


Ordenações Bidirecionais e Distribuídas



Fundamentação teórica e origem dos algoritmos

Contextualização Histórica

A história dos algoritmos de ordenação acompanha o avanço da computação. Desde os anos 1950, pesquisadores estudam formas mais rápidas e eficientes de ordenar dados. Muitos dos algoritmos conhecidos hoje foram criados para resolver problemas específicos com restrições de tempo e espaço.



Bucket Sort – Distribuição, Aplicação e Eficiência

O Bucket Sort é especialmente útil para dados reais ou inteiros que estejam distribuídos de forma relativamente uniforme.



Cocktail Sort – Origem, Funcionamento e Complexidade

O Cocktail Sort funciona como uma variação do Bubble Sort, mas com ida e volta a cada iteração. Isso permite que elementos pequenos 'subam' rapidamente para o início e elementos grandes 'desçam' para o final.

Pigeonhole Sort – Princípio Matemático e Eficiência

O Pigeonhole Sort é baseado no princípio de que, se temos mais itens que 'casas', ao menos uma casa terá mais de um item.

Complemento

- O Cocktail Sort é uma otimização didática de um método clássico.
- O Bucket Sort distribui os dados inteligentemente, sendo eficiente com dados bem distribuídos.
- O Pigeonhole Sort elimina comparações usando posicionamento direto — e é extremamente rápido em condições ideais.



Como
funciona

Cocktail Sort

CocktailSort(A):

trocado \leftarrow verdadeiro

início \leftarrow 0, fim \leftarrow tamanho de A - 1

enquanto trocado:

trocado \leftarrow falso

para i de início até fim-1:

se $A[i] > A[i+1]$, trocar e trocado \leftarrow verdadeiro

fim \leftarrow fim - 1

para i de fim até início+1:

se $A[i] < A[i-1]$, trocar e trocado \leftarrow verdadeiro

início \leftarrow início + 1



Cocktail Sort

[Video 1](#)

[Video 2](#)

Bucket Sort

BucketSort(A):

$n \leftarrow$ tamanho de A

 criar $B[0..n-1]$ // n baldes vazios

 para a em A:

$idx \leftarrow$ calcular índice do balde de a

 inserir a em $B[idx]$

 para b em B:

 ordenar b

$A \leftarrow$ concatenar todos os baldes $B[0..n-1]$

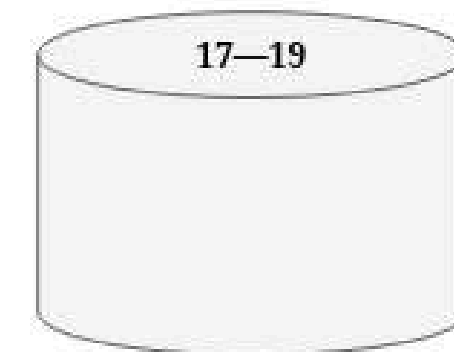
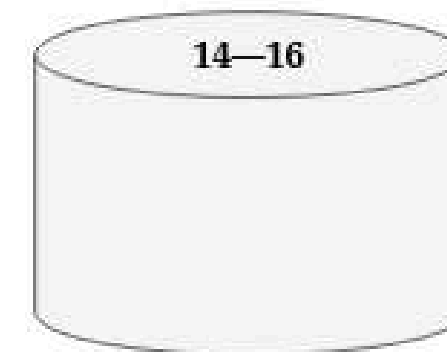
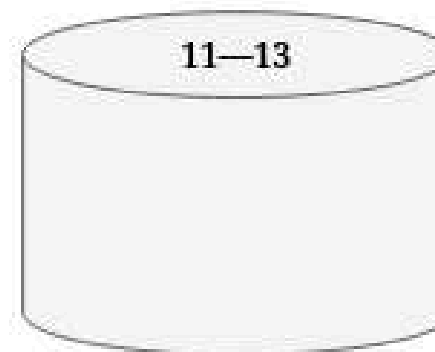
Bucket Sort

Bucket sort

array

18	13	15	11	19	12	14	13
----	----	----	----	----	----	----	----

buckets

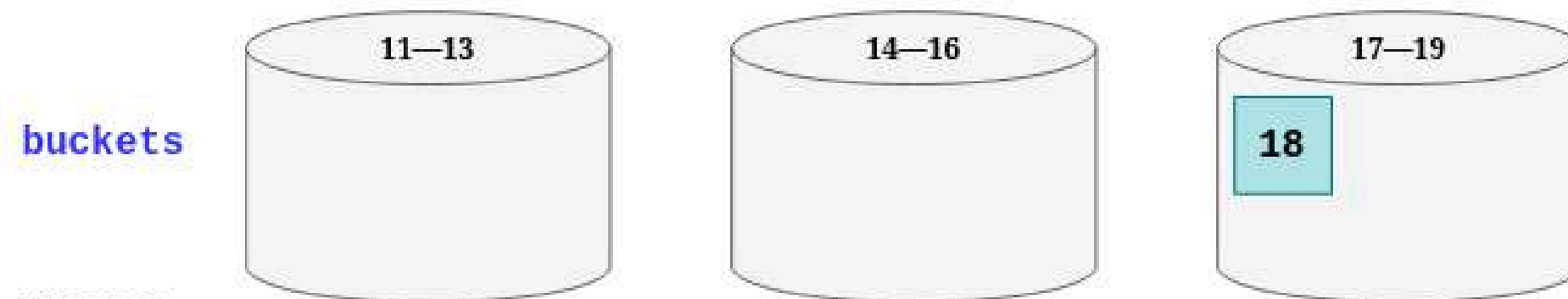
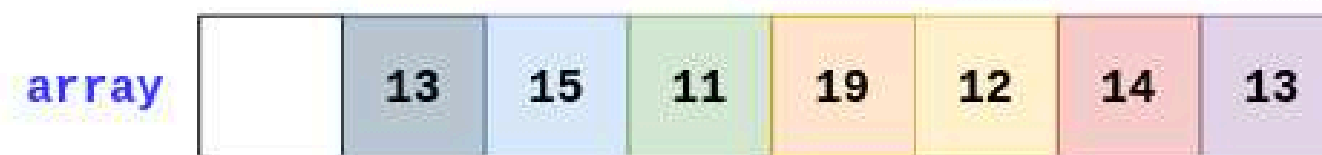


Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

Bucket Sort

Bucket sort



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

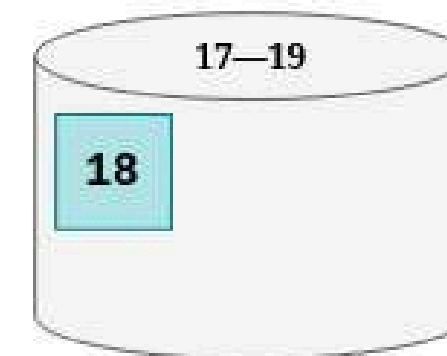
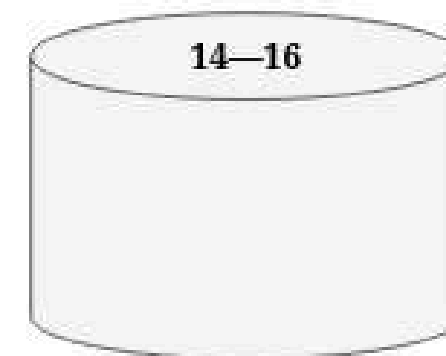
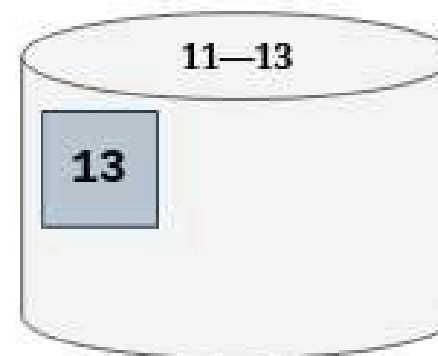
Bucket Sort

Bucket sort

array



buckets



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

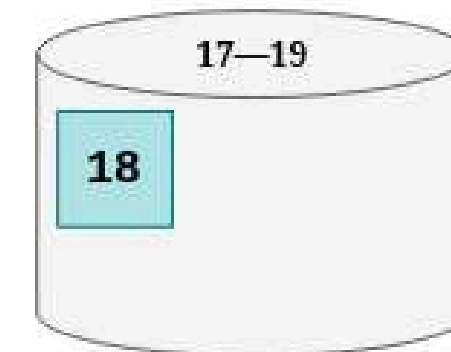
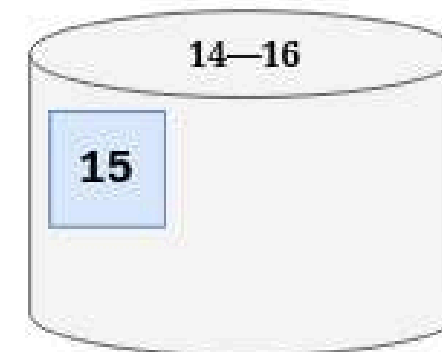
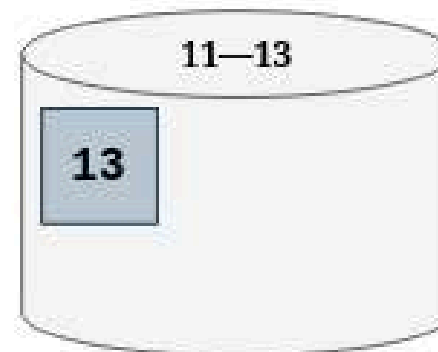
Bucket Sort

Bucket sort

array



buckets



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

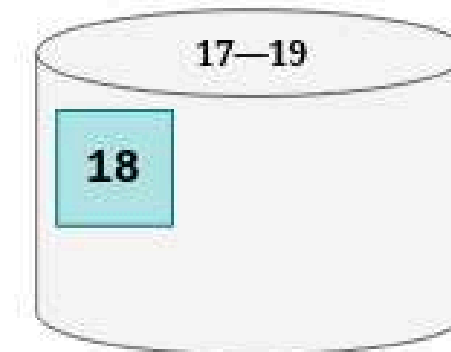
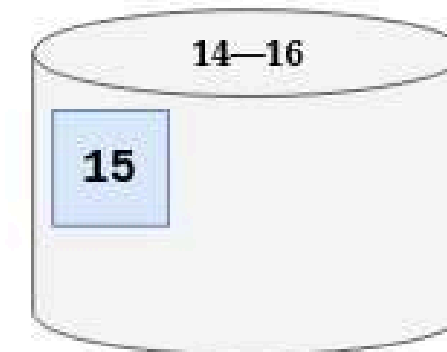
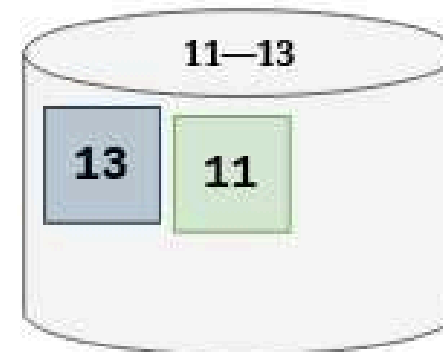
Bucket Sort

Bucket sort

array



buckets



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

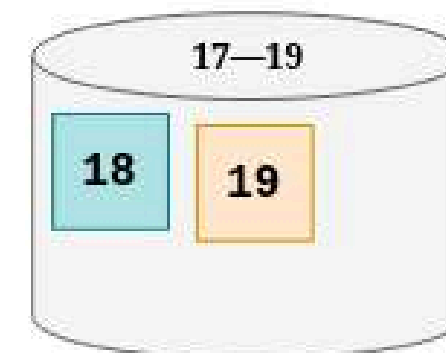
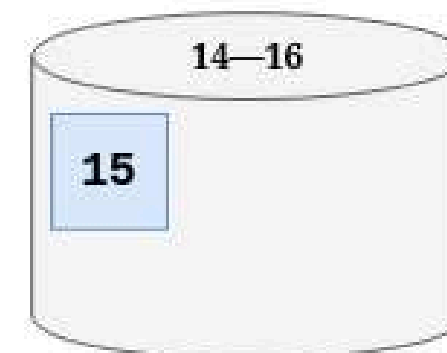
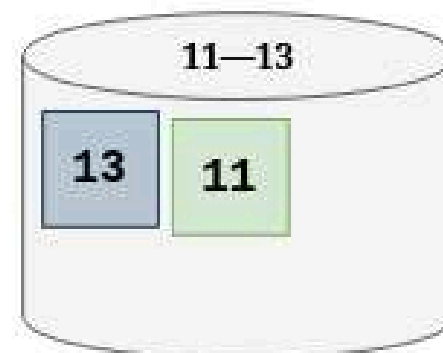
Bucket Sort

Bucket sort

array



buckets



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

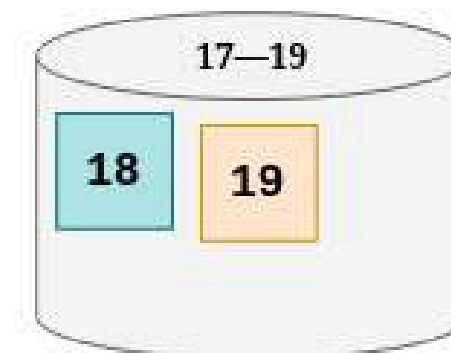
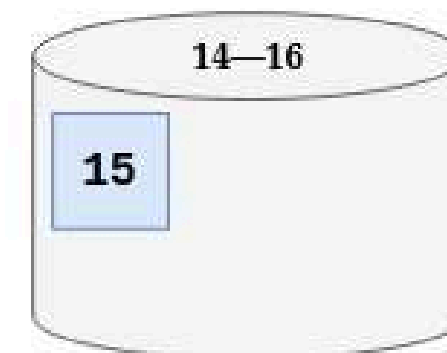
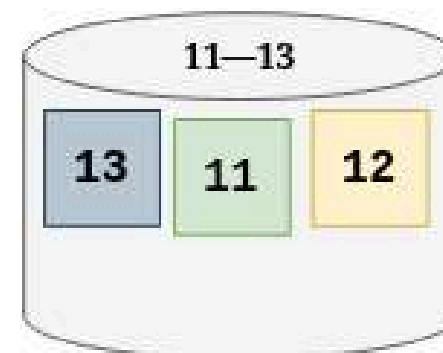
Bucket Sort

Bucket sort

array



buckets



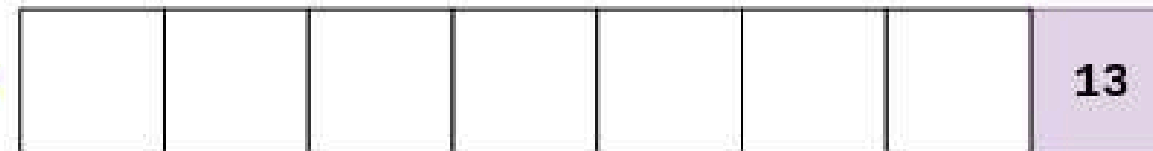
Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

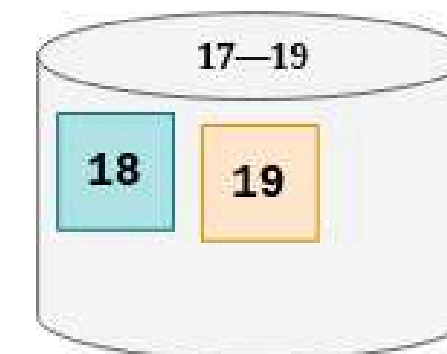
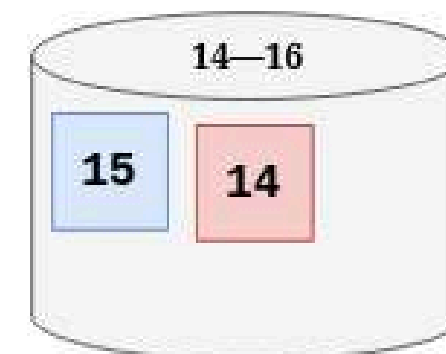
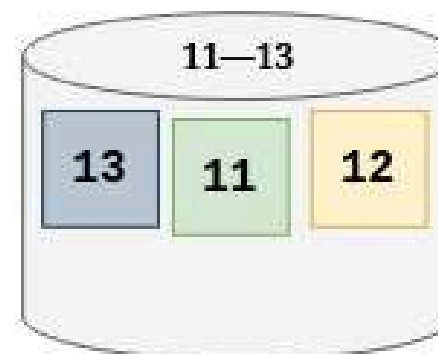
Bucket Sort

Bucket sort

array



buckets



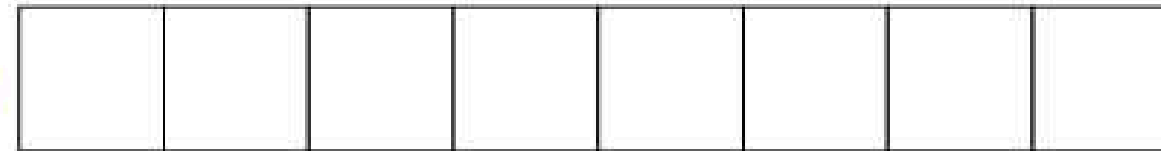
Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

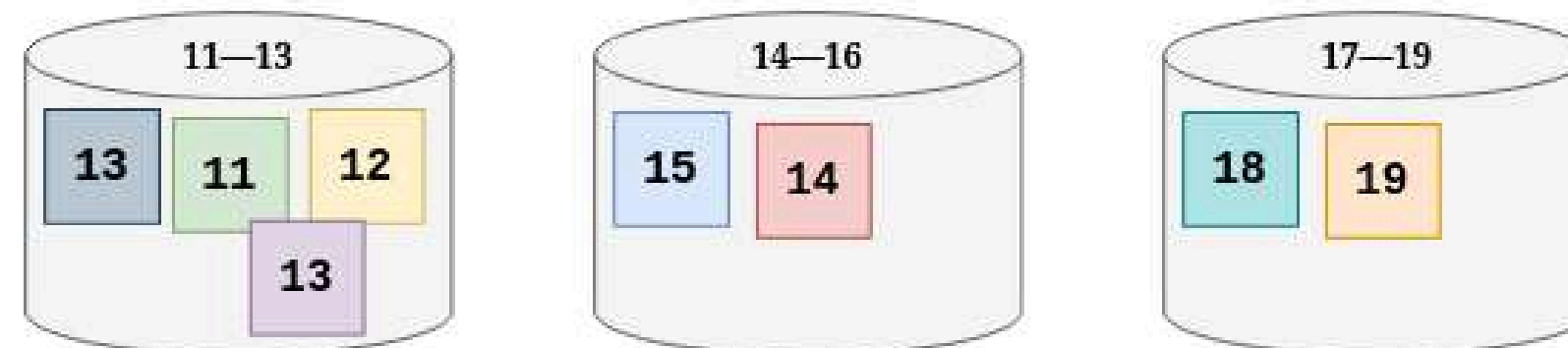
Bucket Sort

Bucket sort

array



buckets



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

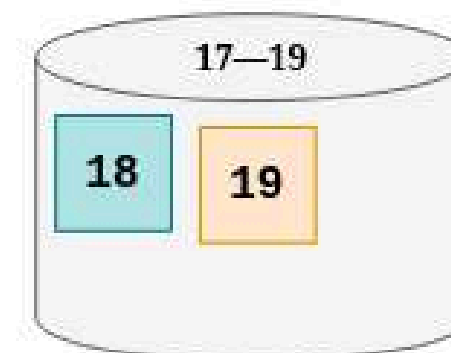
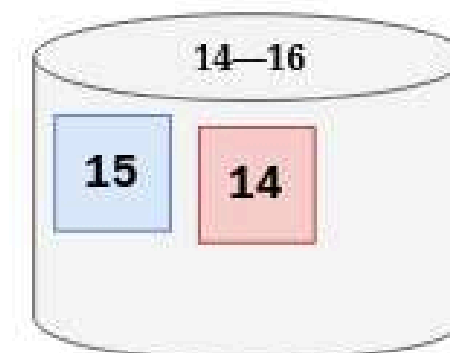
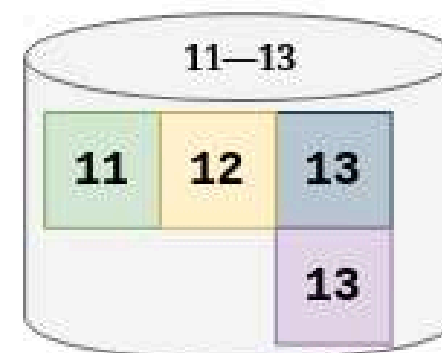
Bucket Sort

Bucket sort

array



buckets



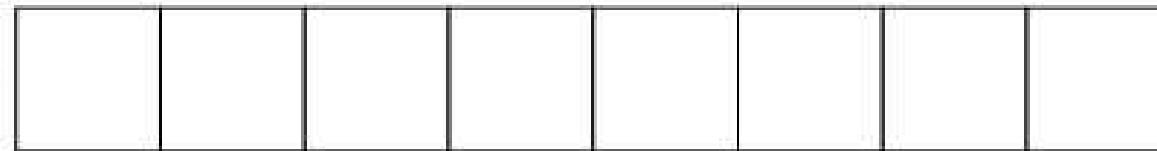
Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

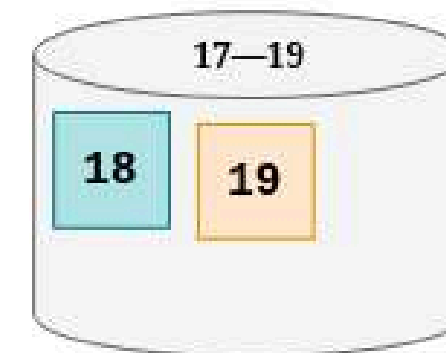
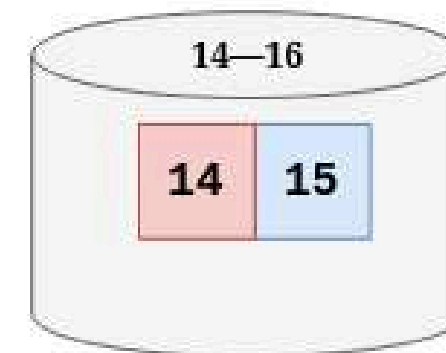
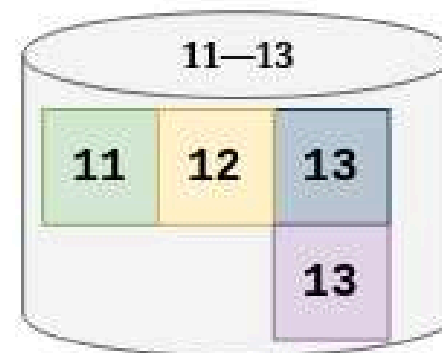
Bucket Sort

Bucket sort

array



buckets



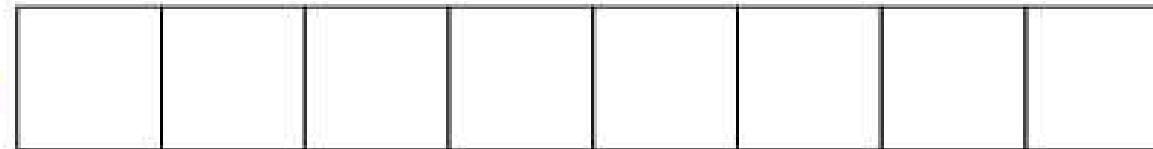
Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

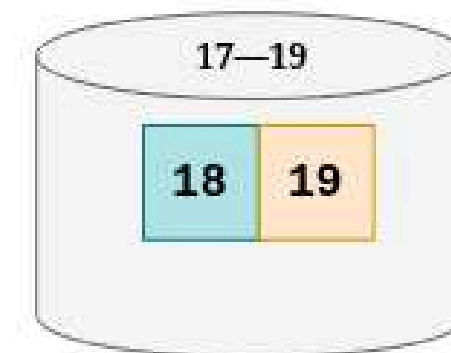
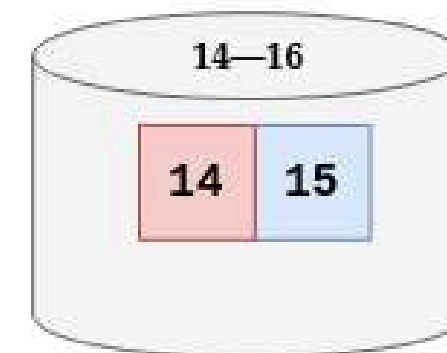
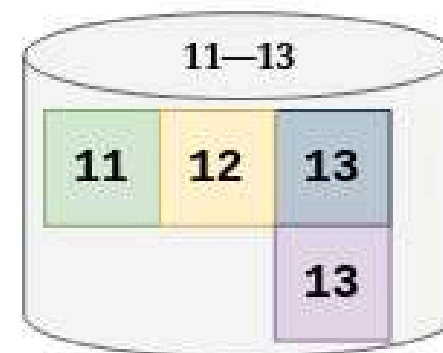
Bucket Sort

Bucket sort

array



buckets

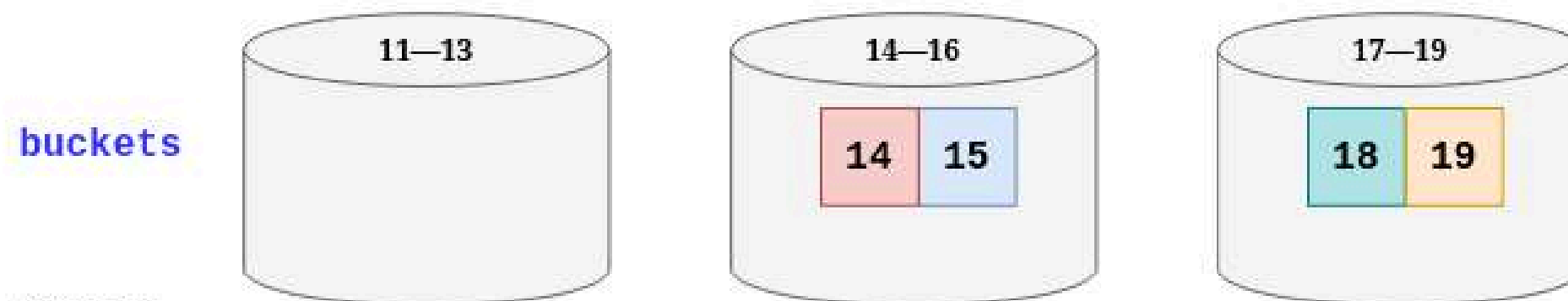


Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

Bucket Sort

Bucket sort



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

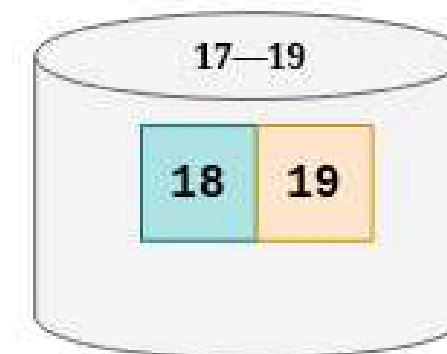
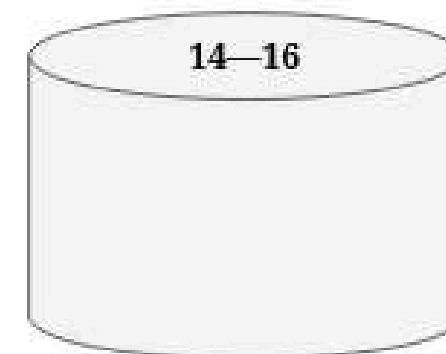
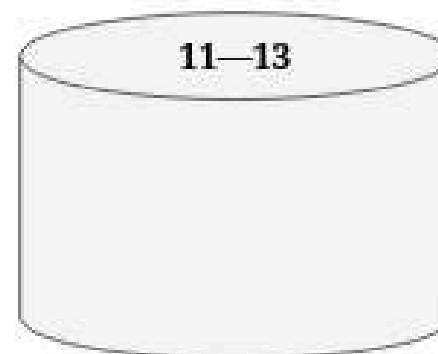
Bucket Sort

Bucket sort

array



buckets



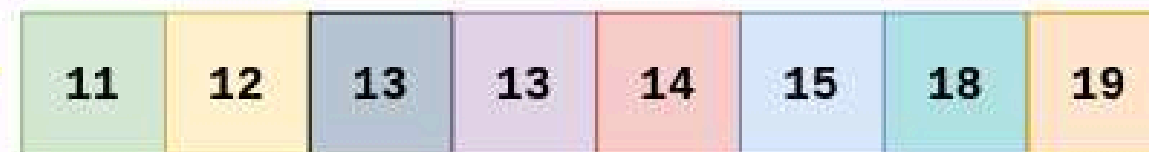
Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

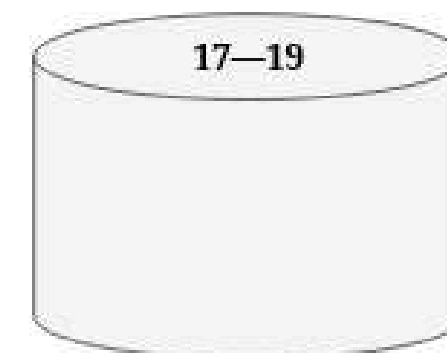
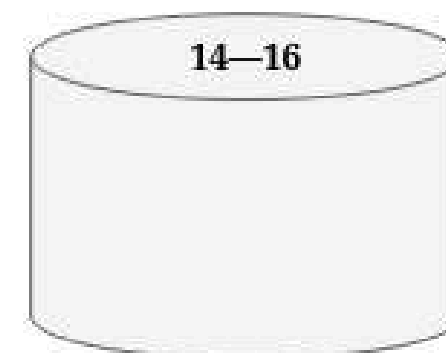
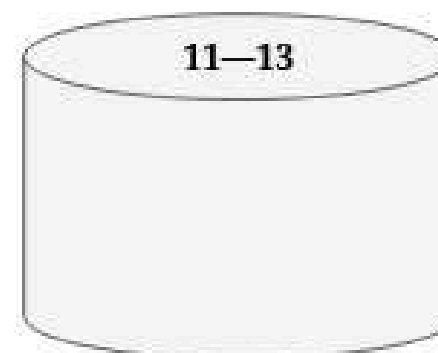
Bucket Sort

Bucket sort

array



buckets



Steps:

1. Create some empty buckets.
2. Place each element in its corresponding buckets.
3. Sort each bucket individually.
4. Merge all buckets.

Bucket Sort

Bucket sort

array

11	12	13	13	14	15	18	19
----	----	----	----	----	----	----	----

The array is sorted now!

Pigeonhole Sort

PigeonholeSort(A):

min \leftarrow menor valor em A

max \leftarrow maior valor em A

size \leftarrow max - min + 1

criar P[0..size-1] // buracos vazios

para a em A:

inserir a em P[a - min]

A \leftarrow concatenar todos os elementos de P em ordem

Pigeonhole Sort

Pigeonhole sort - Initial steps

array

18	13	15	11	19	12	14	13
----	----	----	----	----	----	----	----

`min_value = 11`

`max_value = 19`

`range_value = 9`

pigeon
holes

--	--	--	--	--	--	--	--	--

Steps:

1. `min_value = min(array)`
2. `max_value = max(array)`
3. `range = max_value - min_value + 1`
4. `pigeonholes = array[range]`

Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array

18	13	15	11	19	12	14	13
----	----	----	----	----	----	----	----

min_value = 11

max_value = 19

range_value = 9

i = 0

pigeon
holes

--	--	--	--	--	--	--	--	--

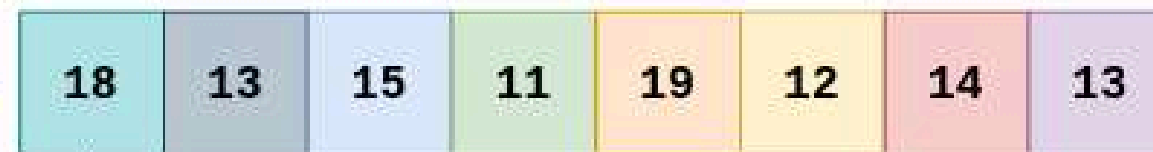
Steps:

```
5. for i from 0 to n:  
    push array[i] into pigeonholes[array[i] - min]
```

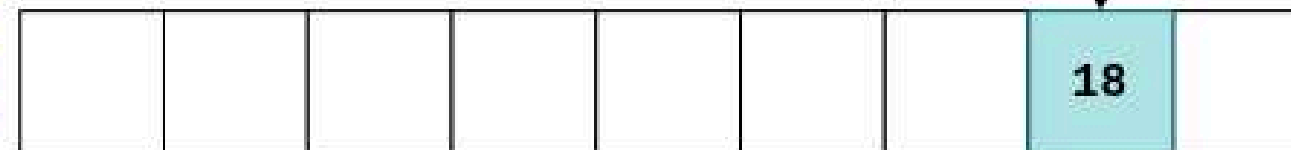
Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array



pigeon
holes



min_value = 11

max_value = 19

range_value = 9

i = 0

Steps:

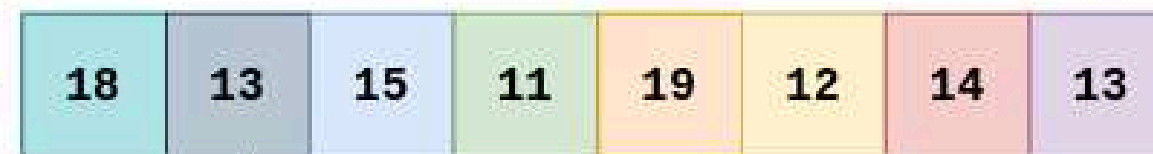
5. for i from 0 to n:

 push array[i] into pigeonholes[array[i] - min]

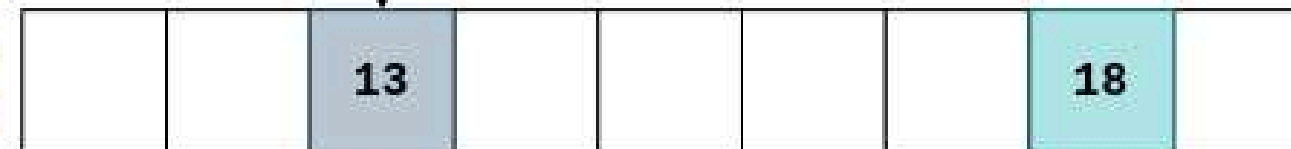
Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array



pigeon
holes



min_value = 11

max_value = 19

range_value = 9

i = 1

Steps:

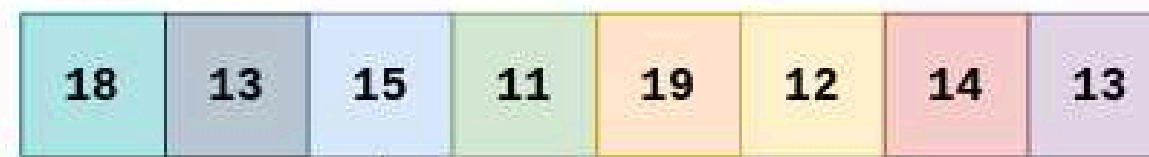
5. for i from 0 to n:

push array[i] into pigeonholes[array[i] - min]

Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array



min_value = 11

max_value = 19

range_value = 9

i = 2

pigeon
holes



Steps:

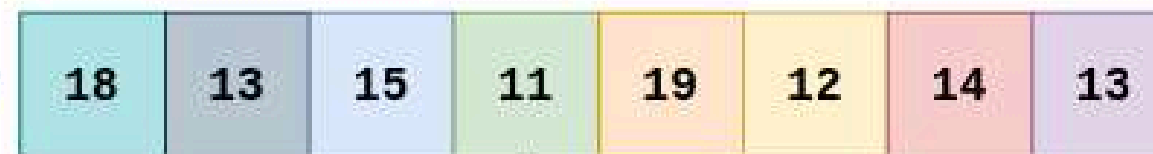
5. for i from 0 to n:

push array[i] into pigenholes[array[i] - min]

Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array



pigeon
holes



min_value = 11

max_value = 19

range_value = 9

i = 3

Steps:

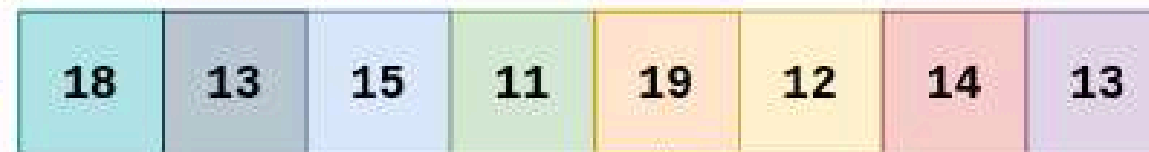
5. for i from 0 to n:

 push array[i] into pigeonholes[array[i] - min]

Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array



min_value = 11

max_value = 19

range_value = 9

i = 4

pigeon
holes



Steps:

5. for i from 0 to n:

 push array[i] into pigeonholes[array[i] - min]

Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

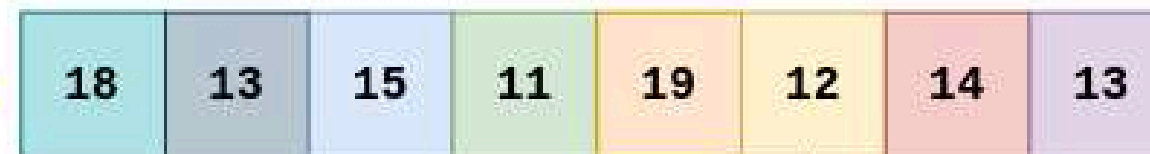
`min_value = 11`

`max_value = 19`

`range_value = 9`

`i = 5`

array



pigeon
holes



Steps:

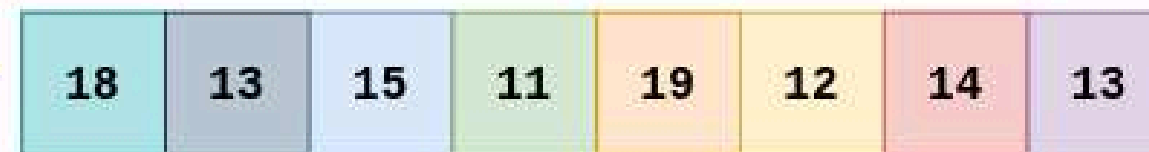
5. for `i` from 0 to `n`:

`push array[i] into pigeonholes[array[i] - min]`

Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array



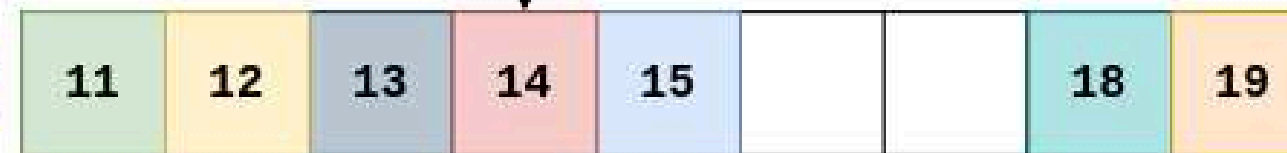
min_value = 11

max_value = 19

range_value = 9

i = 6

pigeon
holes



Steps:

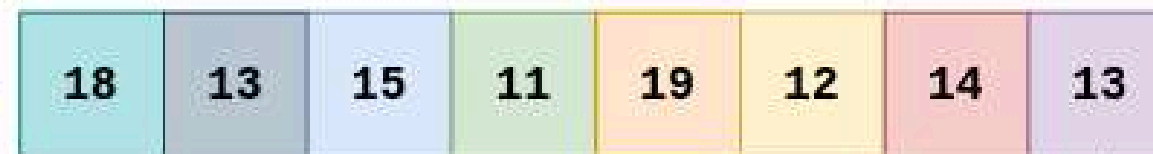
5. for i from 0 to n:

 push array[i] into pigeonholes[array[i] - min]

Pigeonhole Sort

Pigeonhole sort - Shifting values to pigeonholes

array



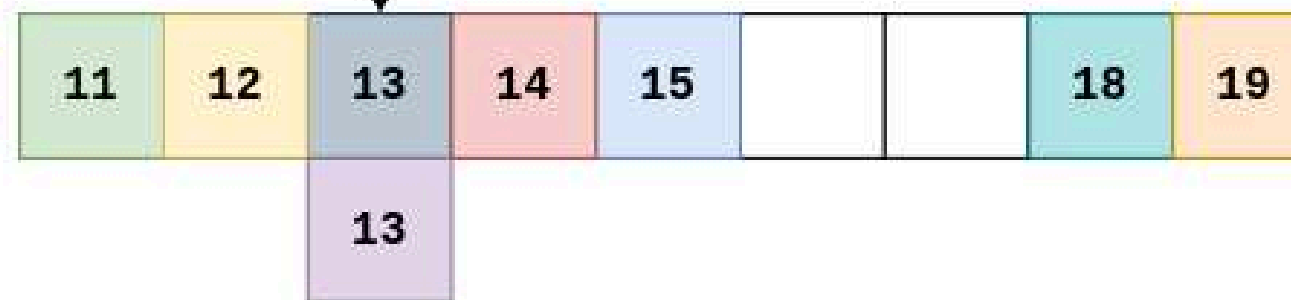
min_value = 11

max_value = 19

range_value = 9

i = 7

pigeon
holes



Steps:

5. for i from 0 to n:

push array[i] into pigeonholes[array[i] - min]

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array

18	13	15	11	19	12	14	13
----	----	----	----	----	----	----	----

pigeon
holes

11	12	13	14	15			18	19
		13						

min_value = 11

max_value = 19

range_value = 9

i = 8

arr_index = 0

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

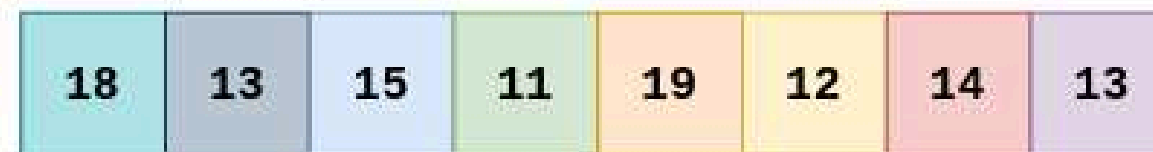
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

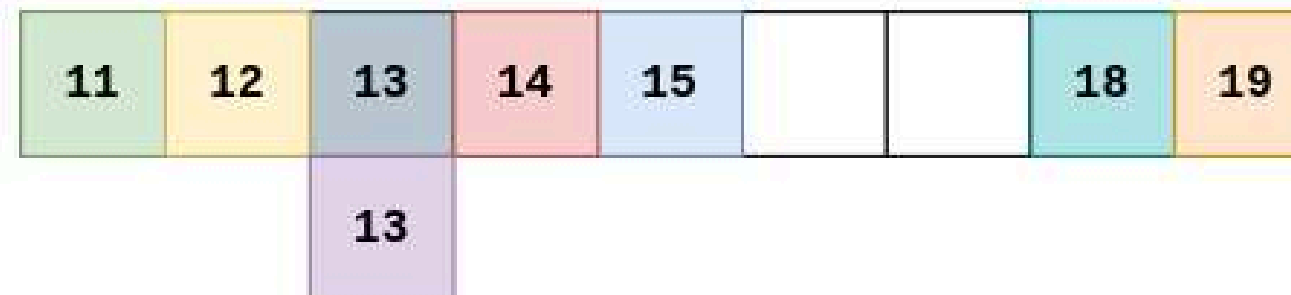
max_value = 19

range_value = 9

i = 0

arr_index = 0

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

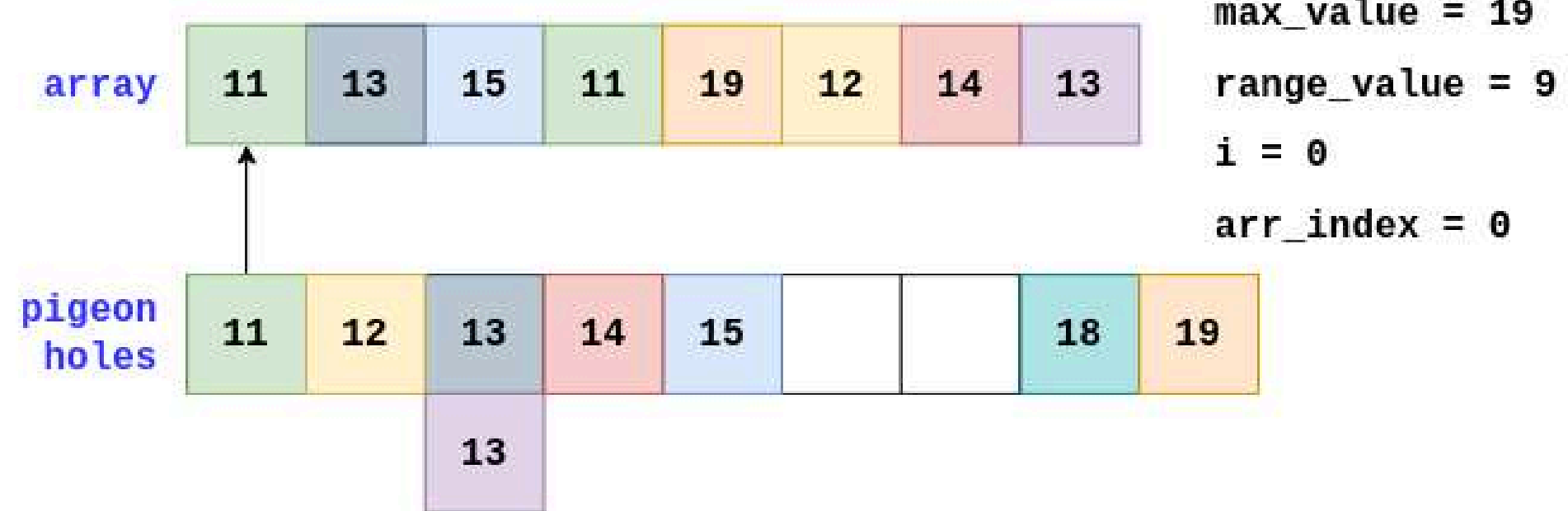
while(pigeonhole[i] is not empty):

place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

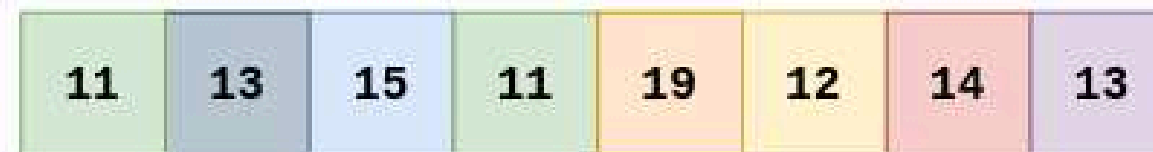
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

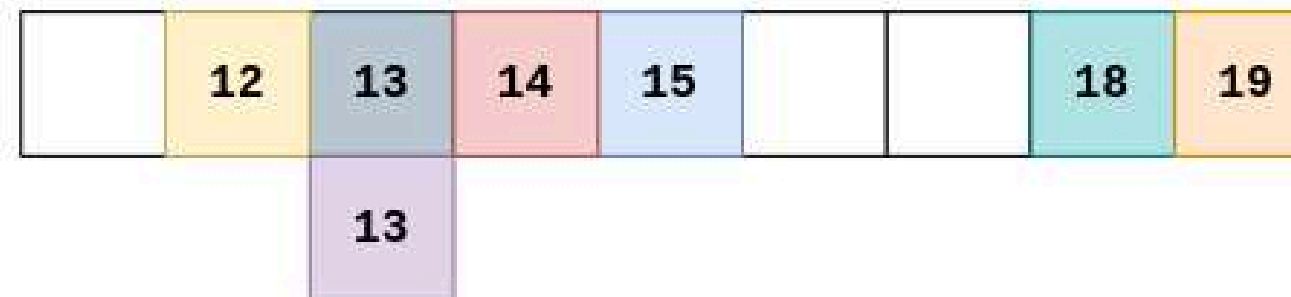
max_value = 19

range_value = 9

i = 1

arr_index = 1

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

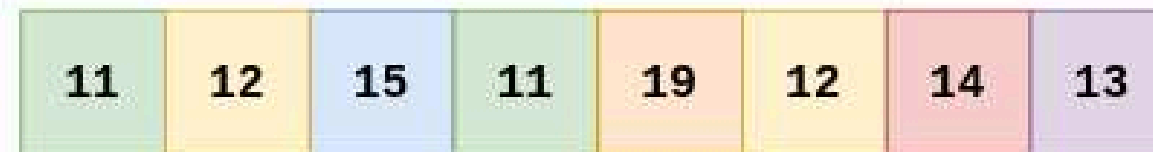
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

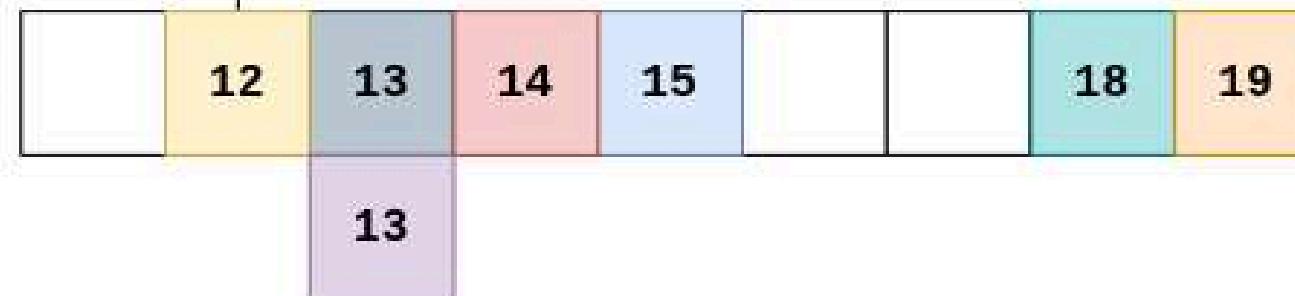
Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



pigeon
holes



min_value = 11

max_value = 19

range_value = 9

i = 1

arr_index = 1

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

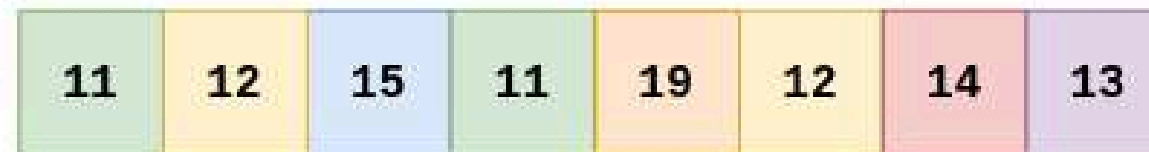
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

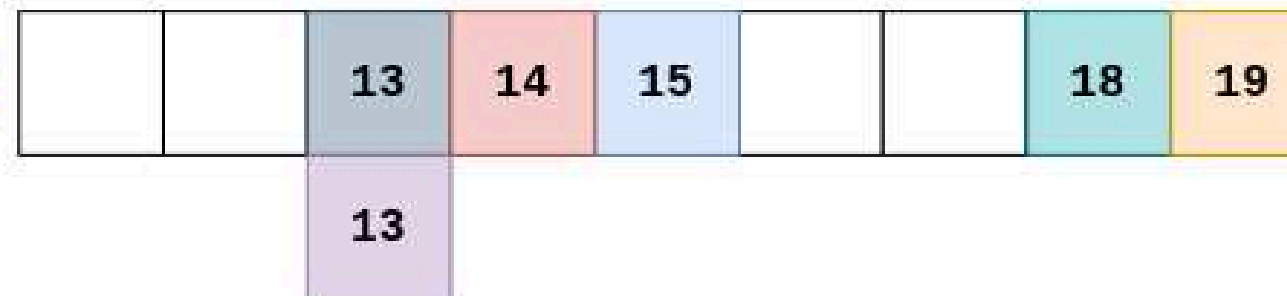
max_value = 19

range_value = 9

i = 2

arr_index = 2

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

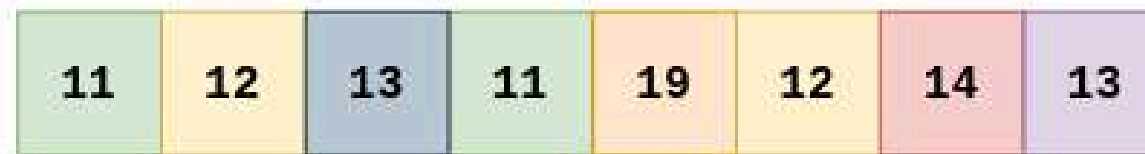
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

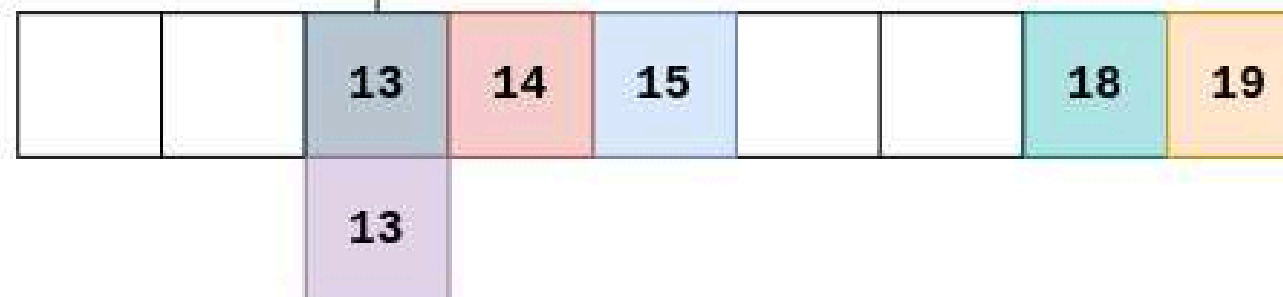
Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



pigeon
holes



min_value = 11

max_value = 19

range_value = 9

i = 2

arr_index = 2

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array

11	12	13	11	19	12	14	13
----	----	----	----	----	----	----	----

min_value = 11

max_value = 19

range_value = 9

i = 2

arr_index = 3

pigeon
holes

		13	14	15			18	19
--	--	----	----	----	--	--	----	----

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

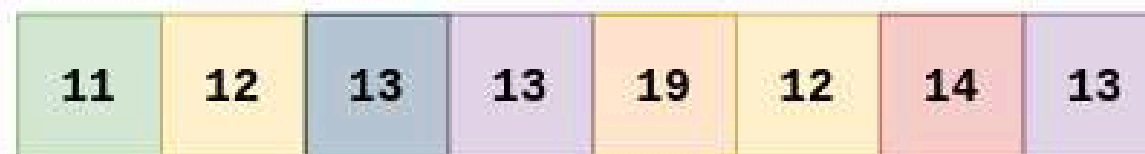
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



pigeon
holes



min_value = 11

max_value = 19

range_value = 9

i = 2

arr_index = 3

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

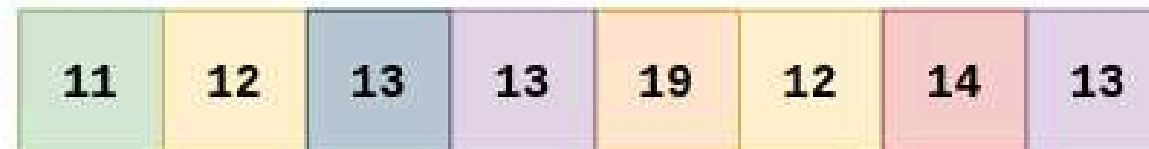
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

max_value = 19

range_value = 9

i = 3

arr_index = 4

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

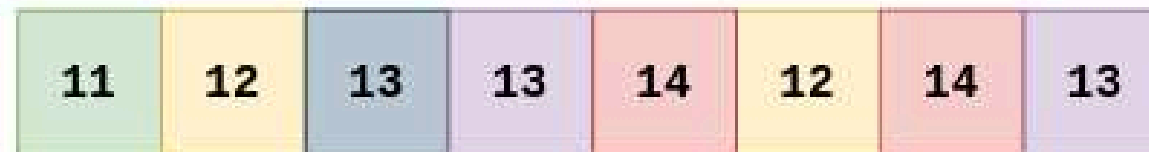
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



pigeon
holes



min_value = 11

max_value = 19

range_value = 9

i = 3

arr_index = 4

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

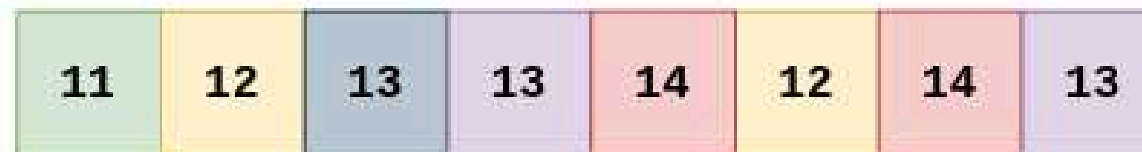
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

max_value = 19

range_value = 9

i = 4

arr_index = 5

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

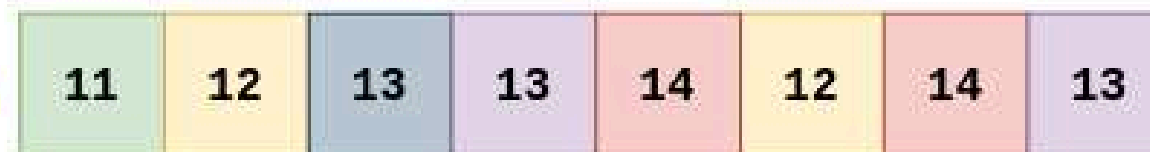
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

max_value = 19

range_value = 9

i = 4

arr_index = 5

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

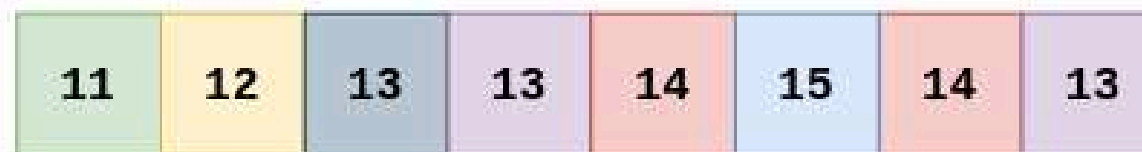
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

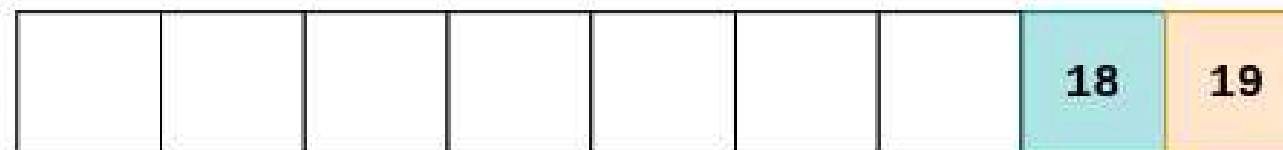
max_value = 19

range_value = 9

i = 5

arr_index = 6

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

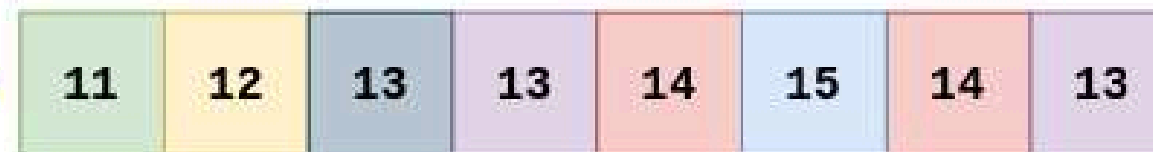
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

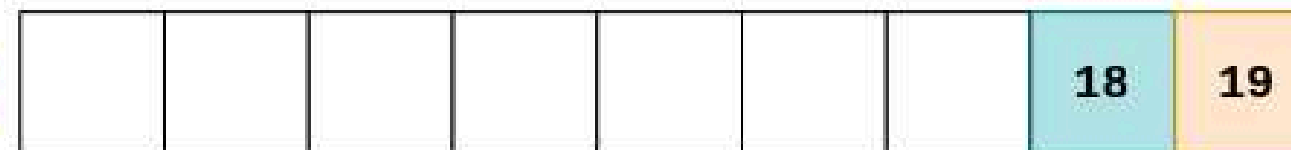
max_value = 19

range_value = 9

i = 6

arr_index = 6

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

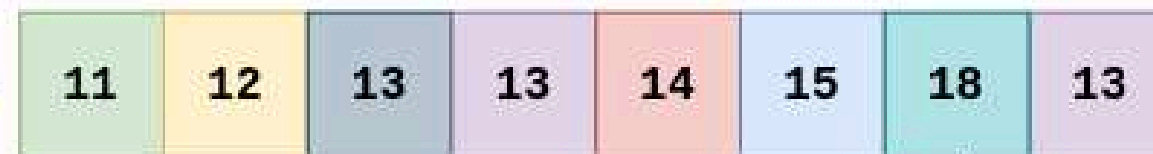
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

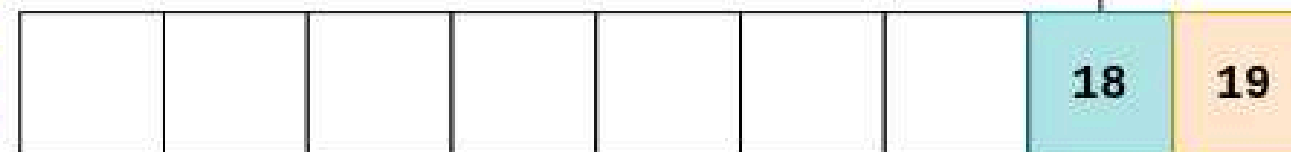
Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



pigeon
holes



min_value = 11
max_value = 19
range_value = 9
i = 7
arr_index = 6

Steps:

6. arr_index = 0

for *i* from 0 to range:

while(pigeonhole[*i*] is not empty):

place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array

11	12	13	13	14	15	18	13
----	----	----	----	----	----	----	----

pigeon
holes

								19
--	--	--	--	--	--	--	--	----

min_value = 11

max_value = 19

range_value = 9

i = 7

arr_index = 7

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

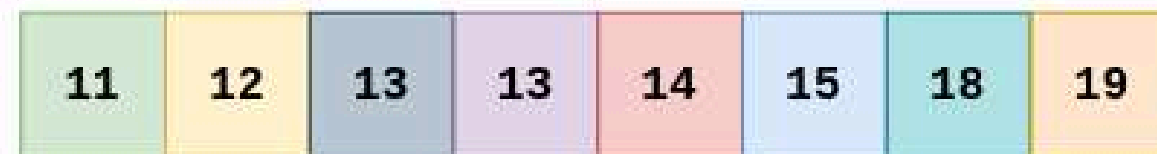
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array



min_value = 11

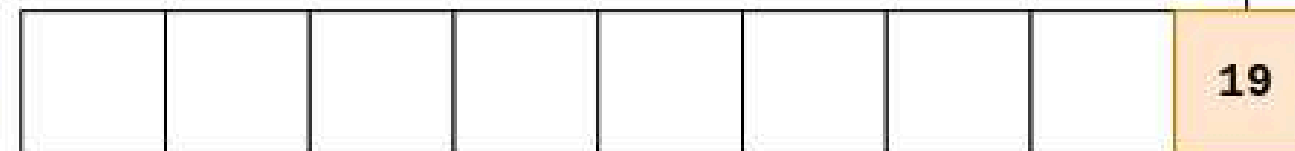
max_value = 19

range_value = 9

i = 7

arr_index = 7

pigeon
holes



Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - Shifting values back to array

array

11	12	13	13	14	15	18	19
----	----	----	----	----	----	----	----

min_value = 11

max_value = 19

range_value = 9

i = 8

arr_index = 8

pigeon
holes

--	--	--	--	--	--	--	--	--

Steps:

6. arr_index = 0

for i from 0 to range:

while(pigeonhole[i] is not empty):

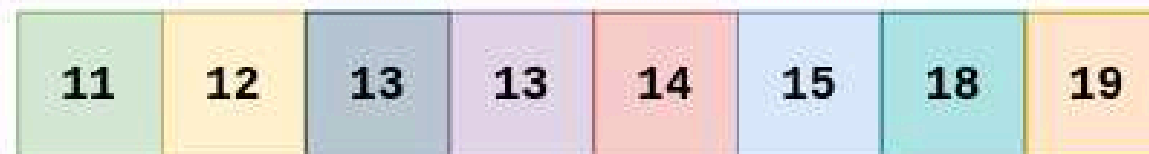
place pigeonhole elements in array[arr_index]

arr_index := arr_index + 1

Pigeonhole Sort

Pigeonhole sort - The array is sorted

array



min_value = 11

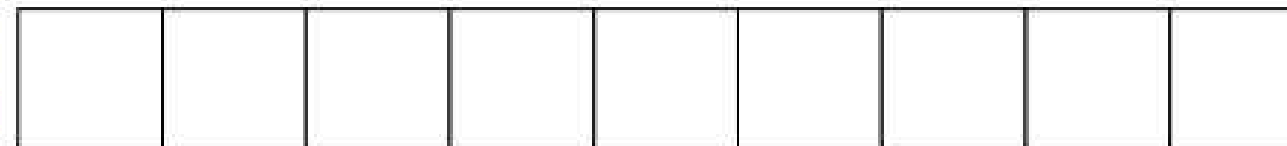
max_value = 19

range_value = 9

i = 8

arr_index = 8

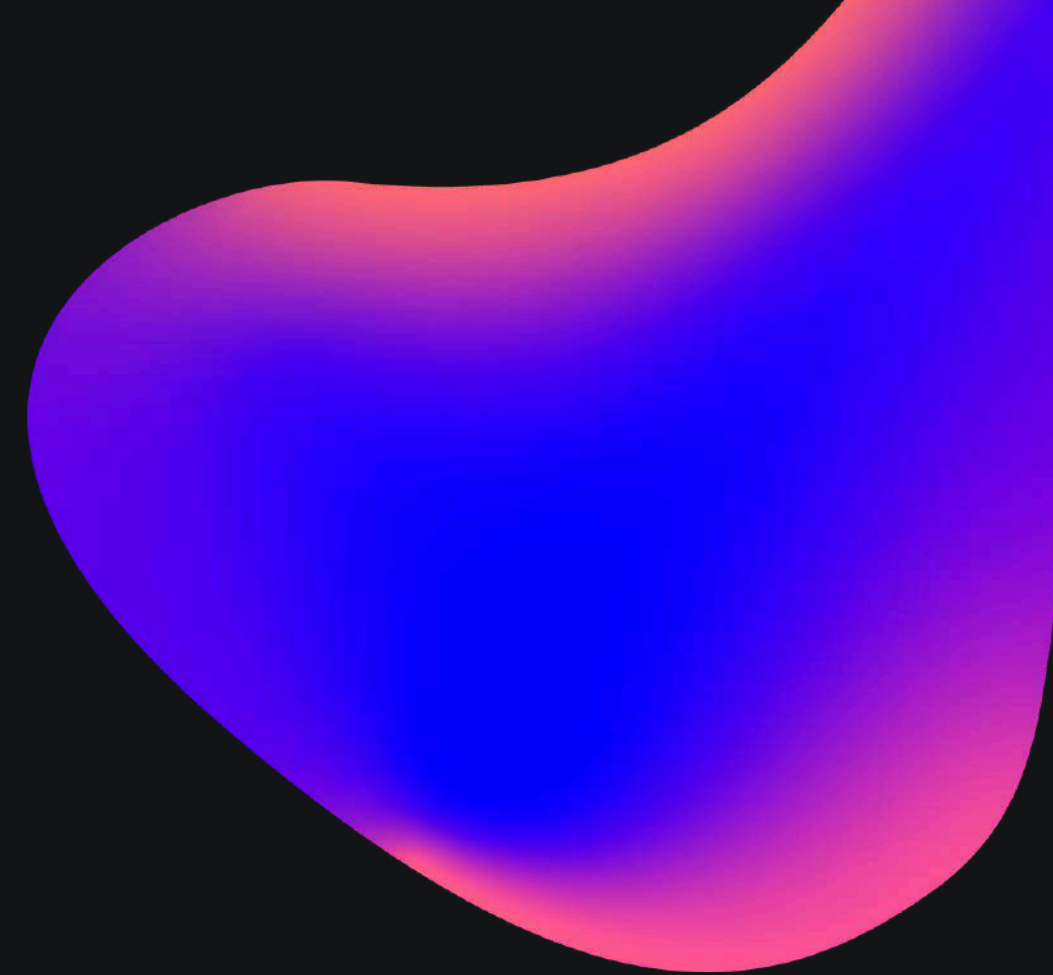
pigeon
holes



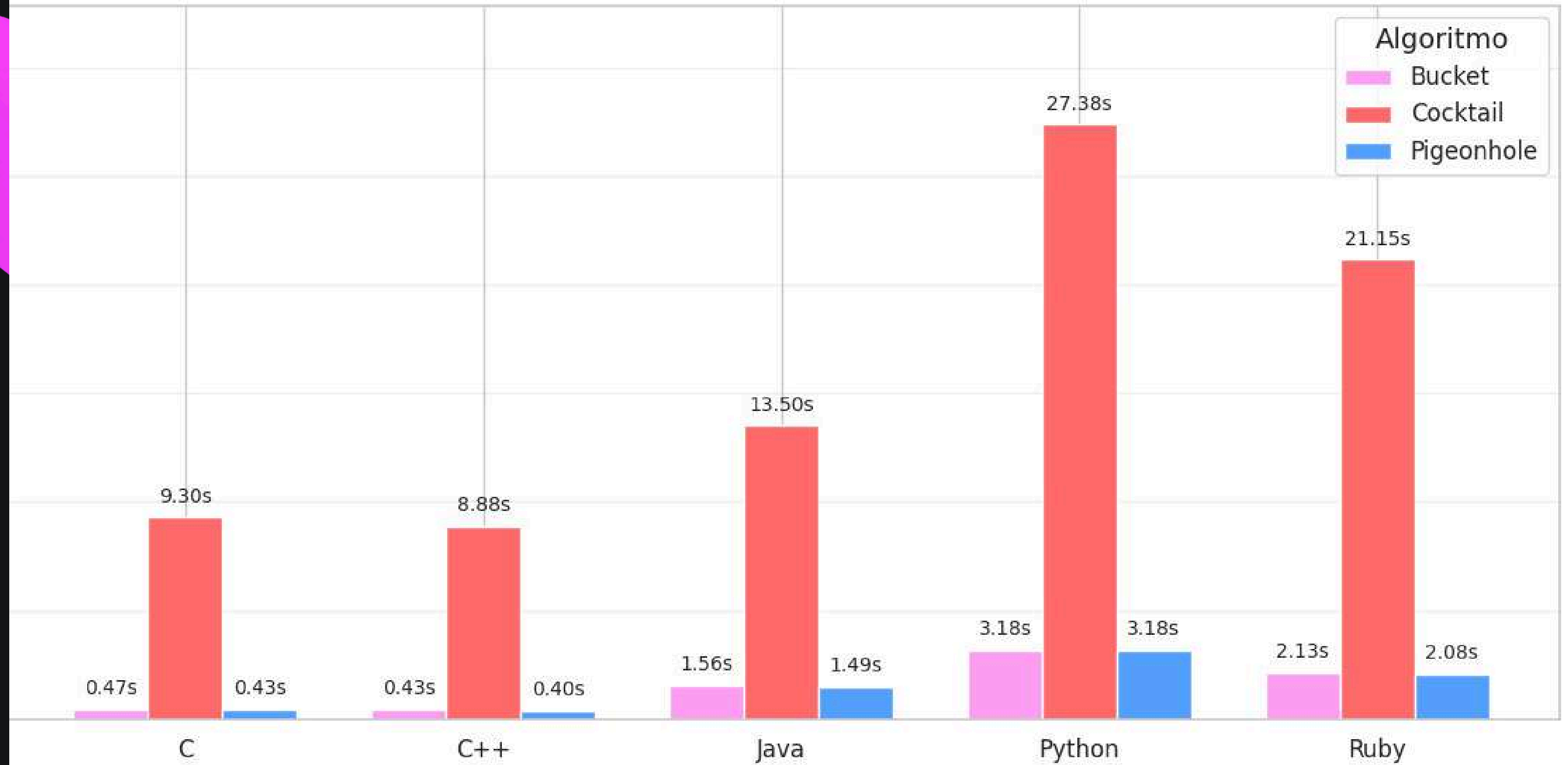
The array is sorted!

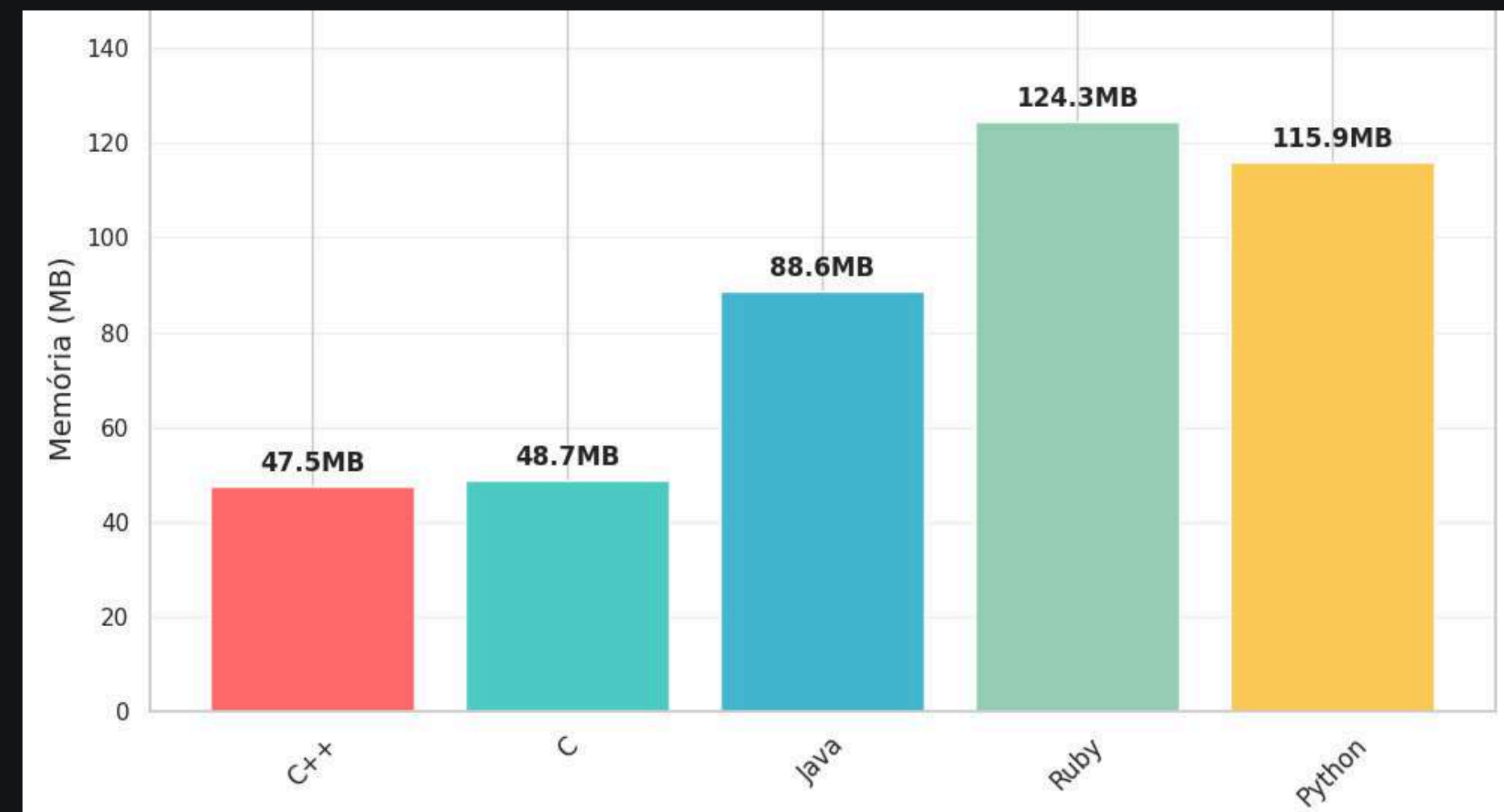
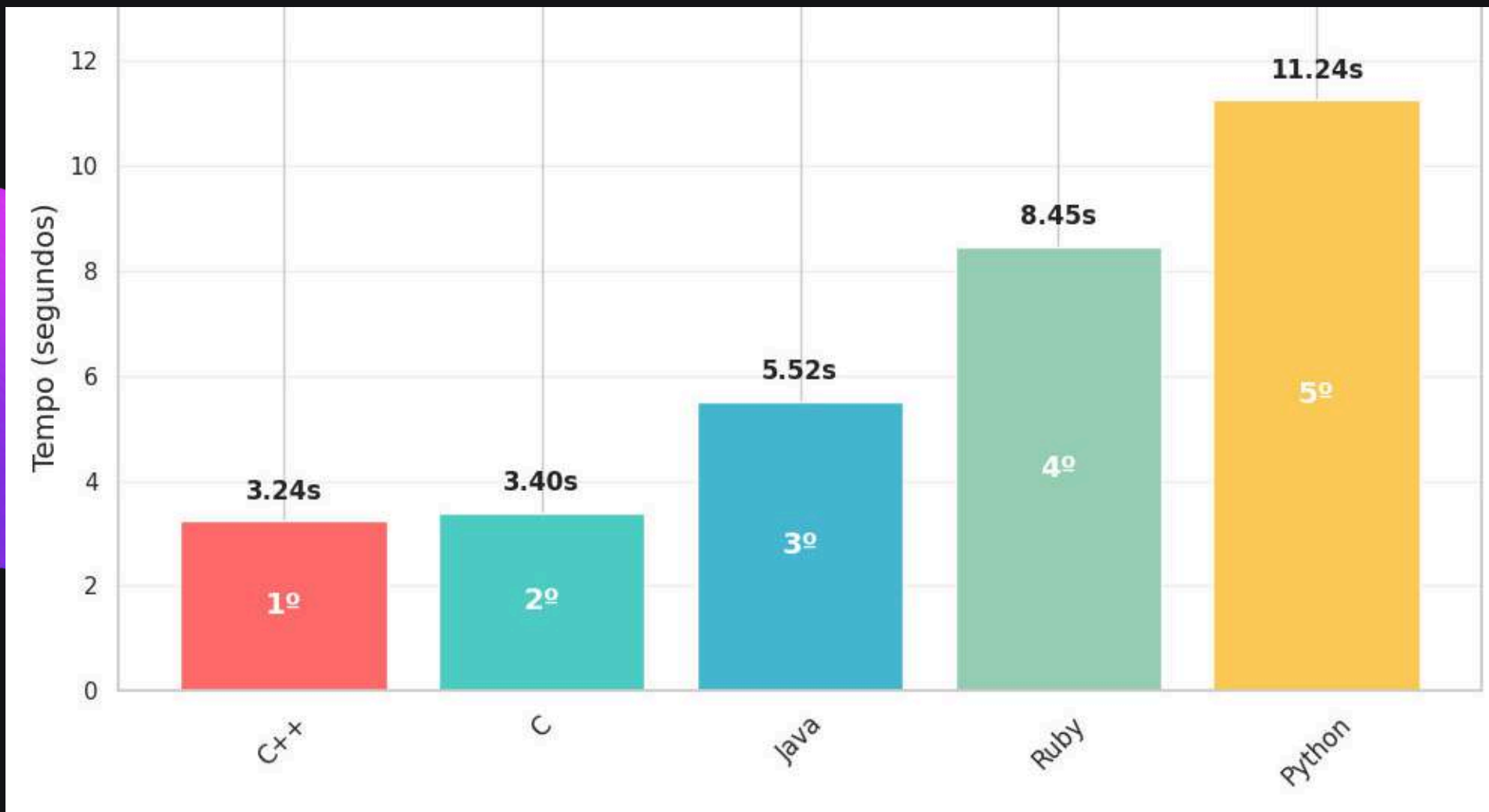
You can see the pigeonhole sort performs the stable sorting. The positions of same element are kept unchanged in the sorted array.

Análise de desempenho

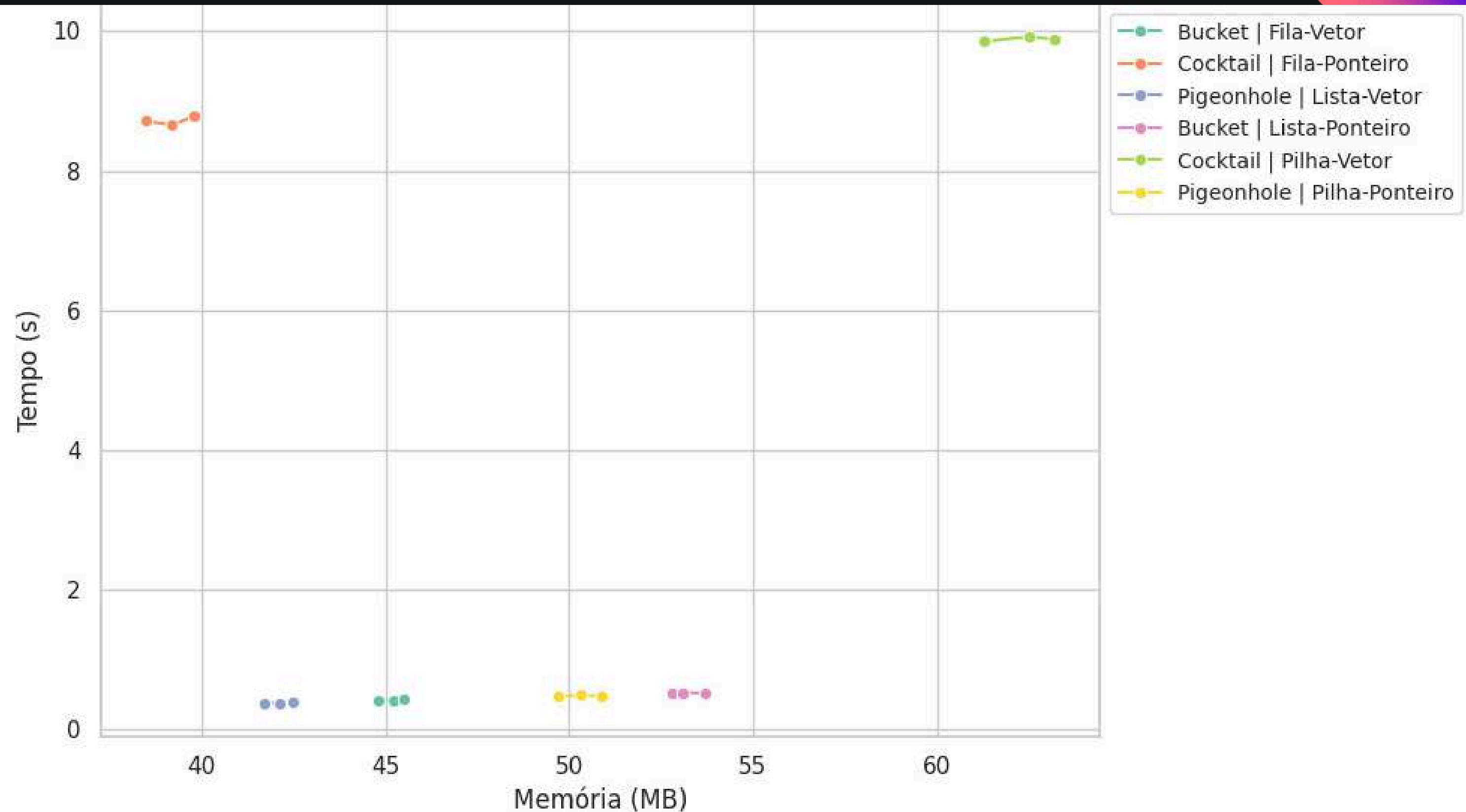


Tempo Médio por Algoritmo e Linguagem

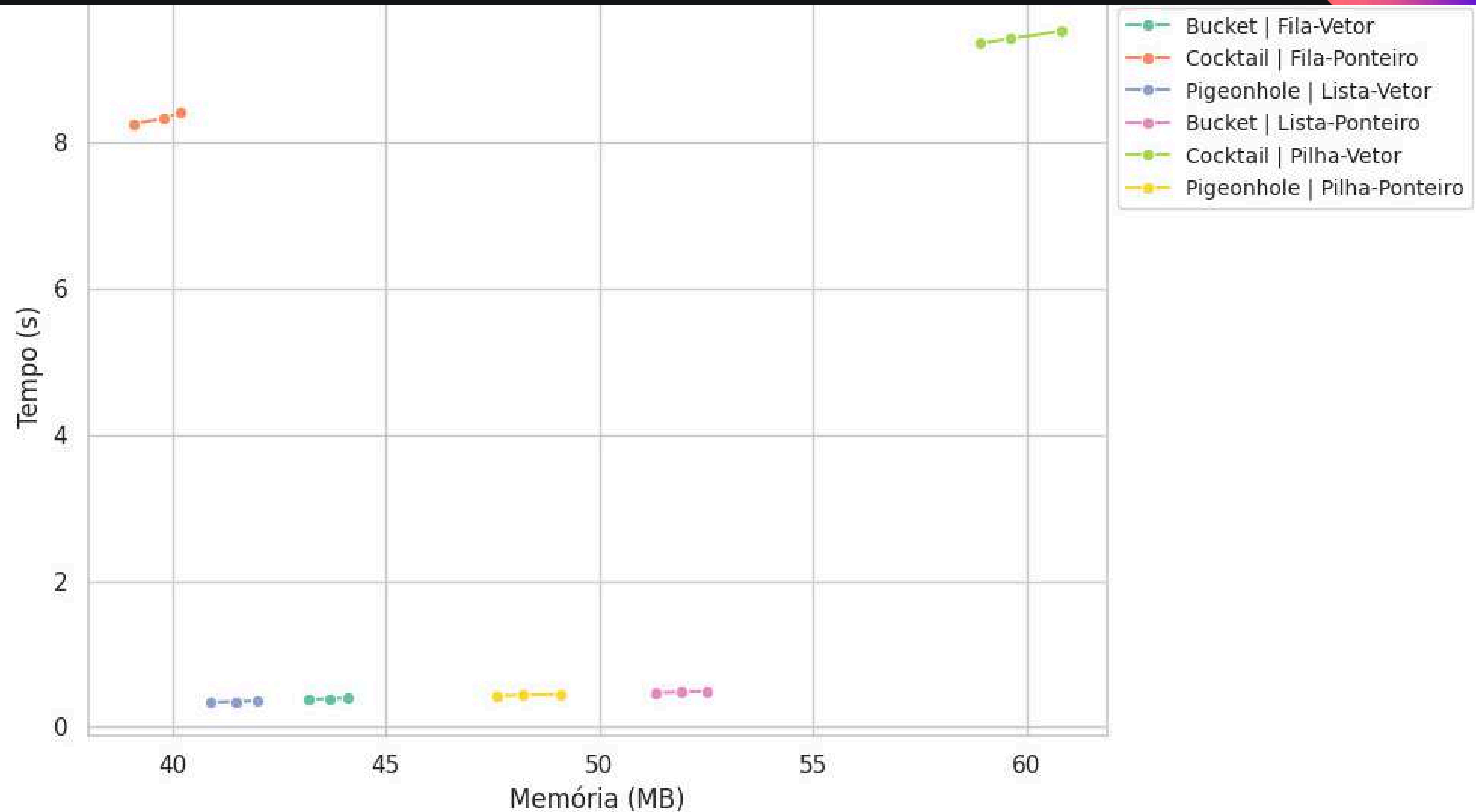




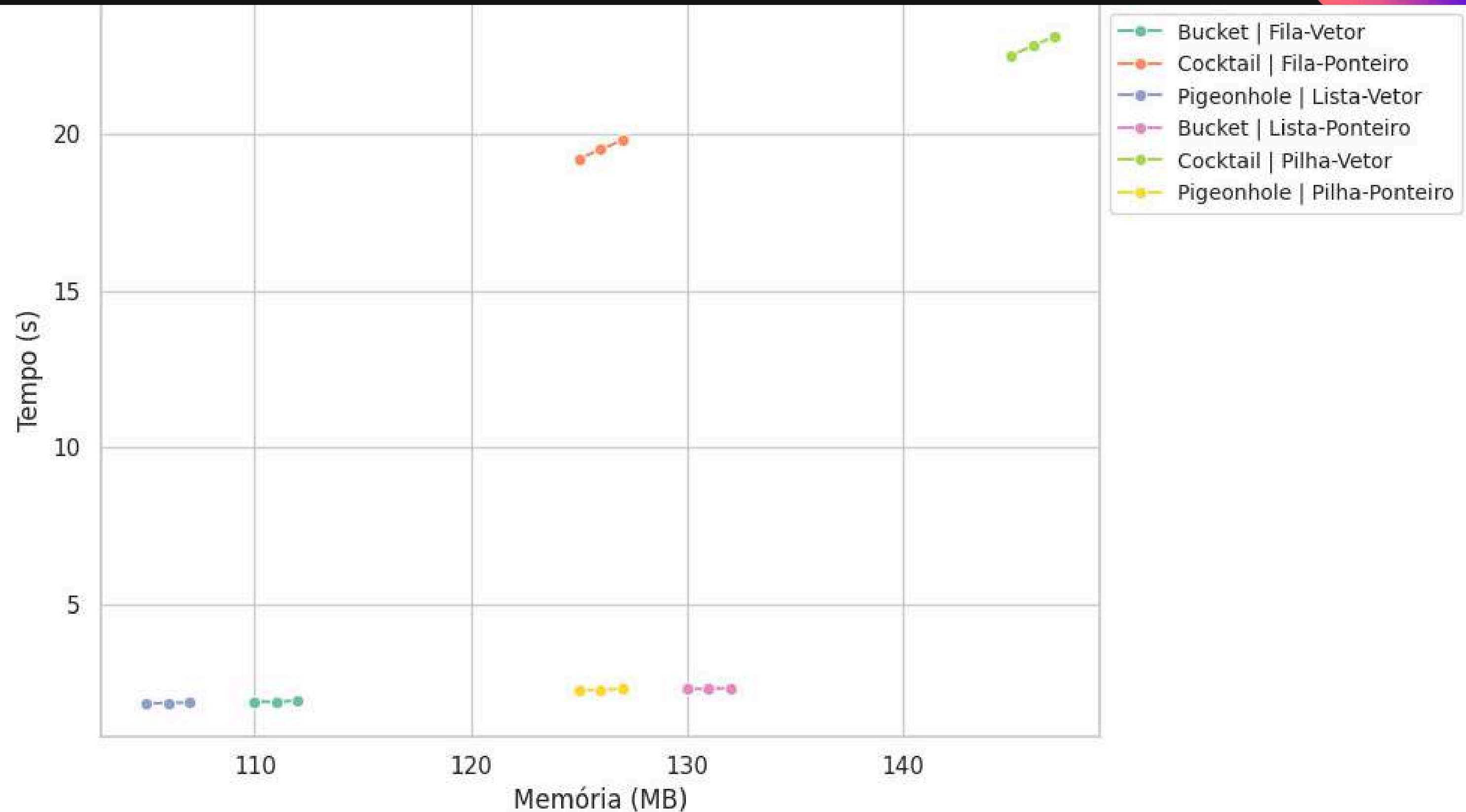
Tempo x Memória - C



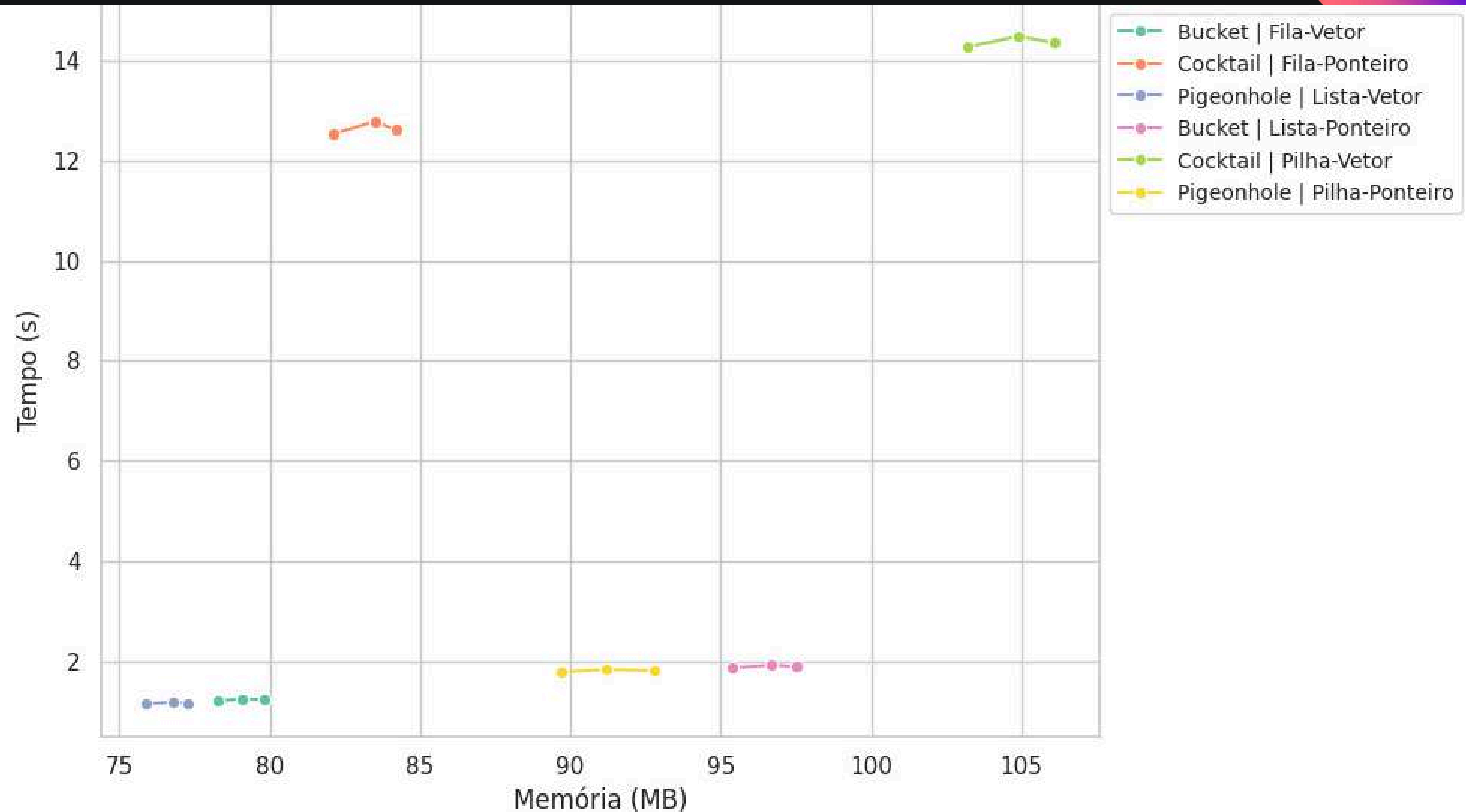
Tempo x Memória - C++



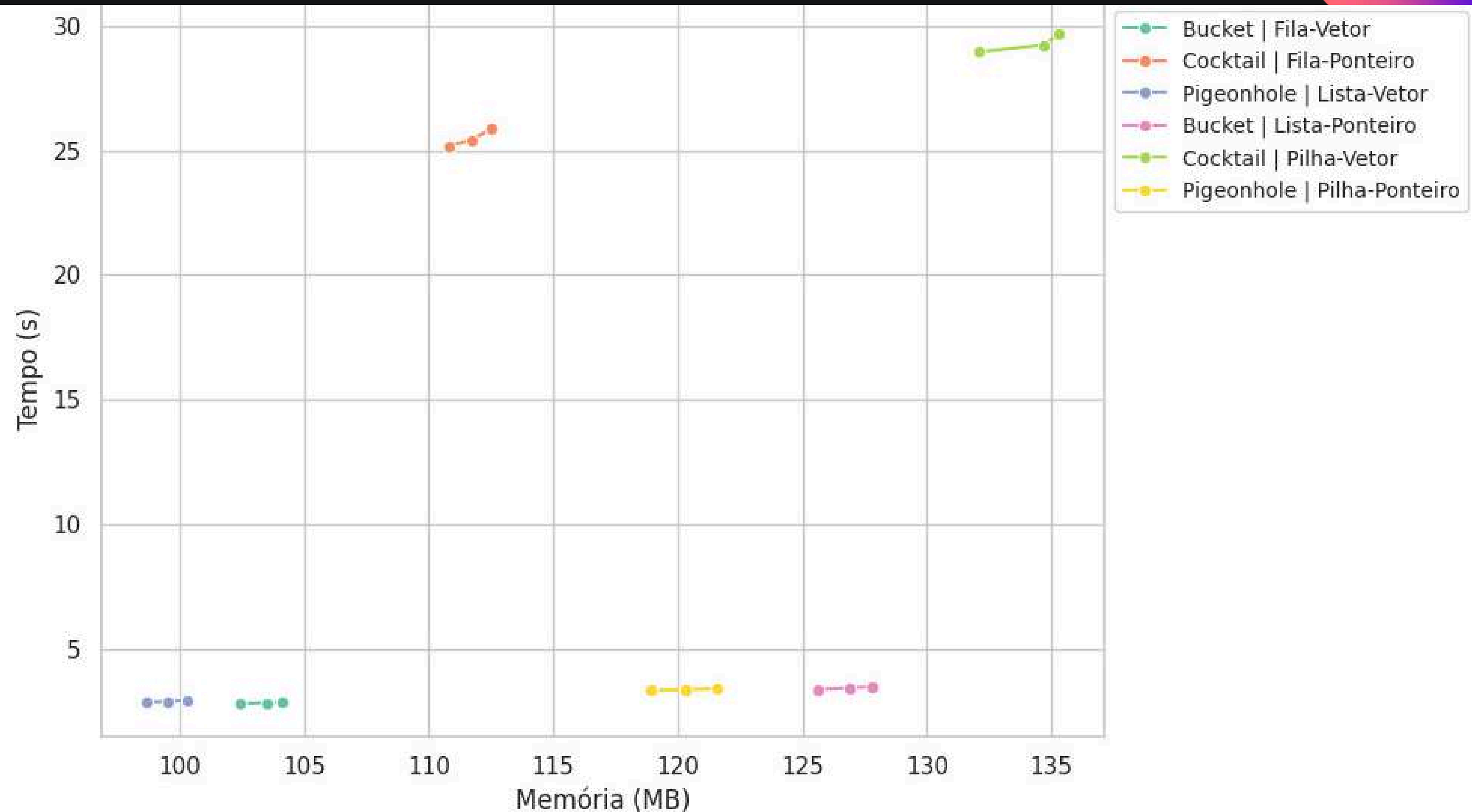
Tempo x Memória - Ruby



Tempo x Memória - Java



Tempo x Memória - Python



Aplicabilidade

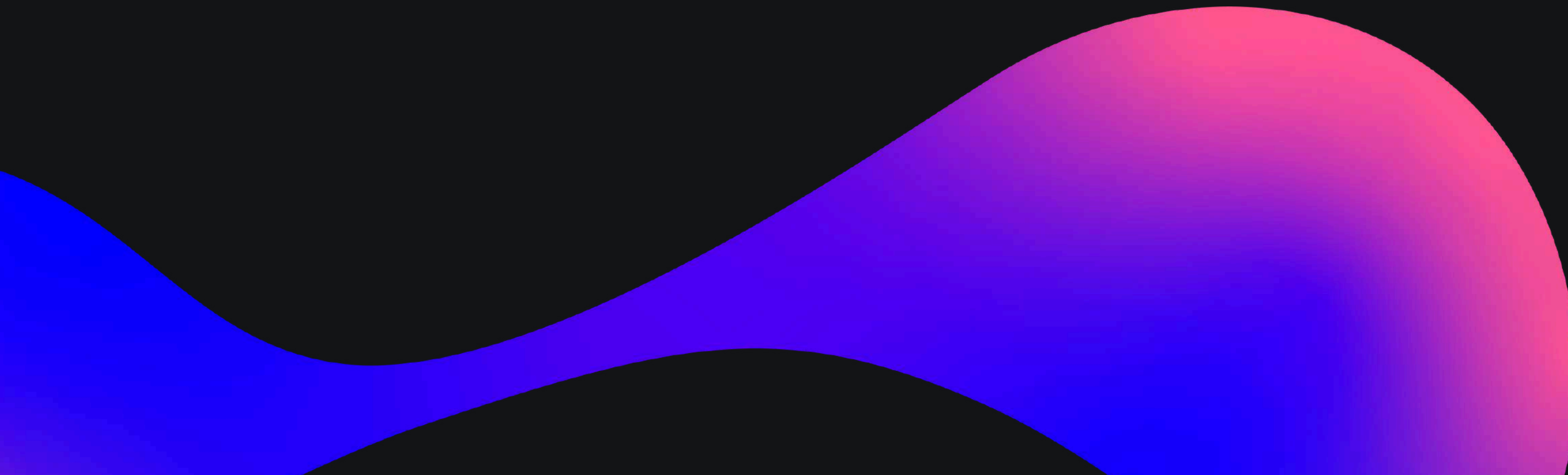


Cocktail Sort

- Algoritmo simples e intuitivo
- Complexidade quadrática
- Usado em práticas educacionais

Bucket Sort

- Uso de métodos auxiliares
- Complexidade baseada no método auxiliar
- Eficiente para grande conjunto de dados
- Pode atingir até tempos lineares





Pigeonhole Sort

- Intervalo e número de elementos aproximados
- Complexidade é a soma dos dois fatores principais do algoritmo
- Performance ligada à densidade dos dados



OBRIGADO!