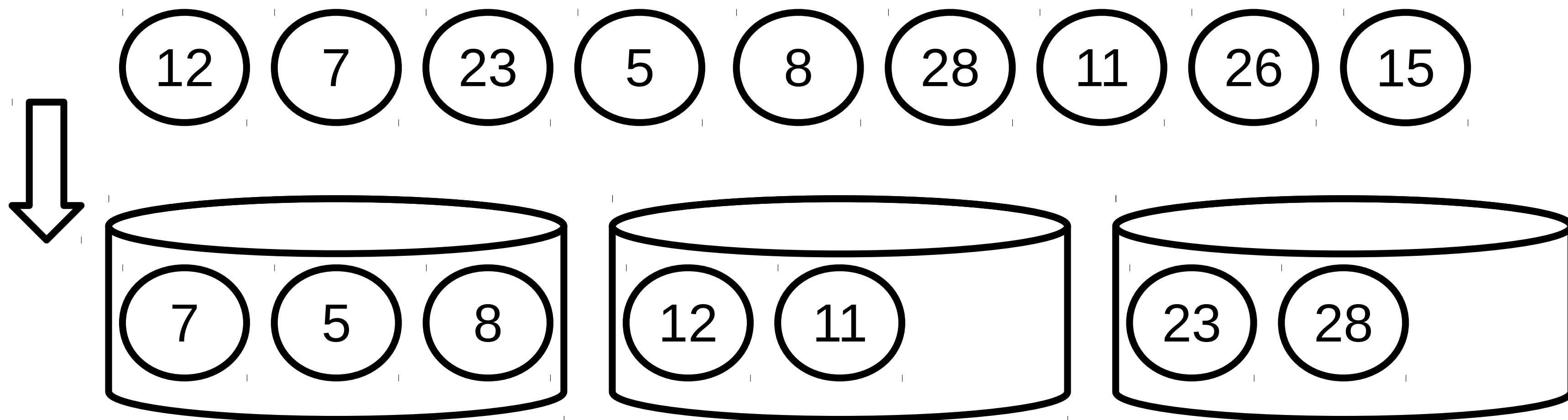
Bucket Sort

- Exemplo:



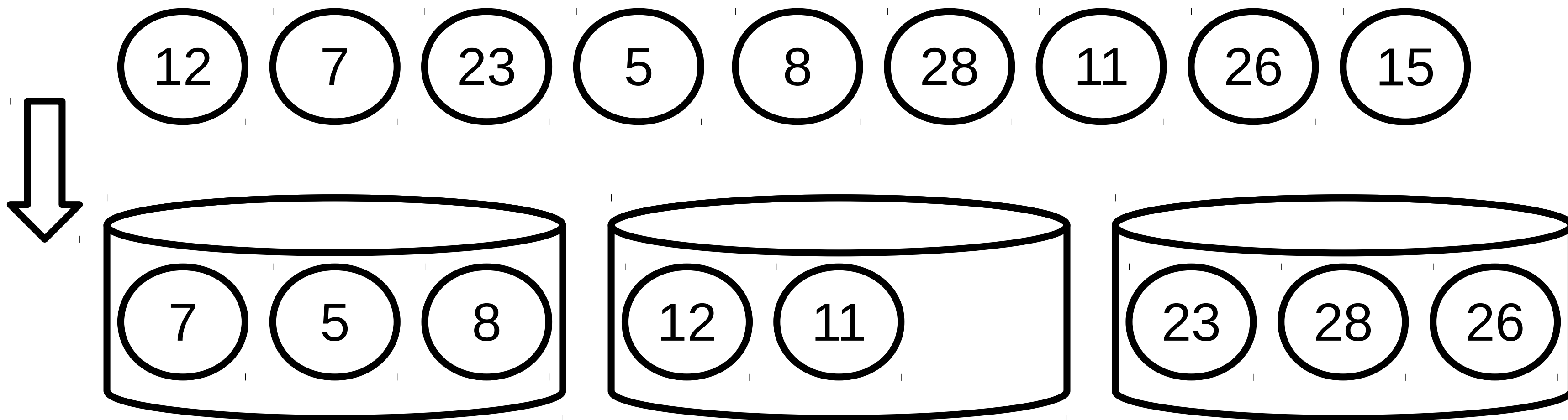
Bucket Sort

- Exemplo:



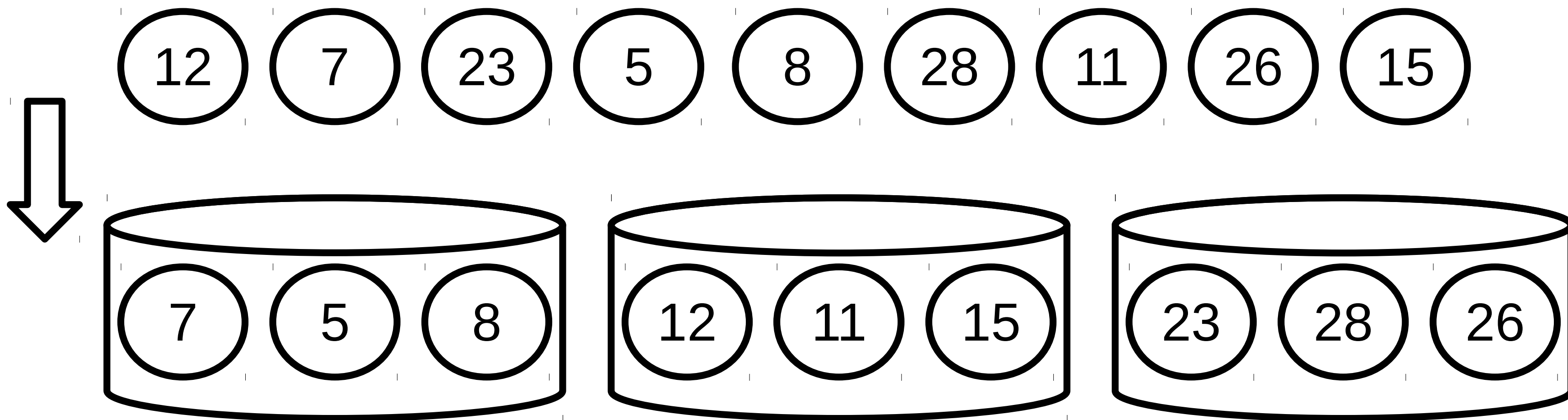
Bucket Sort

- Exemplo:



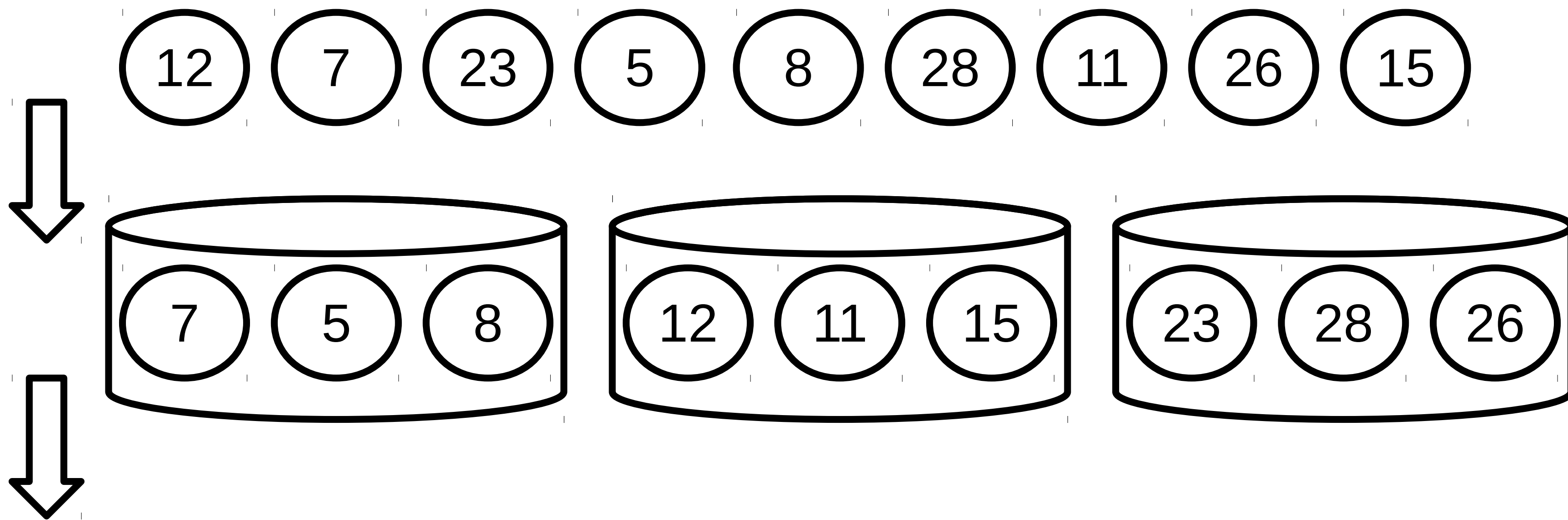
Bucket Sort

- Exemplo:



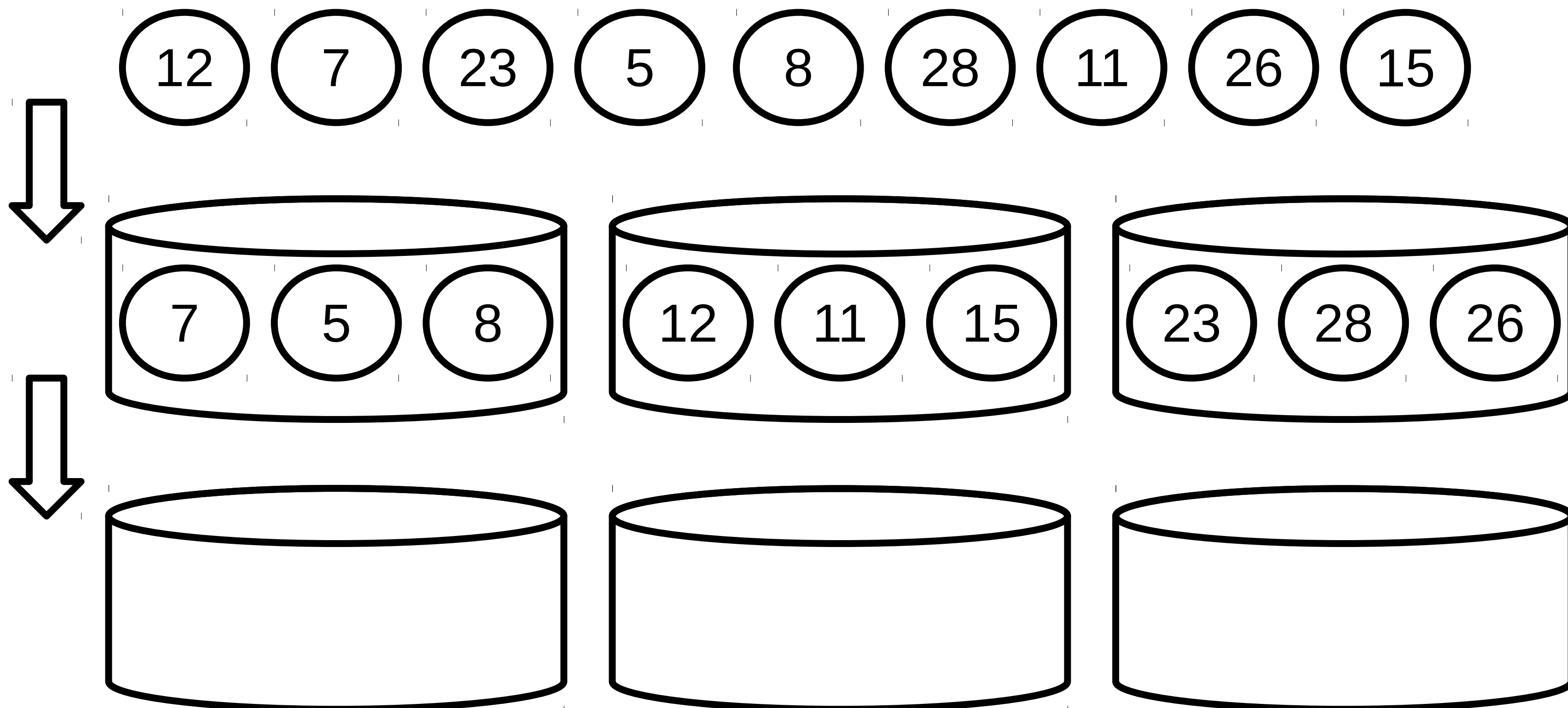
Bucket Sort

- Exemplo:



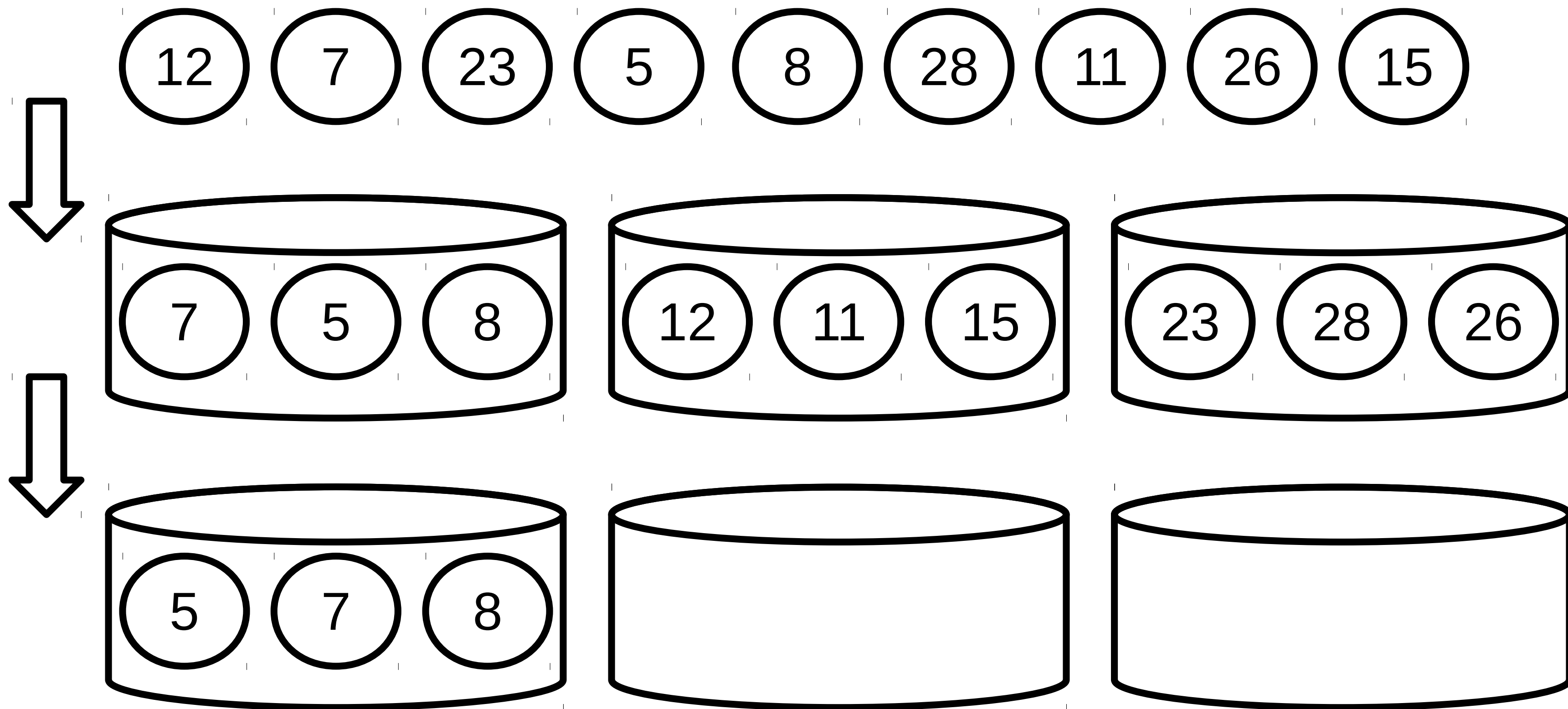
Bucket Sort

- Exemplo:



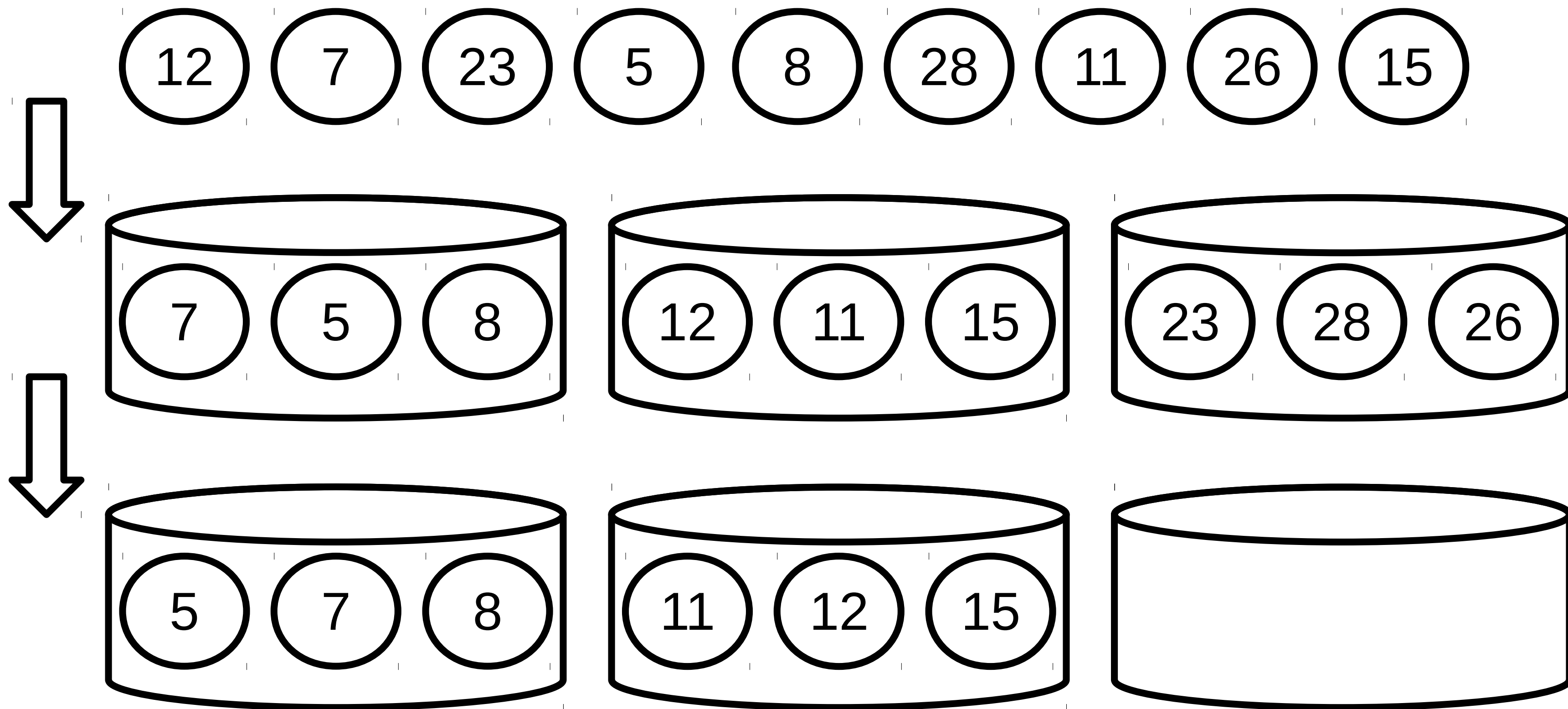
Bucket Sort

- Exemplo:



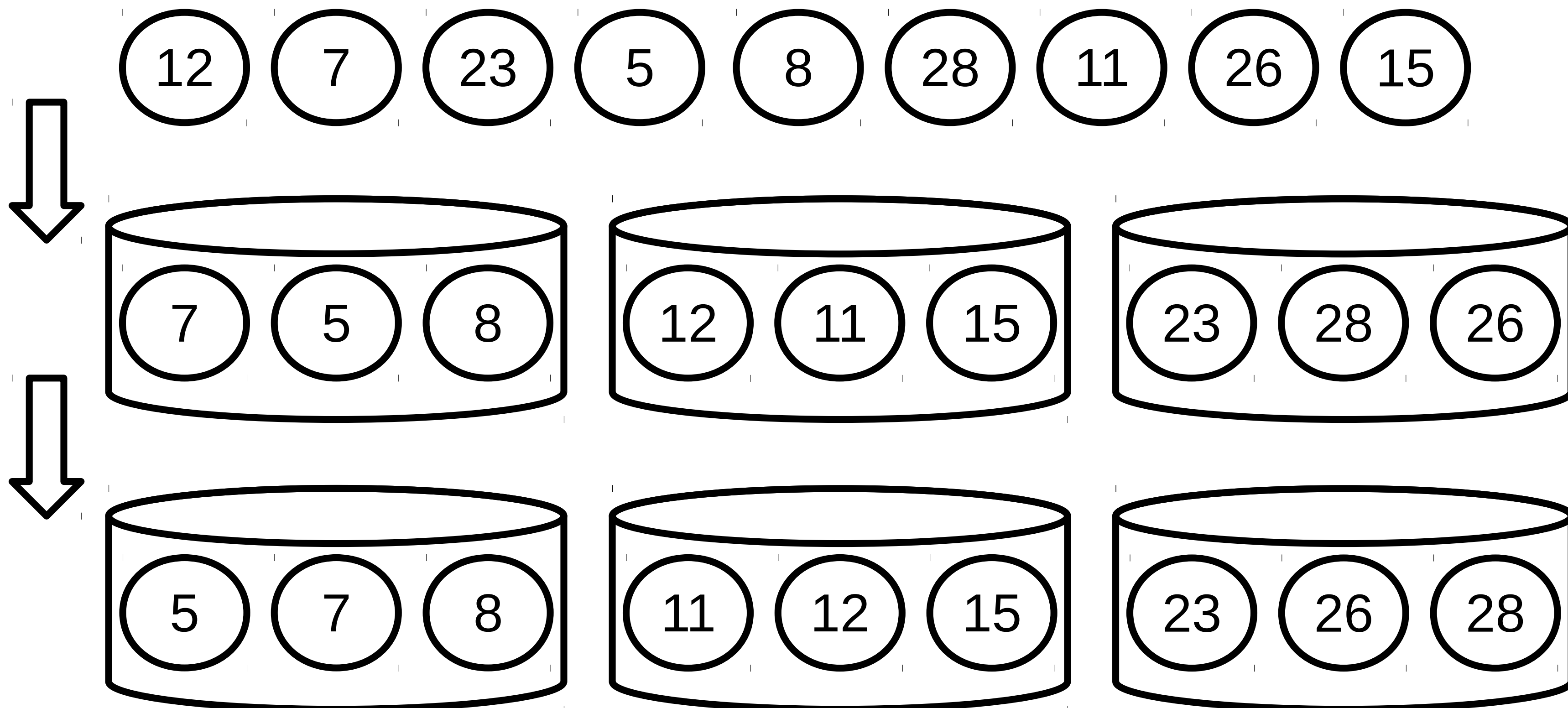
Bucket Sort

- Exemplo:



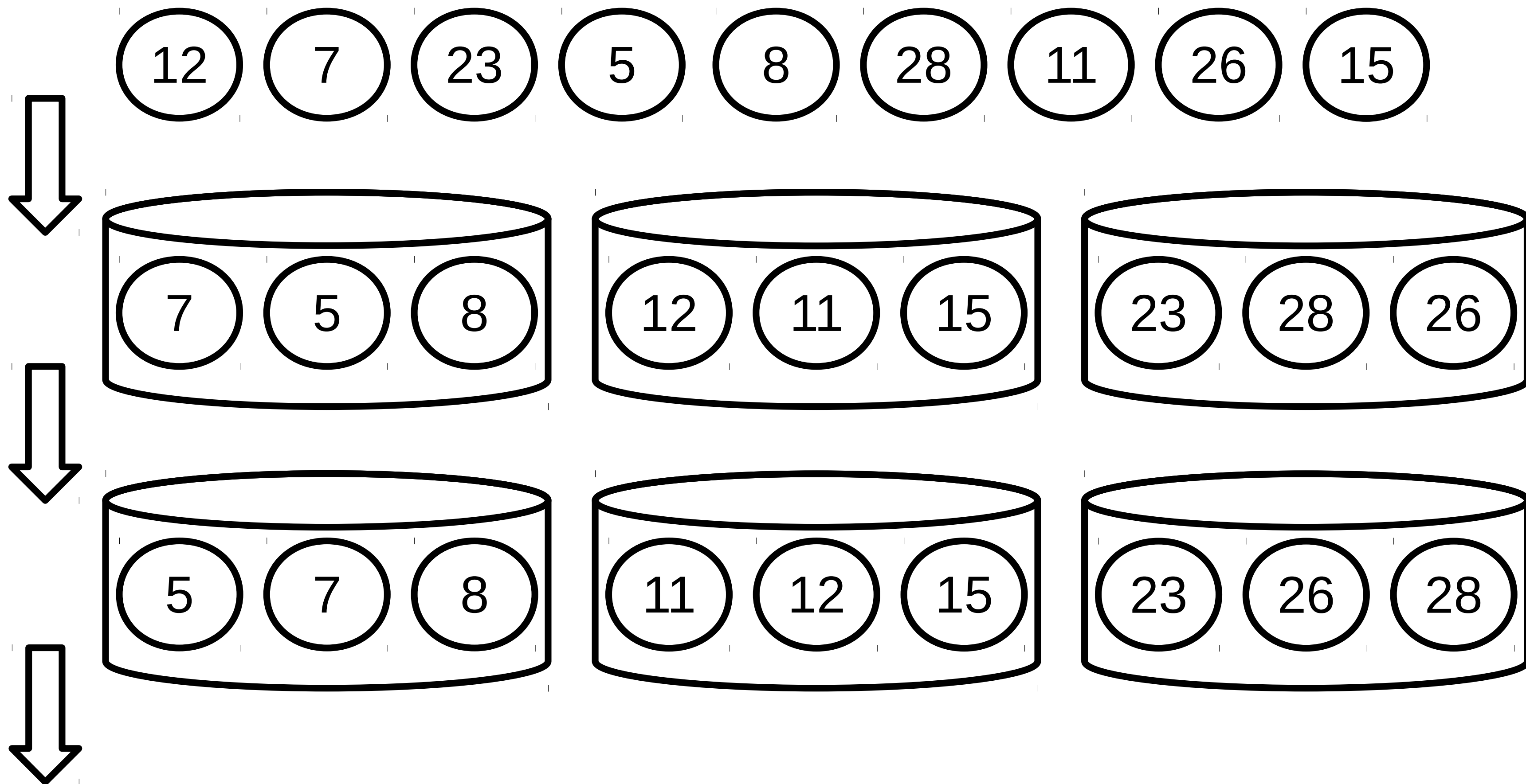
Bucket Sort

- Exemplo:



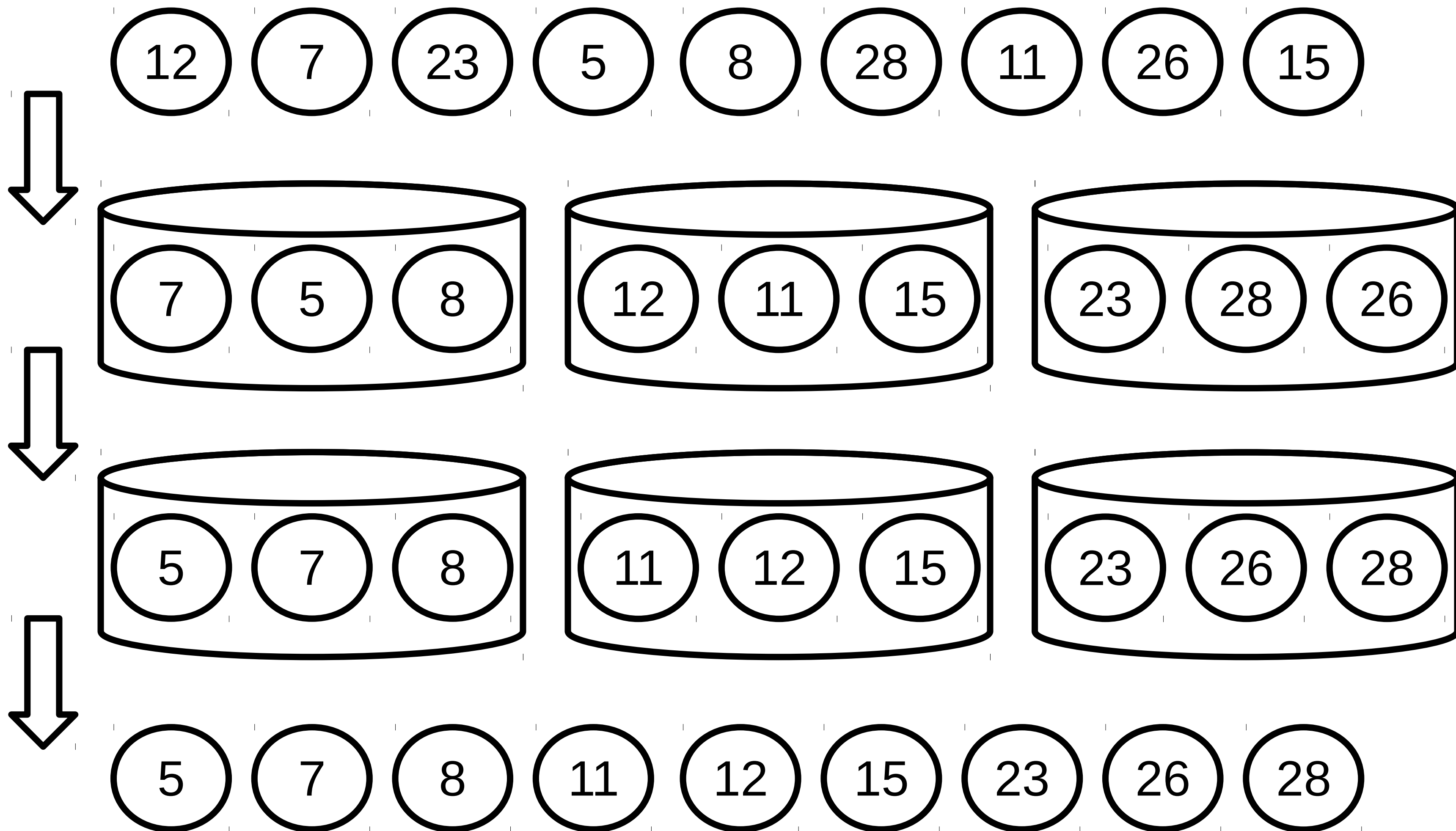
Bucket Sort

- Exemplo:



Bucket Sort

- Exemplo:



Bucket Sort

Bucket Sort

- Algoritmo

Bucket Sort

- Algoritmo
  - Inicialize um *array* de “baldes” inicialmente vazios.

Bucket Sort

- Algoritmo
  - Inicialize um *array* de “baldes” inicialmente vazios.
  - Distribua os elementos colocando-os cada um dos “baldes”.

Bucket Sort

- Algoritmo
  - Inicialize um *array* de “baldes” inicialmente vazios.
  - Distribua os elementos colocando-os cada um dos “baldes”.
  - Ordene cada balde não-vazio.

Bucket Sort

- Algoritmo
  - Inicialize um *array* de “baldes” inicialmente vazios.
  - Distribua os elementos colocando-os cada um dos “baldes”.
  - Ordene cada balde não-vazio.
  - Coloque os elementos dos baldes que não estão vazios no *array* original.

Bucket Sort

Bucket Sort

- Análise

Bucket Sort

- Análise
  - O algoritmo assume que a entrada de dados tem uma distribuição uniforme.

Bucket Sort

- Análise
  - O algoritmo assume que a entrada de dados tem uma distribuição uniforme.
  - A complexidade computacional tem relação com o número de “baldes”.

Bucket Sort

- Análise
  - O algoritmo assume que a entrada de dados tem uma distribuição uniforme.
  - A complexidade computacional tem relação com o número de “baldes”.
  - O método pode ser eficiente dependendo da distribuição dos elementos e da quantidade de *buckets*.

Bucket Sort

- Análise
  - O algoritmo assume que a entrada de dados tem uma distribuição uniforme.
  - A complexidade computacional tem relação com o número de “baldes”.
  - O método pode ser eficiente dependendo da distribuição dos elementos e da quantidade de *buckets*.
  - O algoritmo é $O(n+k)$ para o caso médio e $O(n^2)$ para o pior caso.

Resumo Comparativo

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

https://medium.com/@george.seif94/a-tour-of-the-top-5-sorting-algorithms-with-python-code-43ea9aa02889

Resumo Comparativo

Name	Average Case	Worst Case	Auxiliary Memory	Is Stable?	Other Notes
Bubble	$O(n^2)$	$O(n^2)$	1	yes	Small code
Selection	$O(n^2)$	$O(n^2)$	1	no	Stability depends on the implementation.
Insertion	$O(n^2)$	$O(n^2)$	1	yes	Average case is also $O(n + d)$, where d is the number of inversions.
Shell	-	$O(nlog^2n)$	1	no	
Merge sort	$O(nlogn)$	$O(nlogn)$	depends	yes	
Heap sort	$O(nlogn)$	$O(nlogn)$	1	no	
Quick sort	$O(nlogn)$	$O(n^2)$	$O(logn)$	depends	Can be implemented as a stable sort depending on how the pivot is handled.
Tree sort	$O(nlogn)$	$O(n^2)$	$O(n)$	depends	Can be implemented as a stable sort.

Exercícios

- Considere o código abaixo e responda ao que se pede:
  - A função MetodoX implementa qual algoritmo de ordenação?
  - Qual é a finalidade da Linha 6 do programa?

```
 1 def MetodoX(array, maxval):
 2     n = len(array)
 3     m = maxval + 1
 4     count = [0] * m
 5     for a in array:
 6         count[a] += 1
 7     i = 0
 8     for a in range(m):
 9         for c in range(count[a]):
10             array[i] = a
11             i += 1
12     return array
13
14 print(counting_sort( [1, 4, 7, 2, 1, 3, 2, 1, 4, 2, 3, 2, 1], 7 ))
```

Exercícios

- Considere os seguintes métodos de ordenação: bolha, seleção e inserção. Qual deles executa o menor número de comparações para uma lista de entrada contendo valores idênticos?
- Mostre um exemplo que demonstre que o ShellSort é instável para a sequência de h=1,2.
- Um amigo lhe diz que é capaz de ordenar qualquer sequência de 6 números com no máximo 8 comparações. O seu amigo está falando a verdade? Justifique a sua resposta.

Exercícios

- Quantas trocas são realizadas quando executamos o algoritmo de Seleção para ordenar uma lista com N elementos?
- Considere a seguinte afirmação: “O algoritmo *counting sort* é eficiente se a variação da entrada de dados não é significativamente maior do que o número de objetos que serão ordenados”. A afirmação é verdadeira? Se for, justifique a afirmação e dê um exemplo que confirme a justificativa. Se não for verdadeira, justifique o motivo e dê um exemplo que corrobore com a sua justificativa.

Exercícios

PROBLEM *What's the best algorithm to use for sorting?*

Exercícios

PROBLEM *You need to sort a variety of different kinds of data about which little is known in advance. Data sets will be small enough to fit in memory, but their size may vary widely. What sorting algorithm would you choose?*

Exercícios

PROBLEM *A system that monitors a manufacturing plant maintains a list of serial numbers of every item that has ever failed quality control. During the day, while the plant is operating, new serial numbers are added to the end of the list. Each night, a batch process runs to re-sort the list. What's the best sorting algorithm for this?*

Referências

- Visualização de algoritmos de ordenação
  - https://www.toptal.com/developers/sorting-algorithms