Algoritmos e Programação de Computadores

Ordenação: Merge Sort

Profa. Sandra Avila
Instituto de Computação (IC/Unicamp)

MC102, 17 Junho, 2019

Introdução

Vamos usar a técnica de **recursão** para resolver o problema de **ordenação**.

- Problema:
  - Temos uma lista `v` de inteiros de tamanho `n`.
  - Devemos deixar `v` ordenada em ordem crescente de valores.

Dividir e Conquistar

- Temos que resolver um problema `P` de tamanho `n`.
- **Dividir**: Quebramos `P` em sub-problemas menores.
- Resolvemos os sub-problemas de forma recursiva.
- **Conquistar**: Unimos as soluções dos sub-problemas para obter solução do problema maior `P`.

Quick Sort

Quick Sort

- Vamos supor que devemos ordenar uma lista de uma posição `inicial` até `fim`.
- **Dividir**:
  - Escolha em elemento especial da lista chamado `pivô`.
  - Particione a lista em uma posição `pos` tal que todos elementos de `inicial` até `pos - 1` são menores ou iguais do que o `pivô`, e todos elementos de `pos` até `fim` são maiores ou iguais ao `pivô`.

Quick Sort: Particionamento

A função retorna a posição de partição. Ela considera o último elemento como o `pivô`.

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
    # o laço para quando inicio == fim => checamos o vetor inteiro
        while (inicio < fim) and (v[inicio] <= pivo):
        # acha posição de elemento maior que pivo
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
        # acha posição de elemento menor ou igual que pivo
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio] # troca elementos de posição
    return inicio
```

Quick Sort

- Resolvemos o problema de ordenação de forma recursiva para estas duas sub-listas (uma de `inicial` até `pos-1` e a outra de `pos` até `fim`).
- **Conquistar**: Nada a fazer já que a lista estará ordenada devido à fase de divisão.

Random Quick Sort

- A única diferença é que escolhemos um elemento aleatório.
- Tal elemento é trocado com o que está no fim (será o pivô).

```
import random
def randomQuickSort(v, inicio, fim):
    if (inicio < fim):
        j = random.randint(inicio, fim)
        v[j], v[fim] = v[fim], v[j]
        pos = particiona(v, inicio, fim)
        randomQuickSort(v, inicio, pos-1)
        randomQuickSort(v, pos, fim)
```

Exercícios

1. Aplique o algoritmo de particionamento sobre o vetor `(13,19,9,5,12,21,7,4,11,2,6,6)` com pivô igual a `6`.
2. Qual o valor retornado pelo algoritmo de particionamento se todos os elementos do vetor tiverem valores iguais?
3. **Faça uma execução passo-a-passo do `quickSort` com o vetor `(4,3,6,7,9,10,5,8)`.**
4. Modifique o algoritmo `quickSort` para ordenar vetores em ordem decrescente.

Merge Sort

Dividir e Conquistar

- Temos que resolver um problema `P` de tamanho `n`.
- **Dividir**: Quebramos `P` em sub-problemas menores.
- Resolvemos os sub-problemas de forma recursiva.
- **Conquistar**: Unimos as soluções dos sub-problemas para obter solução do problema maior `P`.

Merge Sort: Ordenação por Intercalação

- O Merge Sort é um algoritmo baseado na técnica **dividir e conquistar**.
- Neste caso temos que ordenar uma lista de tamanho `n`.
  - **Dividir**: Dividimos a lista de tamanho `n` em duas sub-listas de tamanho aproximadamente iguais (de tamanho `n/2`).
  - Resolvemos o problema de ordenação de forma recursiva para estas duas sub-listas.
  - **Conquistar**: Com as duas sub-listas ordenadas, construímos uma lista ordenada de tamanho `n` ordenado.

Merge Sort: Ordenação por Intercalação

- **Conquistar**: Dados duas listas `v1` e `v2` ordenadas, como obter uma outra lista ordenada contendo os elementos de `v1` e `v2` ?

3	5	7	10	11	12

4	6	8	9	11	13	14

3	4	5	6	7	8	9	10	11	11	12	13	14

Merge: Fusão

- A ideia é executar um laço que testa em cada iteração quem é o menor elemento dentre `v1[i]` e `v2[j]`, e copia este elemento para uma nova lista.
- Durante a execução deste laço podemos chegar em uma situação onde todos os elementos de uma das listas (`v1` ou `v2`) foram todos avaliados. Neste caso terminamos o laço e copiamos os elementos restantes da outra lista.

Merge: Fusão

```
def merge (v1, v2): # devolve lista com fusão de v1 e v2
    i = 0; j = 0; # índice de v1 e v2 resp.
    v3 = []
    while (i < len(v1) and j < len(v2)): # enquanto não avaliou completamente
        if (v1[i] <= v2[j]):      # um dos vetores, copia menor elemento para v3
            v3.append(v1[i])
            i = i + 1
        else:
            v3.append(v2[j])
            j = j + 1
    while (i < len(v1)): # copia resto de v1
        v3.append(v1[i])
        i = i + 1
    while (j < len(v2)): # copia resto de v2
        v3.append(v2[j])
        j = j + 1
    return v3
```

Merge: Fusão

- A função descrita recebe duas listas ordenadas e devolve uma terceira contendo todos os elementos em ordem.
- Porém no Merge Sort faremos a intercalação de sub-listas de uma mesma lista.
- Isto evita a criação de várias listas durante as várias chamadas recursivas, melhorando a performance do algoritmo.

Merge: Fusão

- Teremos posições `inicio`, `meio`, `fim` de uma lista e devemos fazer a intercalação das duas sub-listas: uma de `inicio` até `meio`, e outra de `meio+1` até `fim`.
    - Para isso a função utiliza uma lista auxiliar, que receberá o resultado da intercalação, e que no final é copiado para a lista original a ser ordenada.

Merge: Fusão

- Faz intercalação de pedaços de `v`. No fim `v` estará ordenada entre as posições `inicio` e `fim`.

```
def merge (v, inicio, meio, fim, aux):
     i = inicio; j = meio+1; k = 0; # indices da metade inf, sup e aux respc.
     while (i <= meio and j <= fim): # enquanto não avaliou completamente um dos
          if (v[i] <= v[j]):         # vetores, copia menor elemento para aux
               aux[k] = v[i]
               k = k + 1
               i = i + 1
          else:
               aux[k] = v[j]
               k = k + 1
               j = j + 1
```

Merge: Fusão

- Faz intercalação de pedaços de `v`. No fim `v` estará ordenada entre as posições `inicio` e `fim`.

```
    while (i <= meio): # copia resto da primeira sub-lista
         aux[k] = v[i]
         k = k + 1
         i = i + 1
    while (j <= fim): # copia resto da segunda sub-lista
         aux[k] = v[j]
         k = k + 1
         j = j + 1
    i = inicio; k = 0;
    while (i <= fim): # copia lista ordenada aux para v
         v[i] = aux[k]
         i = i + 1
         k = k + 1
```

```
def merge (v, inicio, meio, fim, aux):
      i = inicio; j = meio+1; k = 0;   # indices da metade inf, sup e aux respc.
      while (i <= meio and j <= fim):  # enquanto não avaliou completamente um dos
            if (v[i] <= v[j]):         # vetores, copia menor elemento para aux
                  aux[k] = v[i]
                  k = k + 1
                  i = i + 1
            else:
                  aux[k] = v[j]
                  k = k + 1
                  j = j + 1
      while (i <= meio): # copia resto da primeira sub-lista
            aux[k] = v[i]
            k = k + 1
            i = i + 1
      while (j <= fim): # copia resto da segunda sub-lista
            aux[k] = v[j]
            k = k + 1
            j = j + 1
      i = inicio; k = 0;
      while (i <= fim): # copia lista ordenada aux para v
            v[i] = aux[k]
            i = i + 1
            k = k + 1
```

Merge Sort

- O Merge Sort resolve de forma recursiva dois sub-problemas, cada um contendo uma metade da lista original.
- Com a resposta das chamadas recursivas podemos chamar a função `merge` para obter uma lista ordenada.

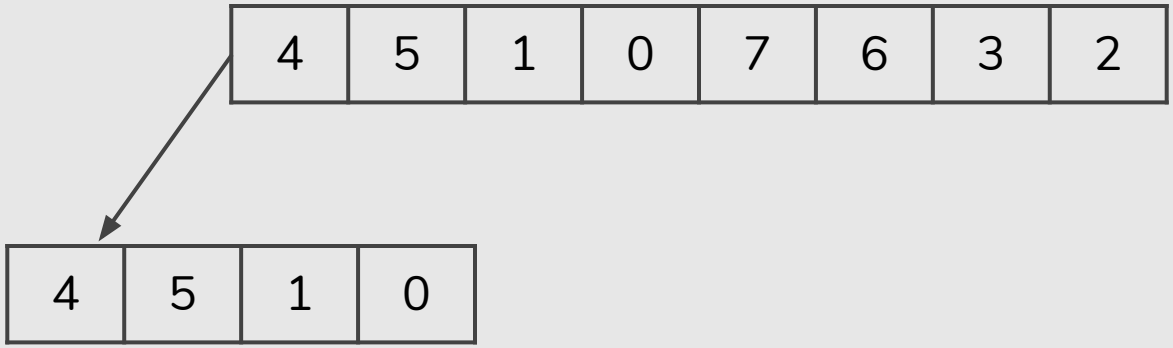
Merge Sort

```
def mergeSort(v, inicio, fim, aux):
    meio = (fim + inicio) // 2
    if (inicio < fim): # lista tem pelo menos 2 elementos
                       # para ordenar
        mergeSort(v, inicio, meio, aux)
        mergeSort(v, meio+1, fim, aux)
        merge(v, inicio, meio, fim, aux)
```

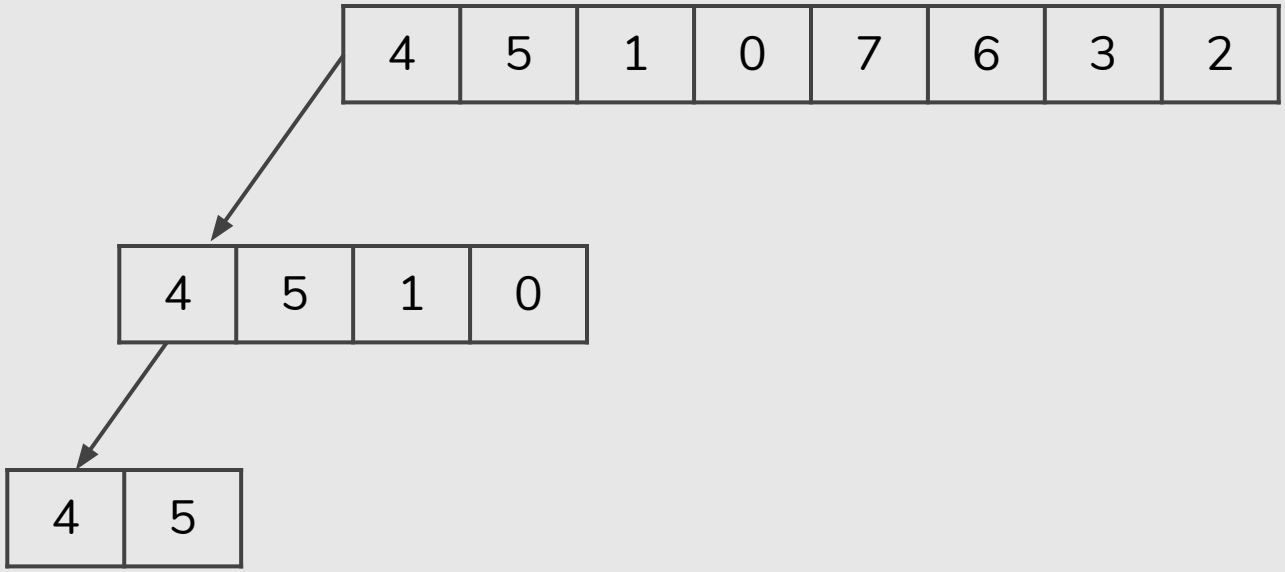
Merge Sort

4	5	1	0	7	6	3	2

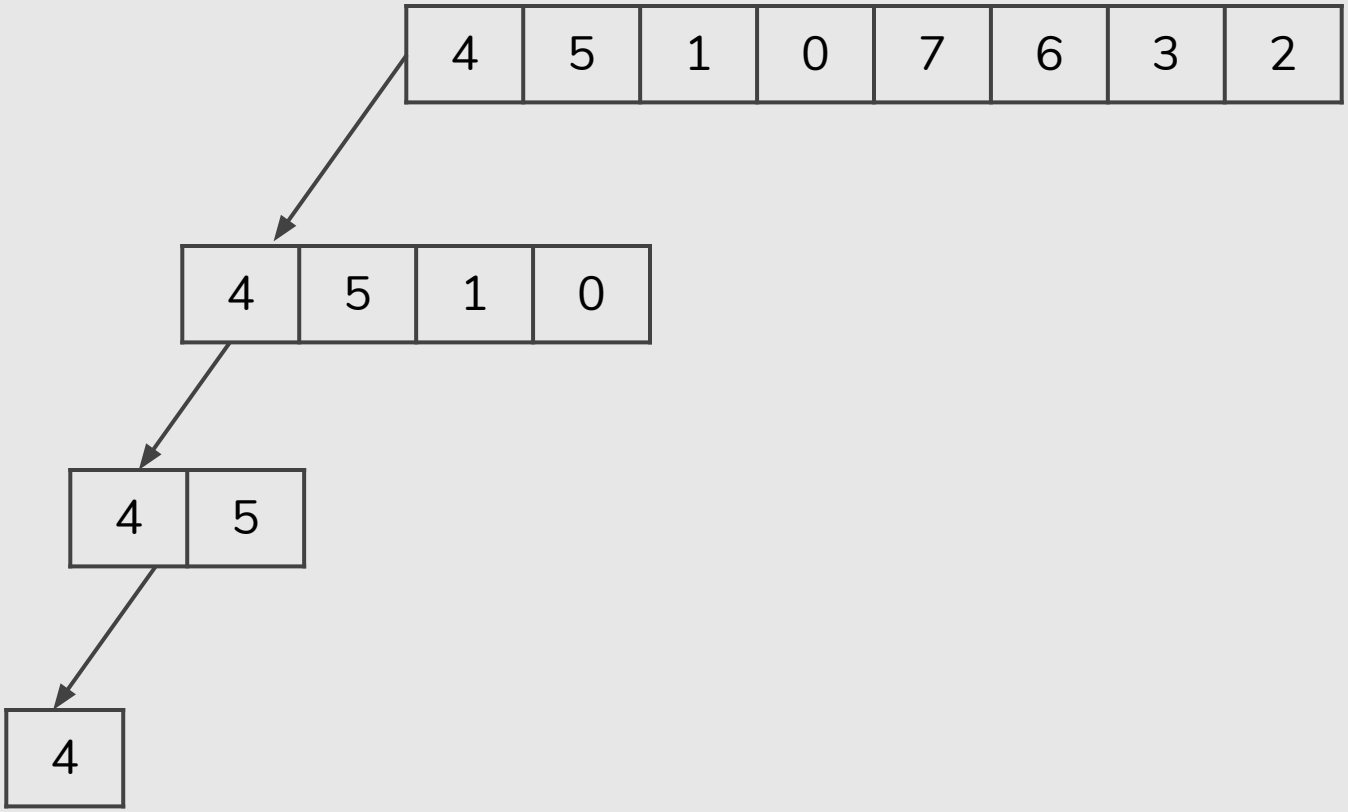
Merge Sort



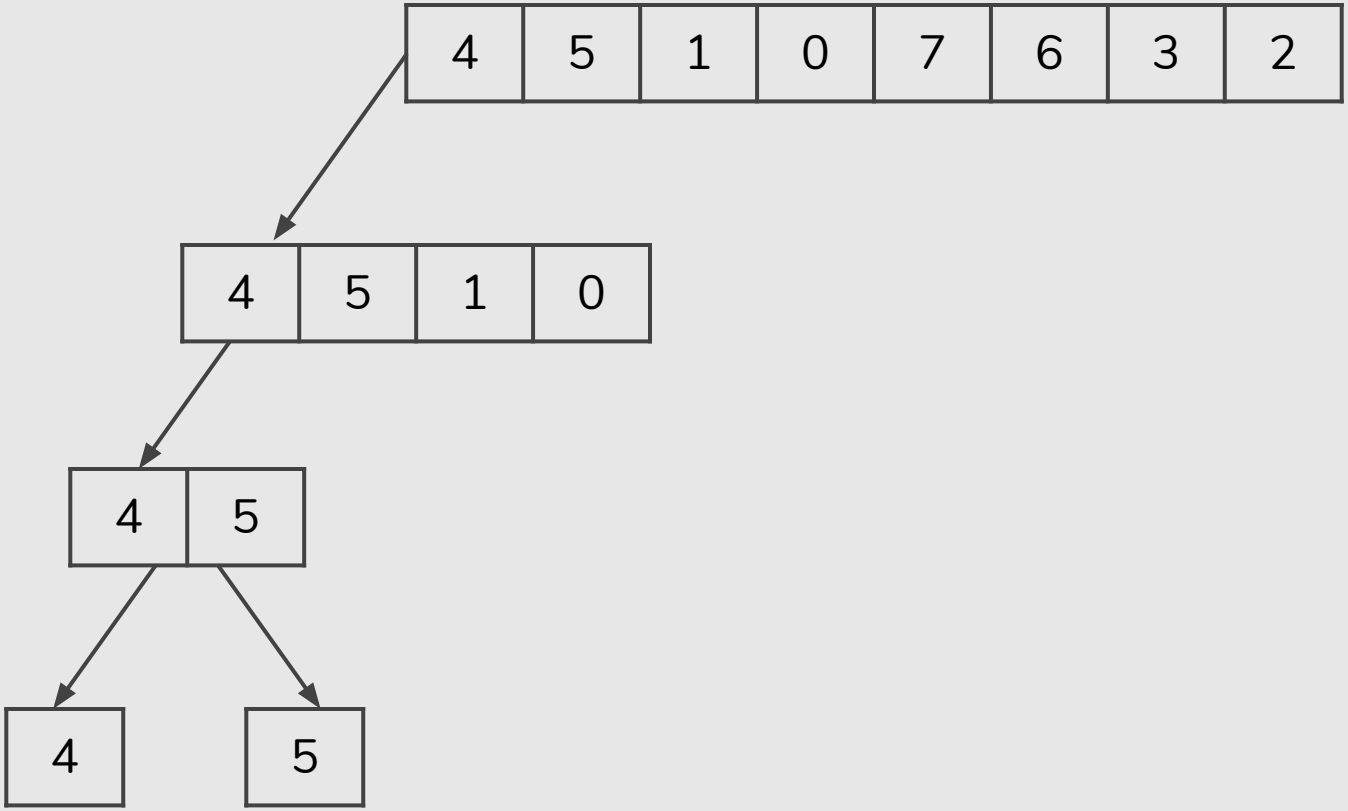
Merge Sort



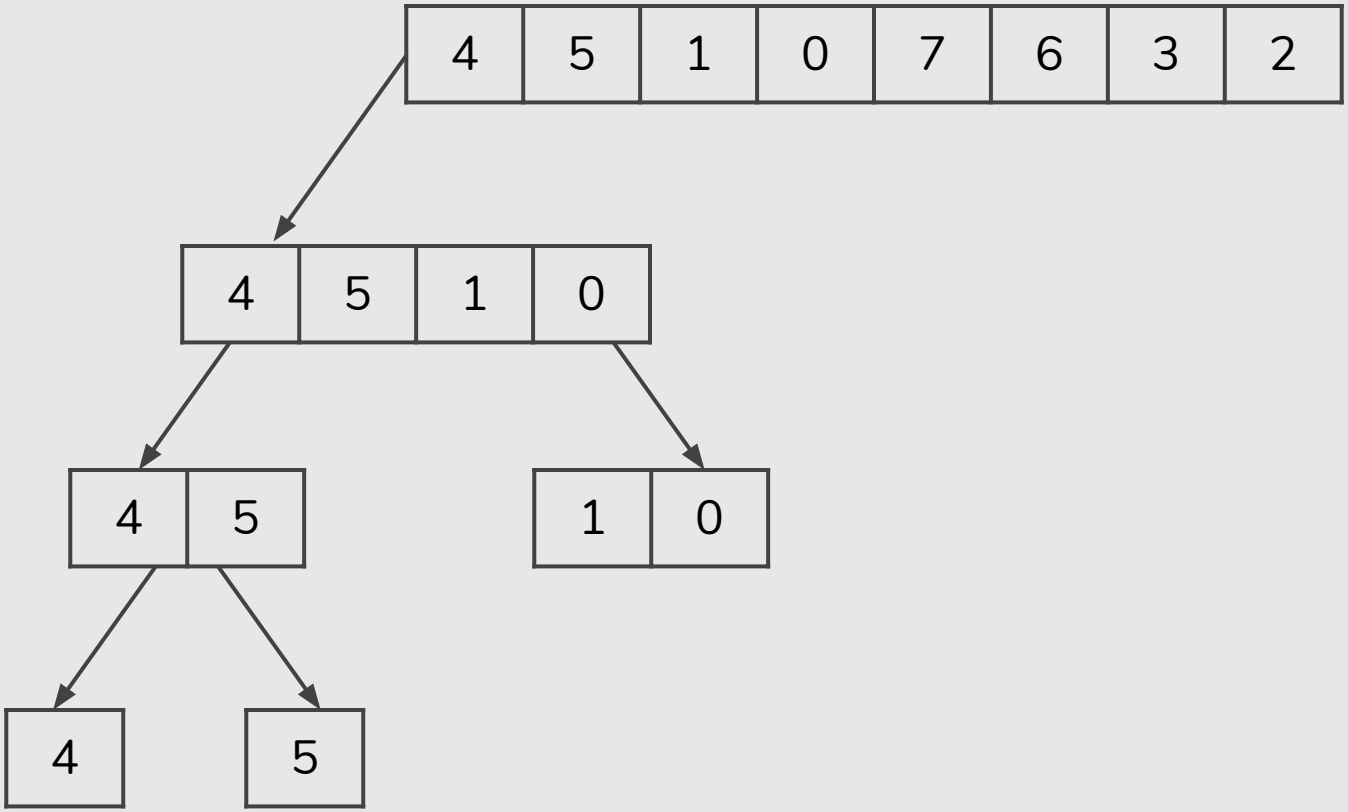
Merge Sort



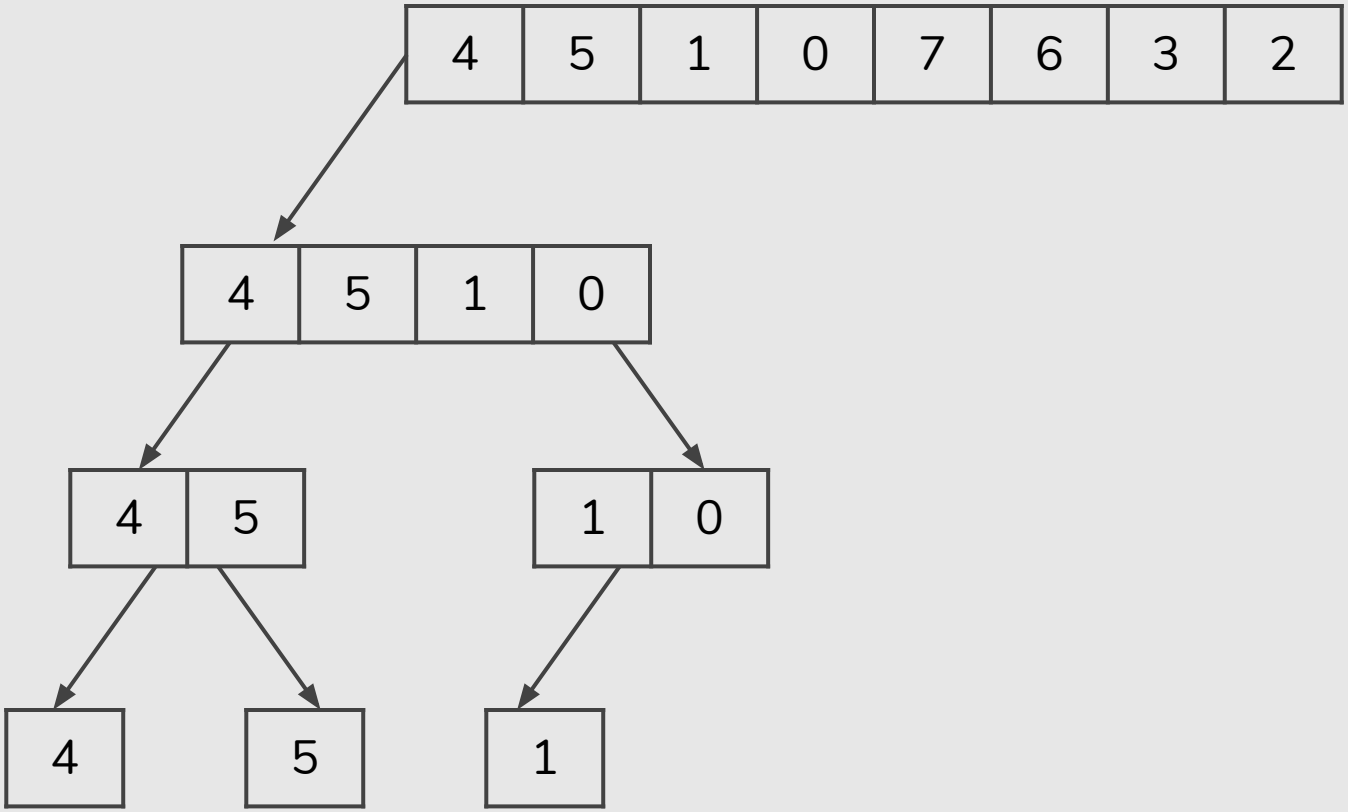
Merge Sort



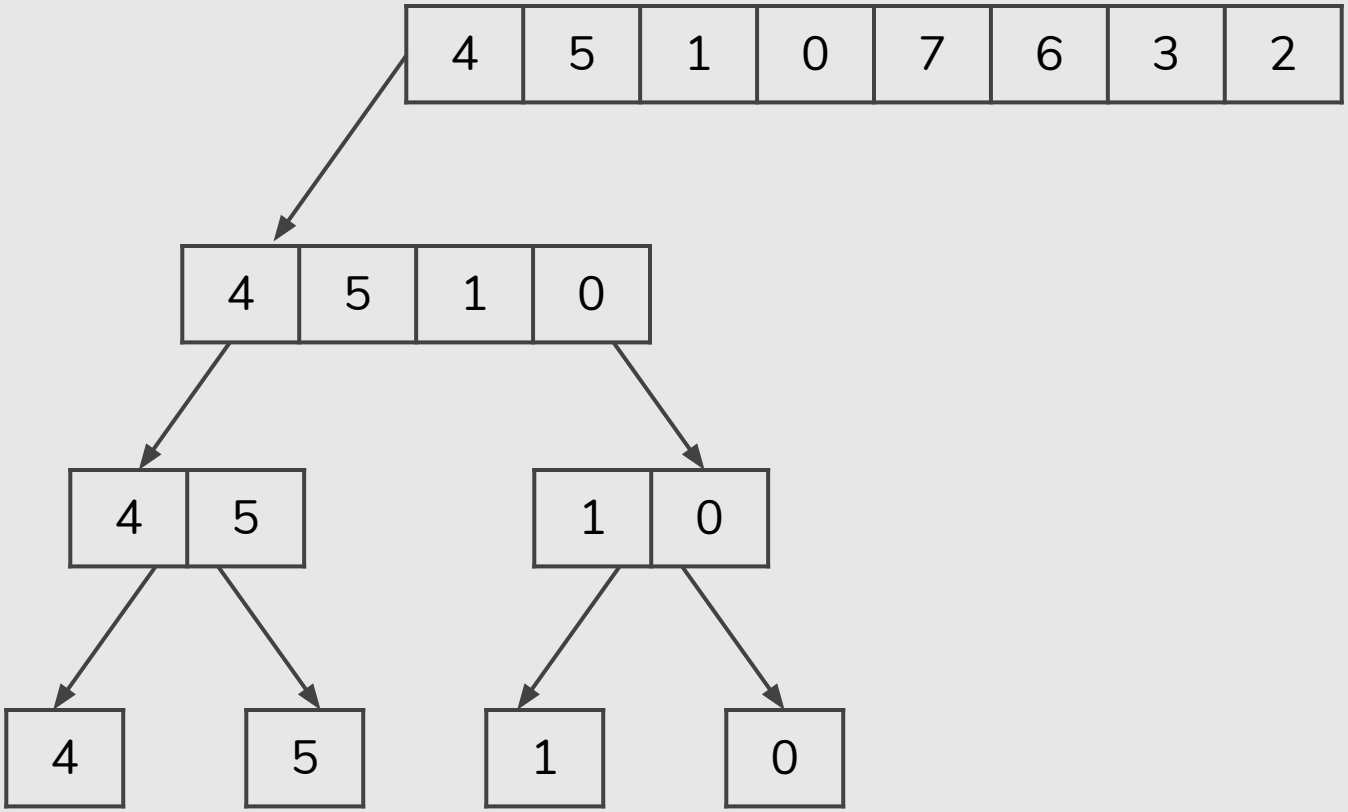
Merge Sort



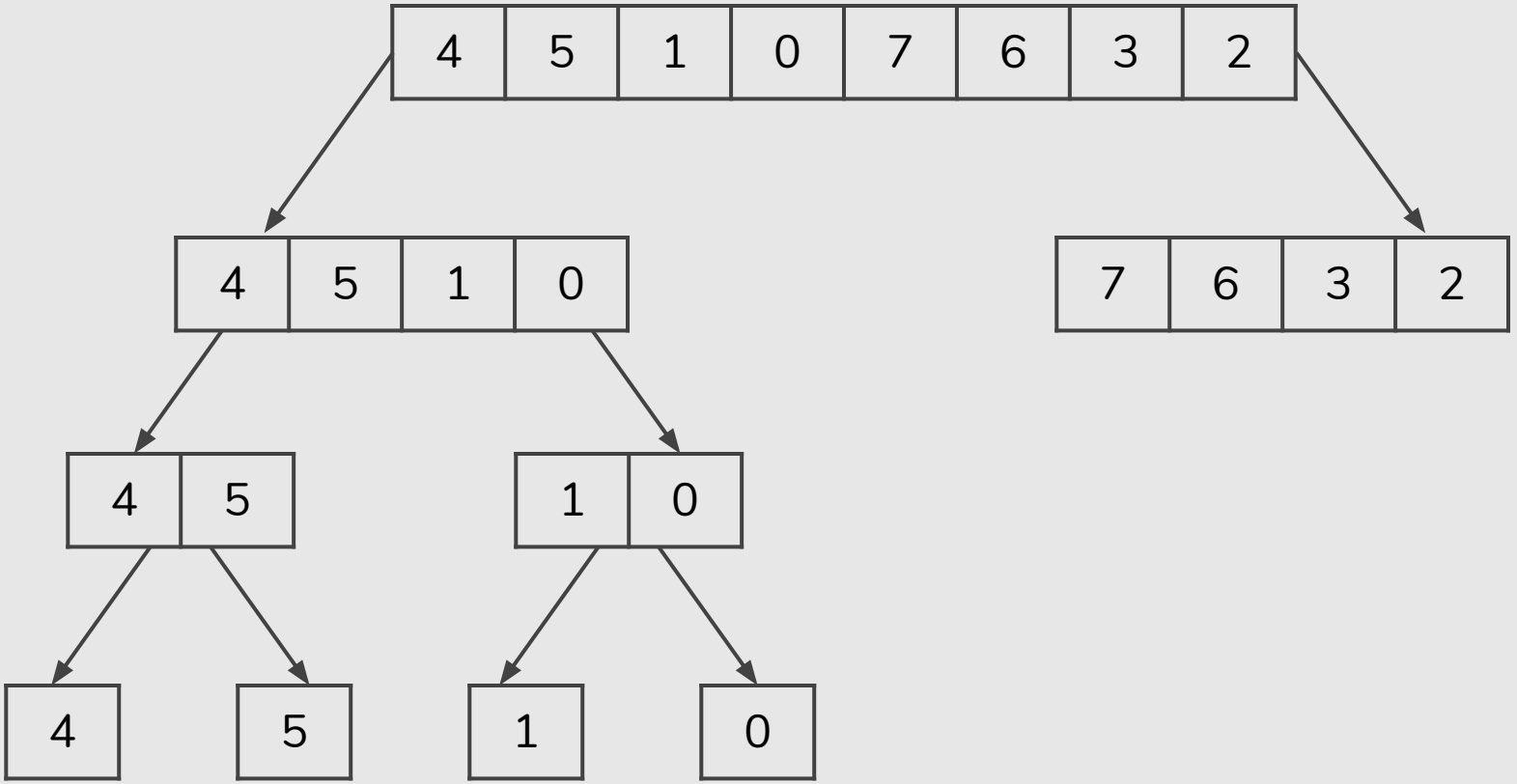
Merge Sort



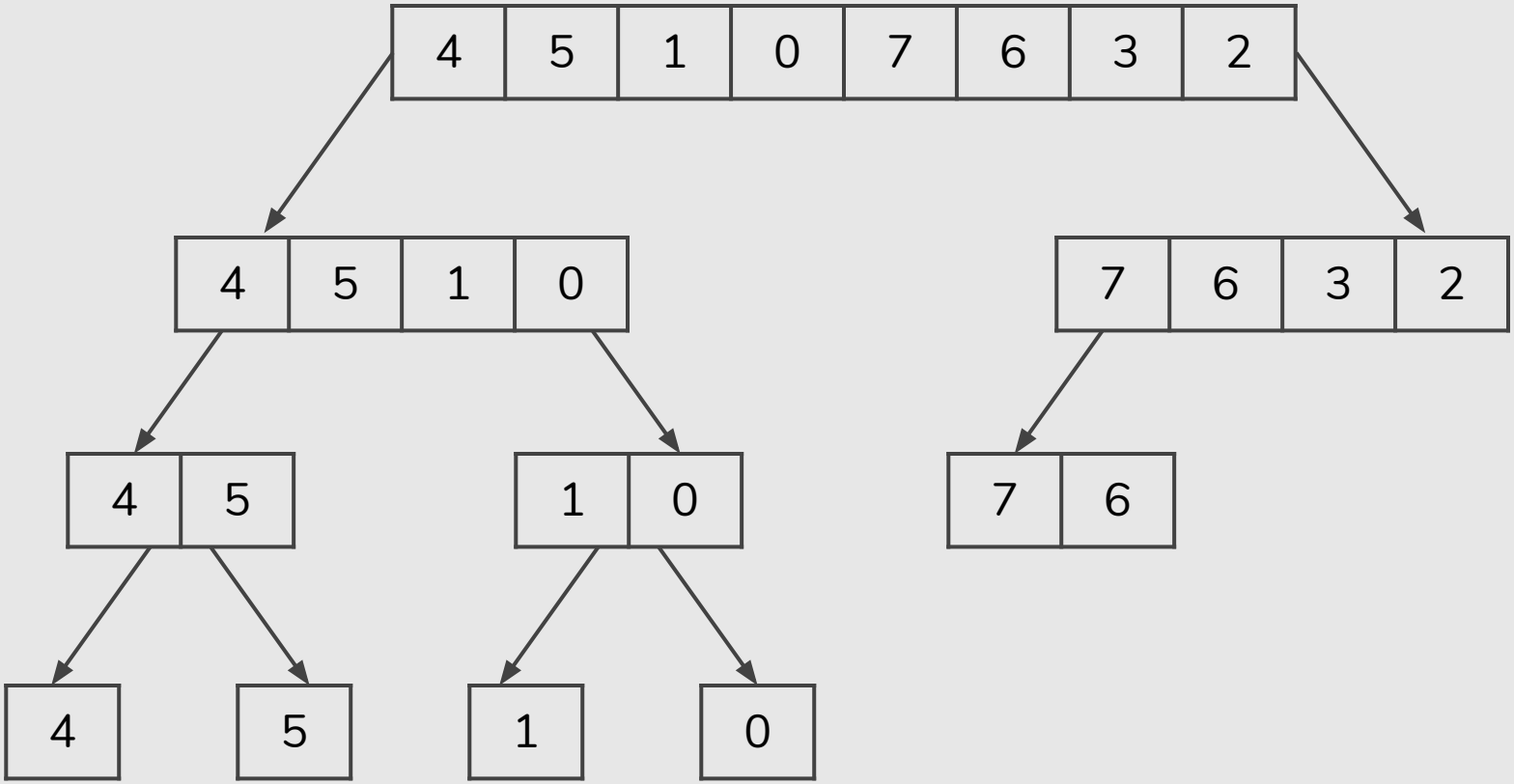
Merge Sort



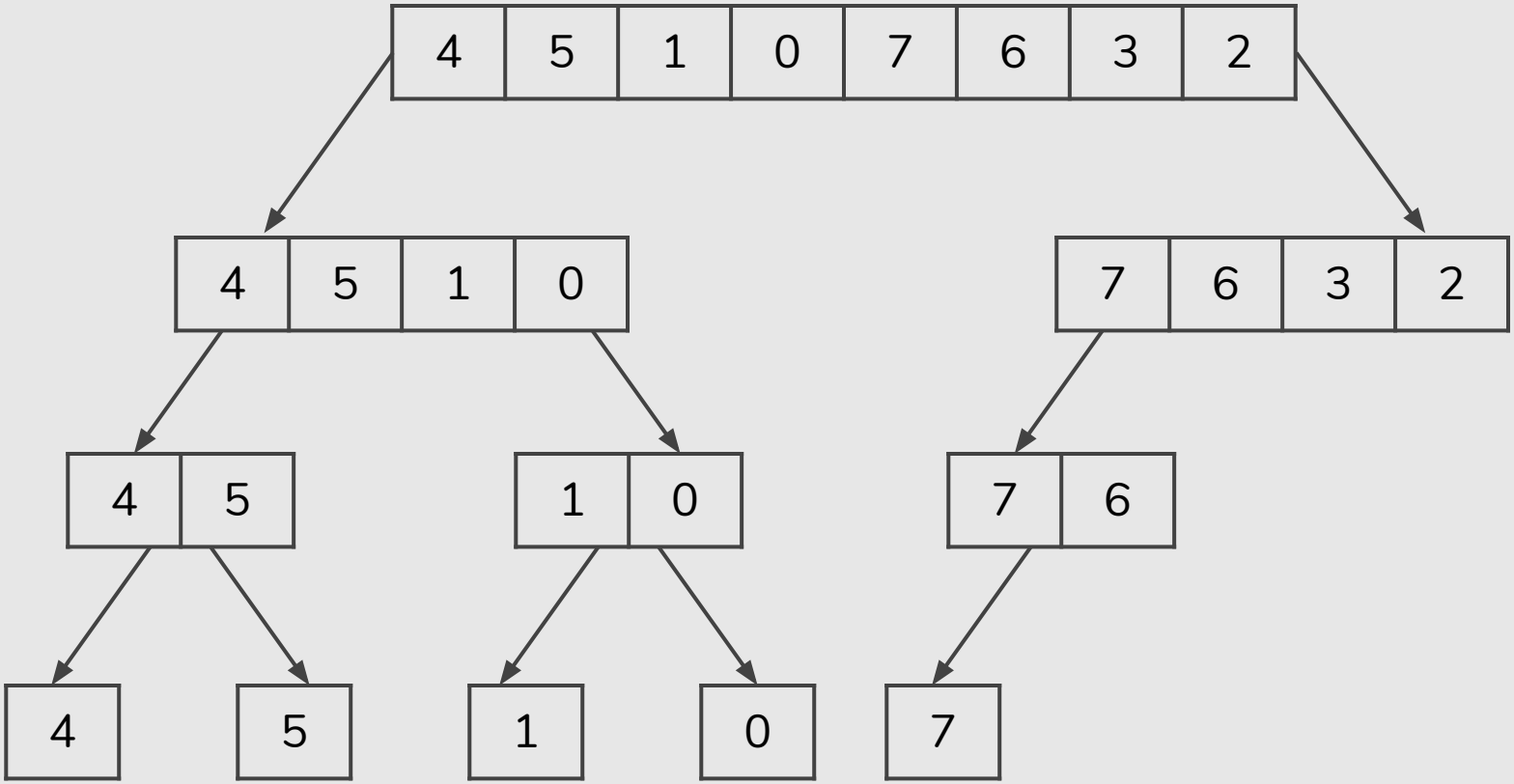
Merge Sort



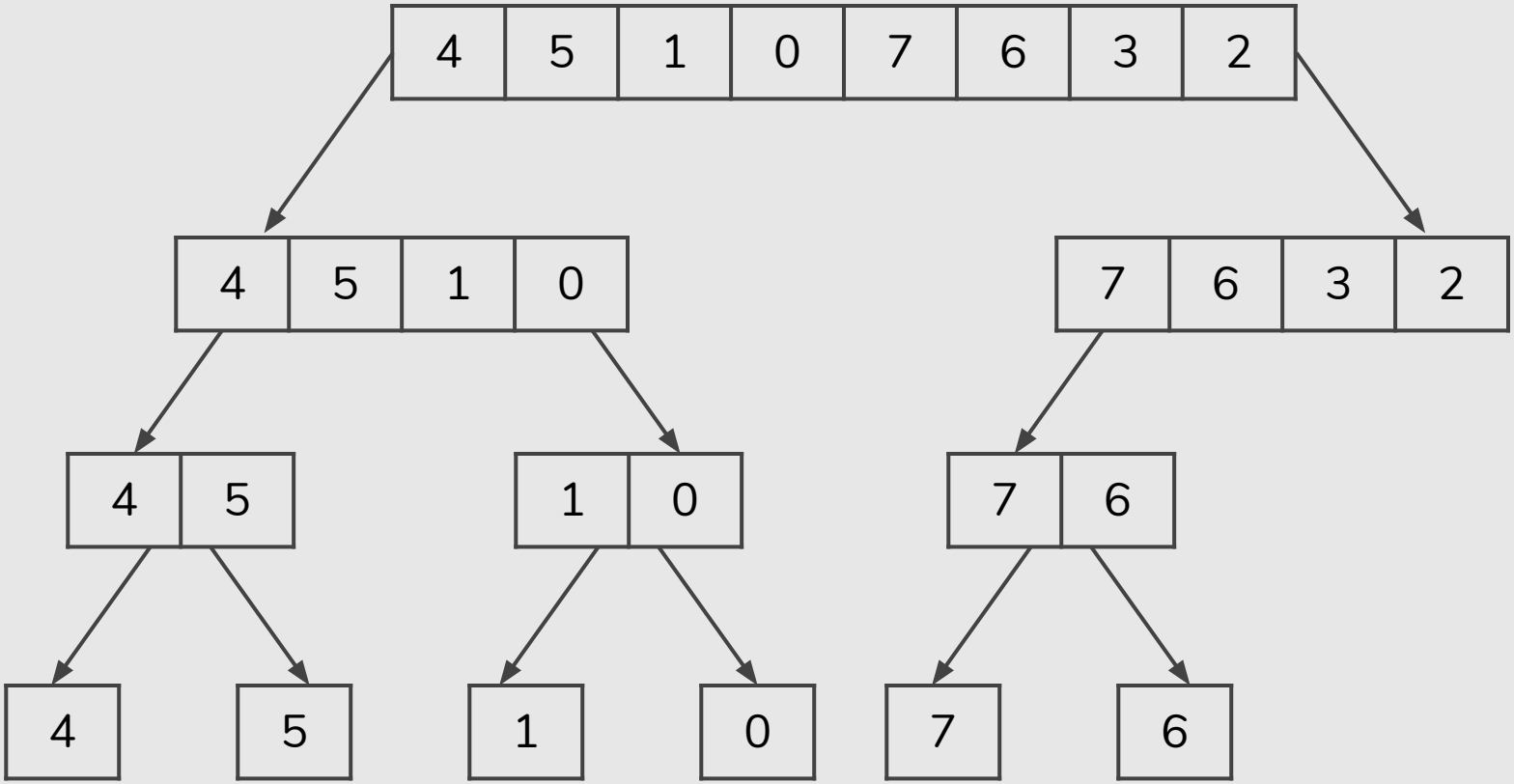
Merge Sort



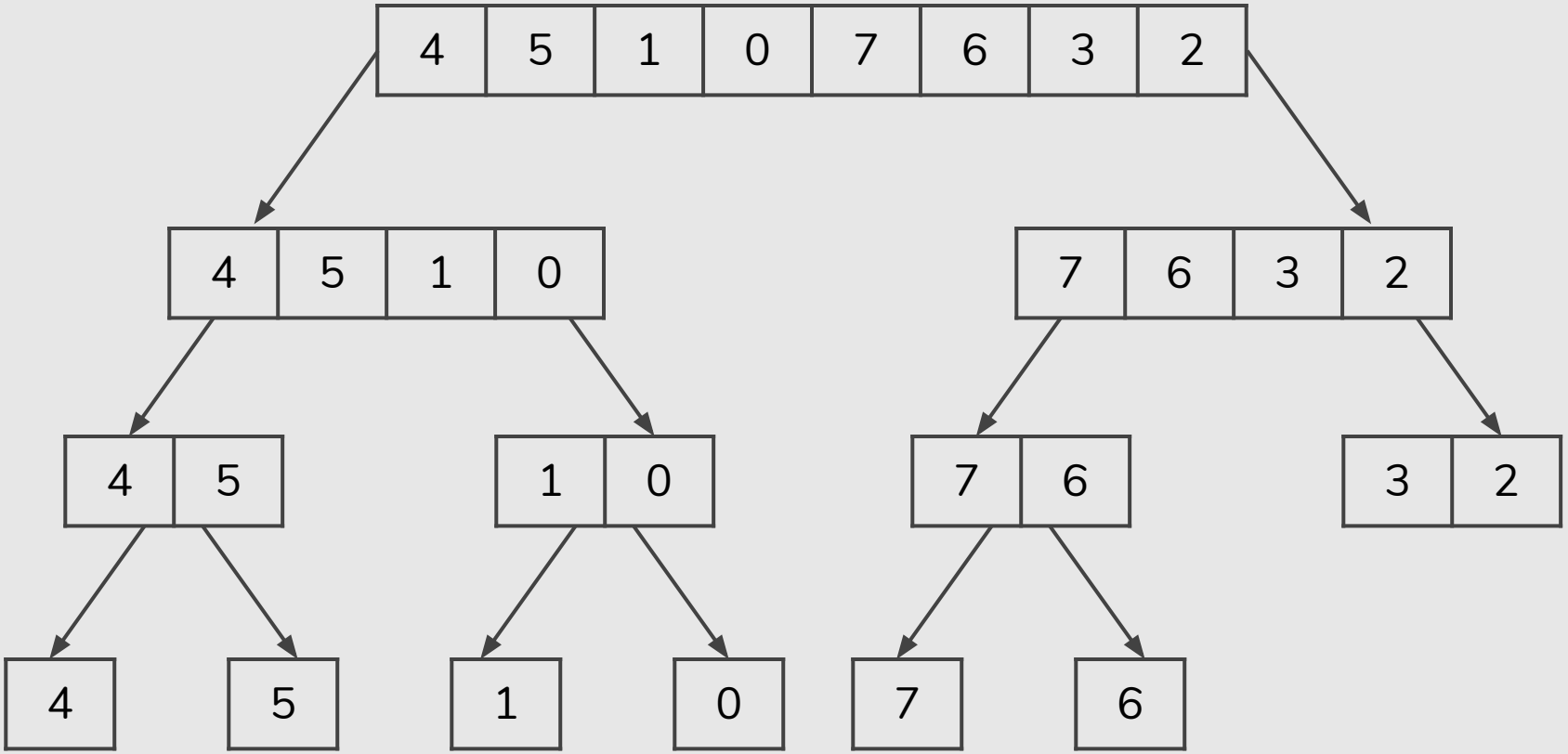
Merge Sort



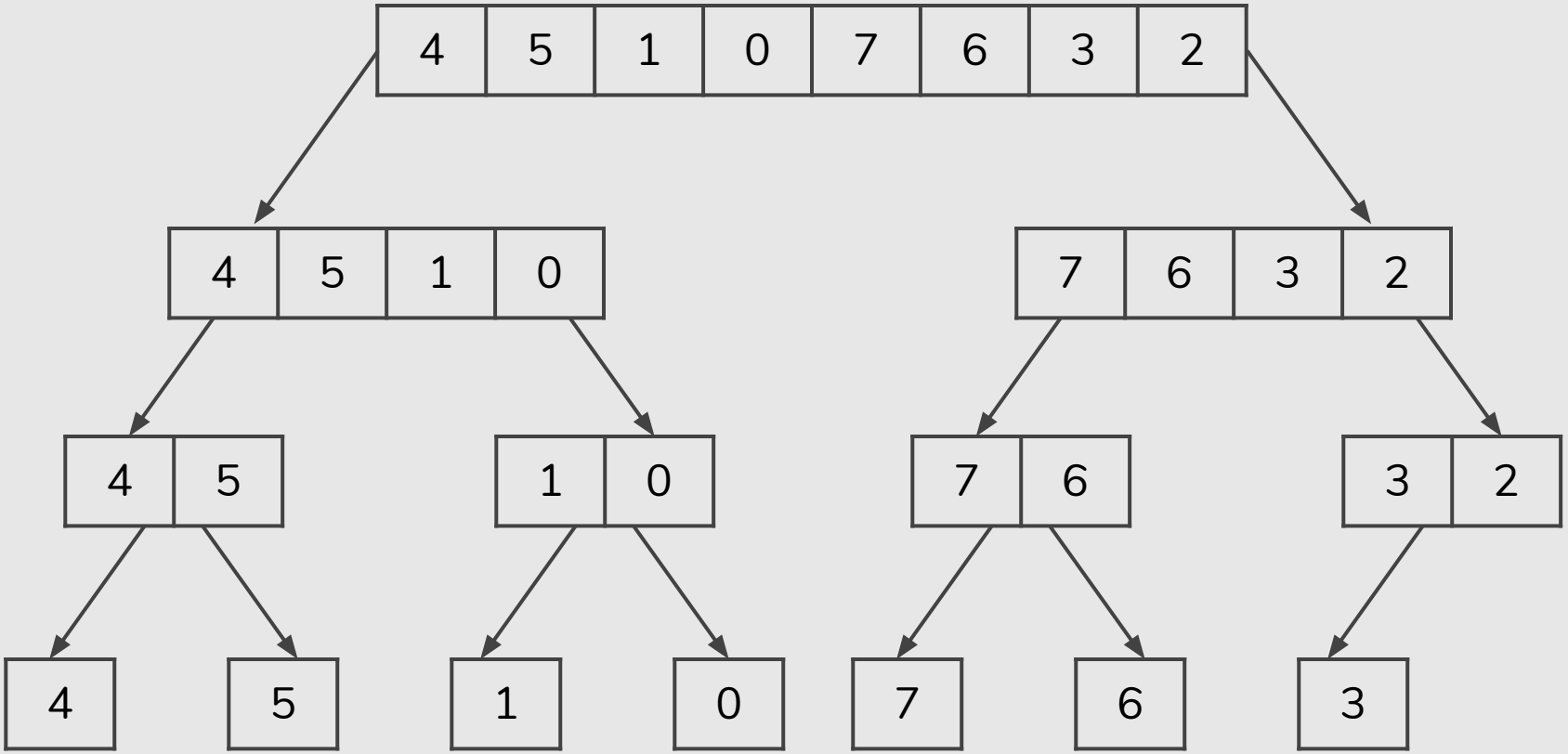
Merge Sort



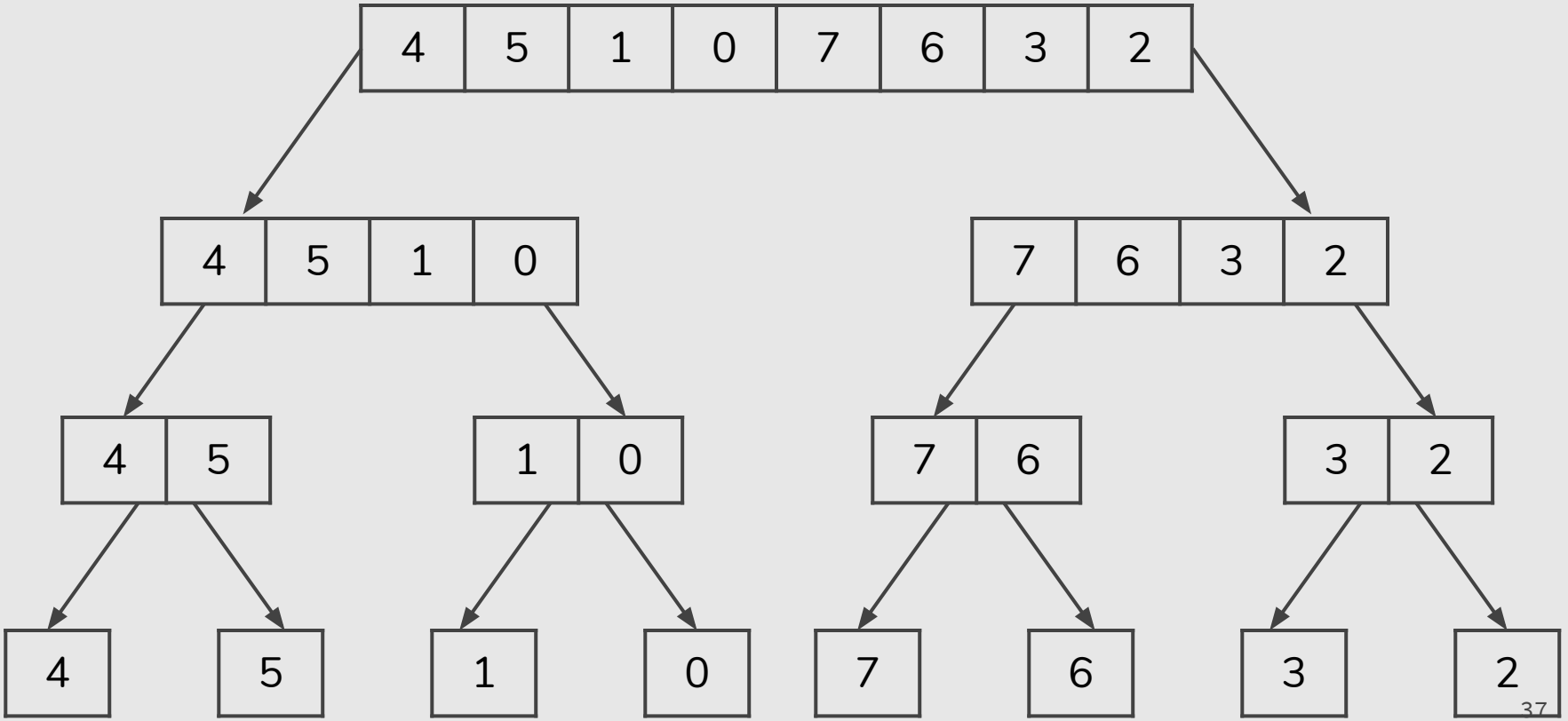
Merge Sort



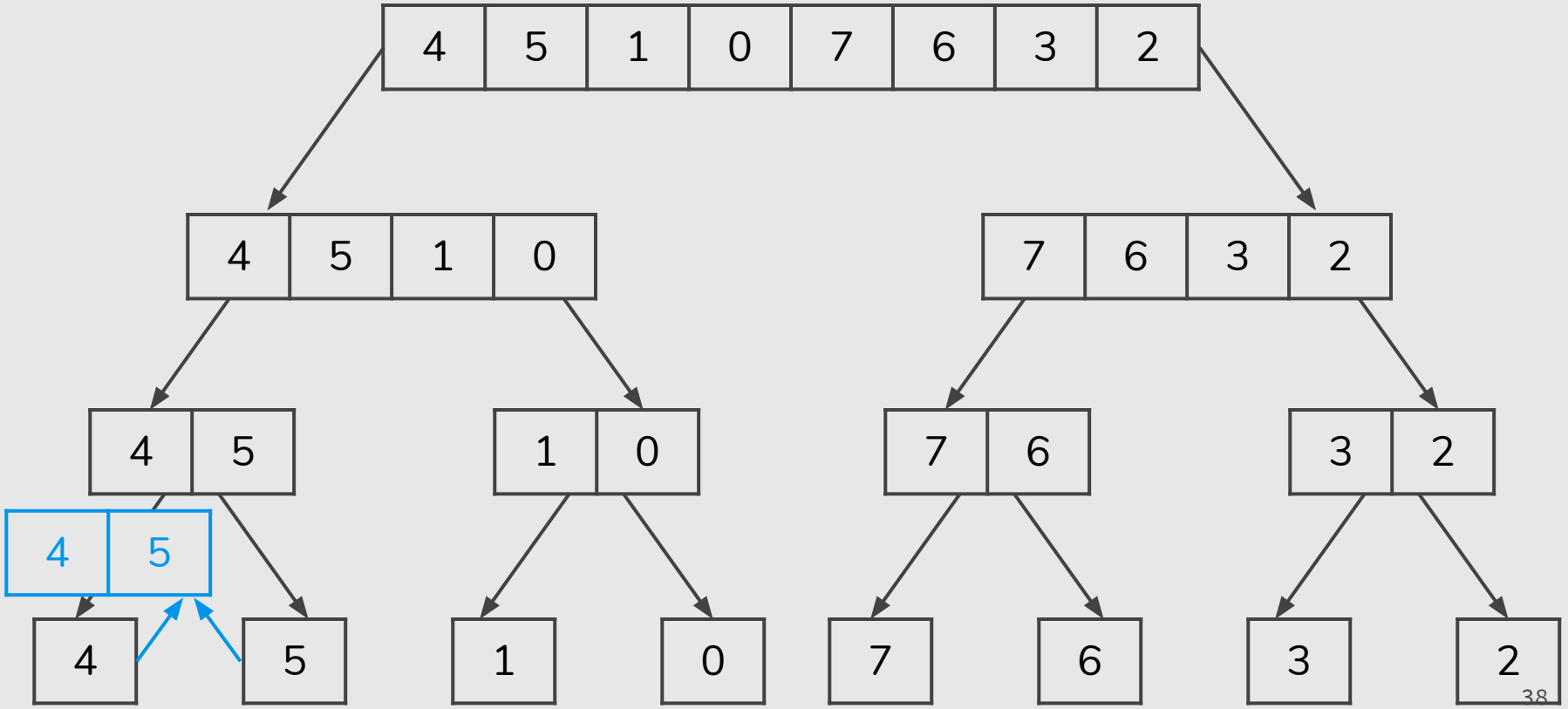
Merge Sort



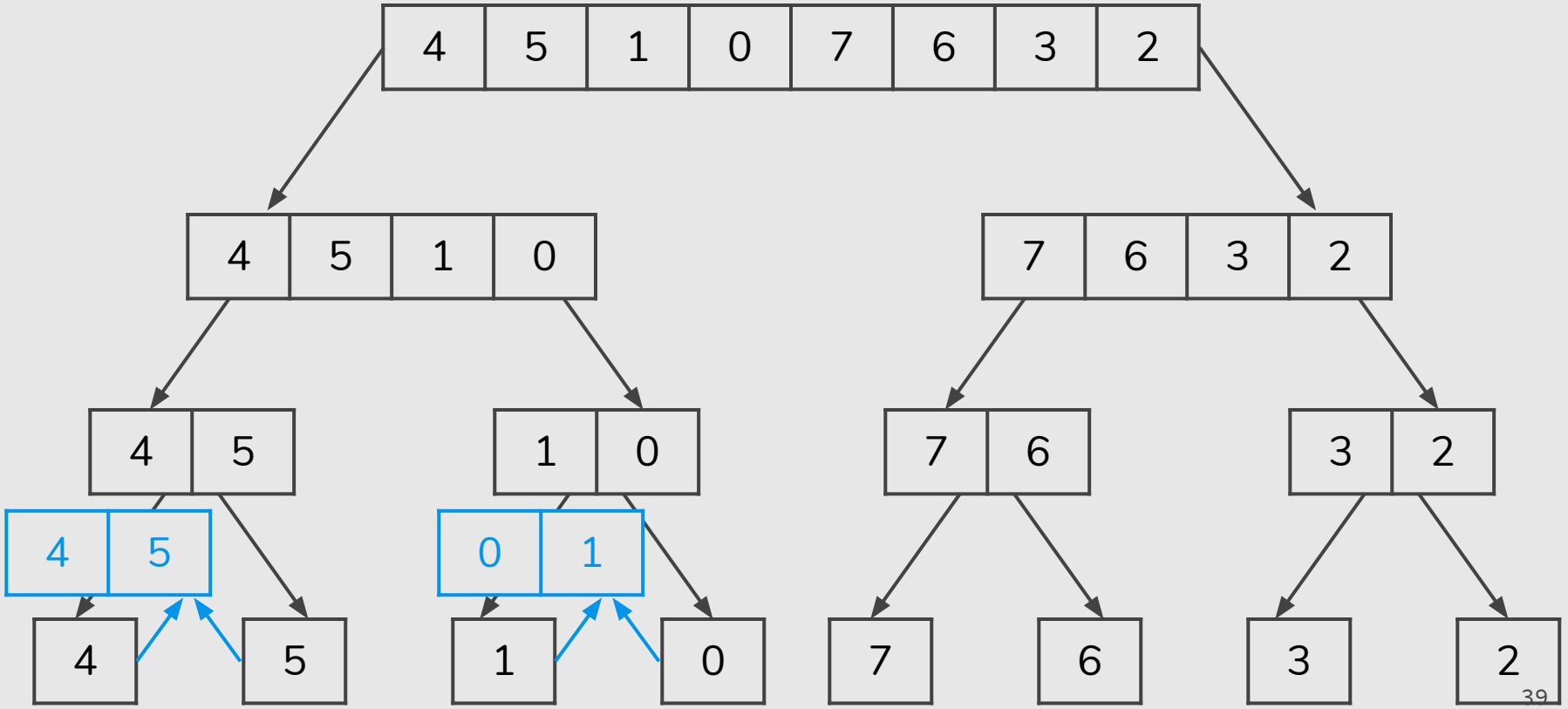
Merge Sort



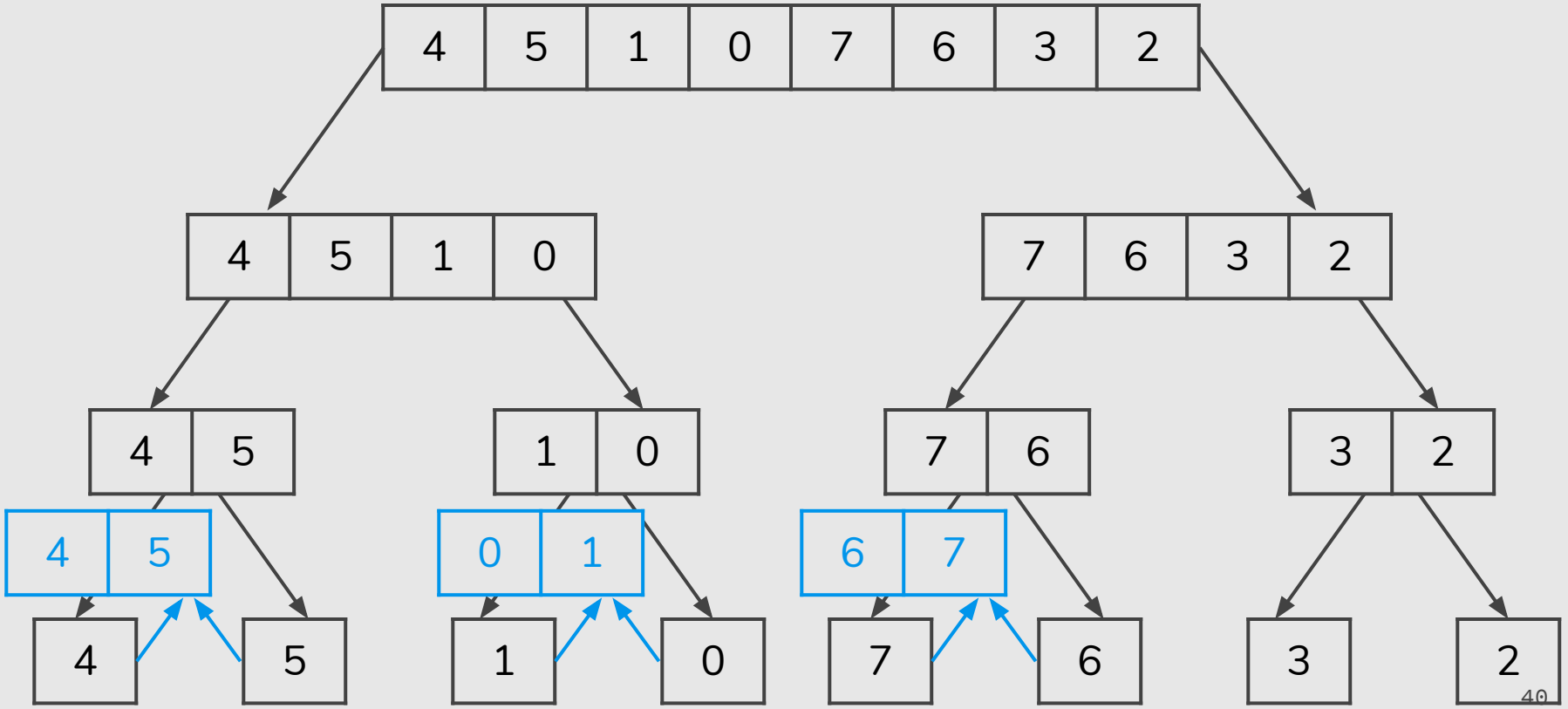
Merge Sort



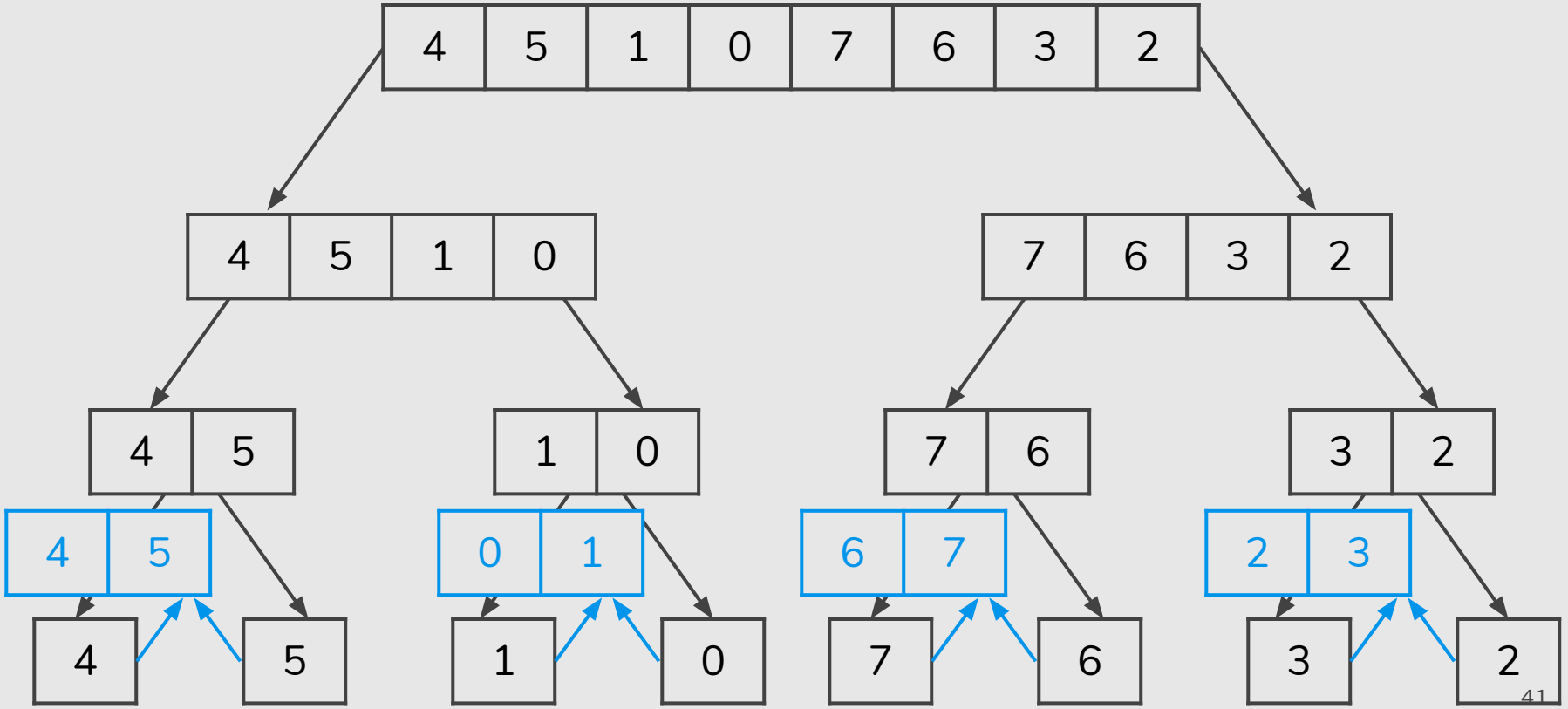
Merge Sort



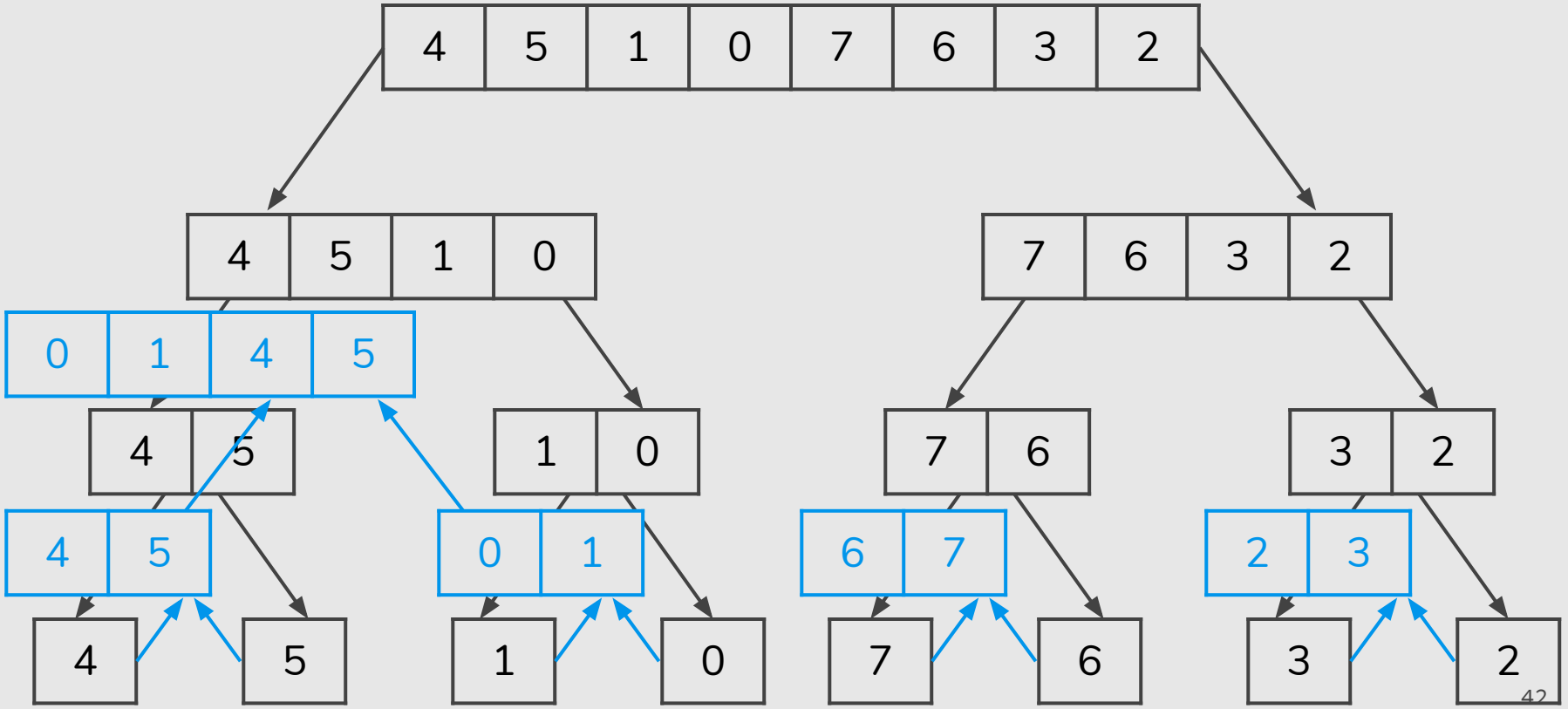
Merge Sort



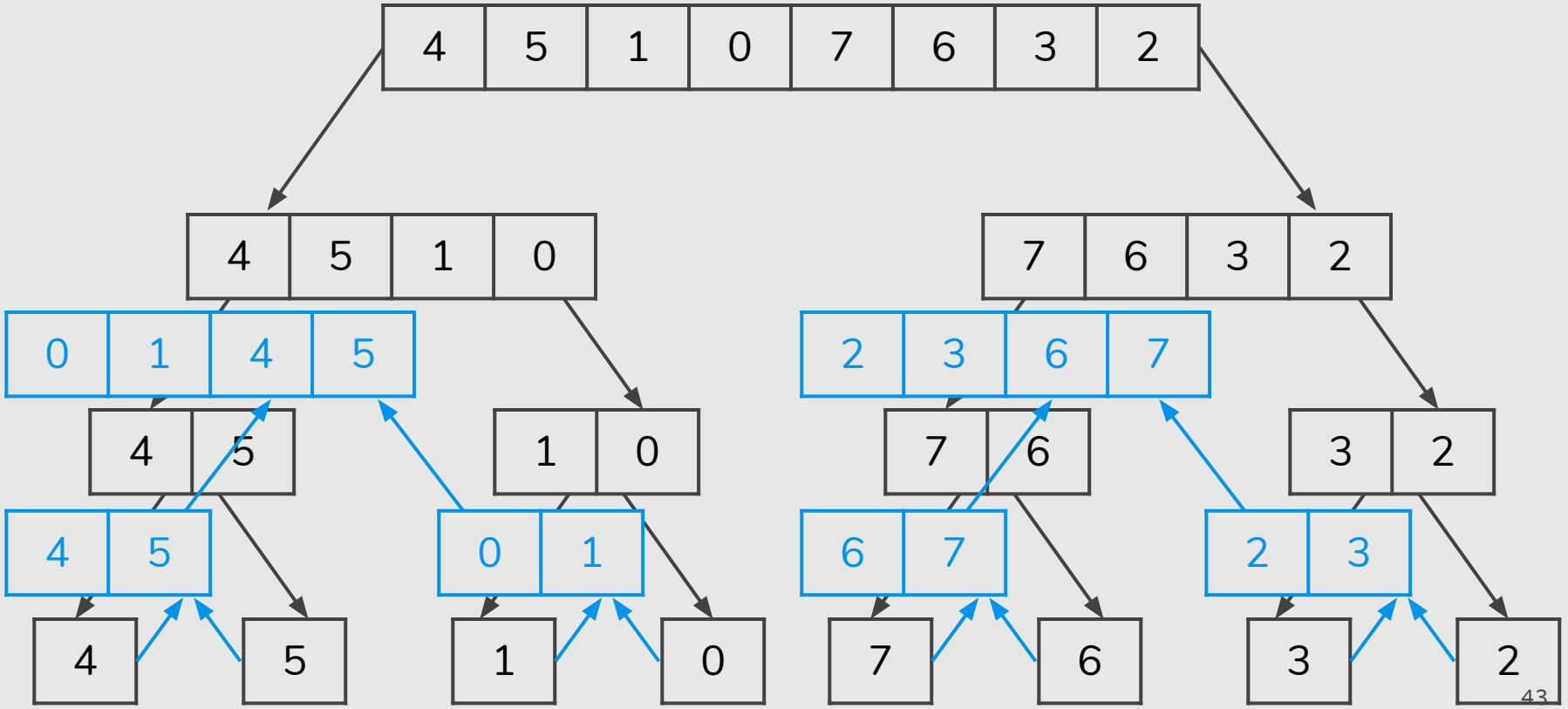
Merge Sort



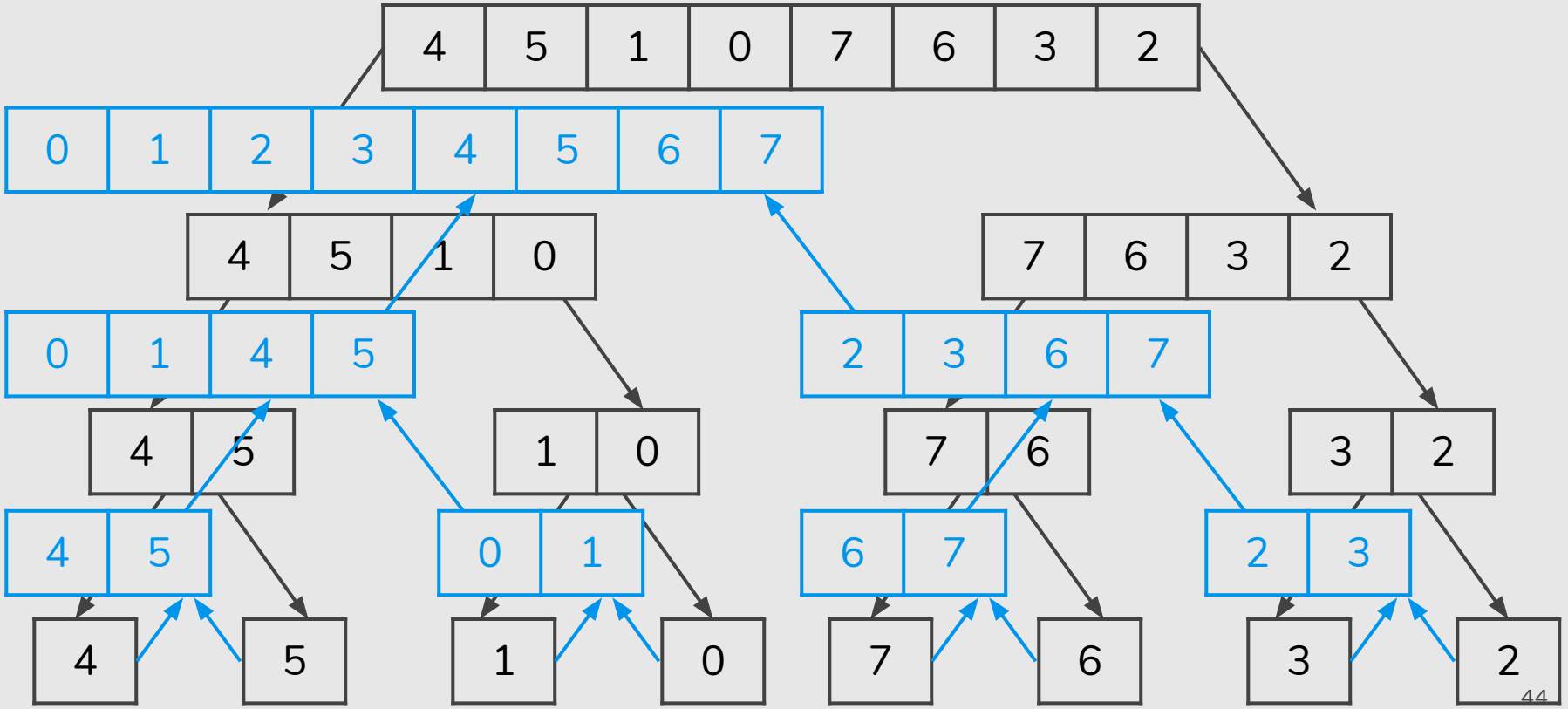
Merge Sort



Merge Sort



Merge Sort


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

Merge Sort

- Note que só criamos 2 listas, `v` a ser ordenada e `aux` do mesmo tamanho de `v`.
- Somente estas duas listas existirão durante todas as chamadas recursivas.

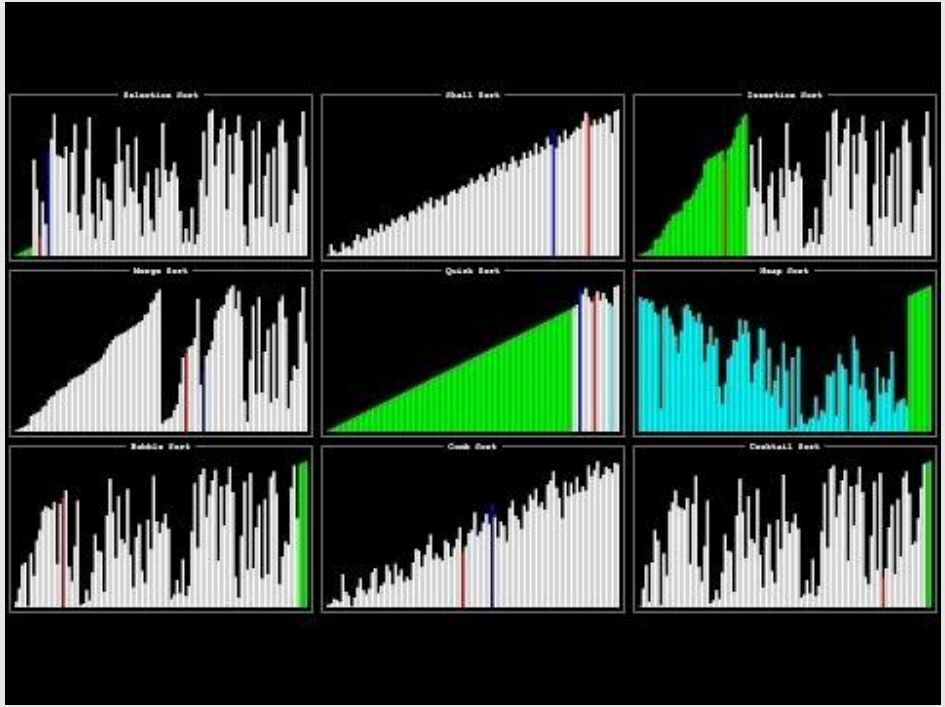
```
v = [12, 90, 47, -9, 78, 45, 78, 3323, 1, 2, 34, 20]
aux = [0 for i in range(12)] # tem o mesmo tamanho de v
print(v)
mergeSort(v, 0, 11, aux)
print(v)
```

Exercícios

1. Mostre passo a passo a execução da função `merge` considerando dois sub-vetores: `(3,5,7,10,11,12)` e `(4,6,8,9,11,13,14)`.
2. Faça uma execução passo-a-passo do `mergeSort` para o vetor: `(30,45,21,20,6,715,100,65,33)`.
3. Reescreva o algoritmo `mergeSort` para que este passe a ordenar um vetor em ordem decrescente.
4. Temos como entrada um vetor de inteiros `v` (não necessariamente ordenado), e um inteiro `x`. Desenvolva um algoritmo que determina se há dois números em `v` cuja soma seja `x`. Tente fazer o algoritmo o mais eficiente possível. Utilize um dos algoritmos de ordenação na sua solução.

Visualization and Comparison of Sorting Algorithms

https://www.youtube.com/watch?v=ZZuD6iUe3Pc



Questão 2:
(Prova 2018/1 Tipo 1)

Questão 2

(2.0 pontos) Carla estava se preparando para estudar ... anotações de aula para um algoritmo:

1. Definir uma lista com n elementos
2. Definir uma variável i como 2
3. Para cada elemento da posição 0 até a posição n-i
    - Se o elemento subsequente ao elemento atual for maior que o elemento atual, troque esses elementos
4. Escrever na tabela de iterações o conteúdo da lista

Nota: Os passos 3 e 4 devem ser executados no total n-1 vezes. Após cada execução, lembrar de incrementar o valor de i em 1.

Questão 2 (a)

(0.5 ponto) Qual o algoritmo de ordenação a que se referem as anotações de Carla?

Ordenação por bolha ou Bubble sort.

Questão 2 (b)

(1.5 ponto) Considere a seguinte lista de números inteiros:

3	7	9	2	5

Preencha a tabela com os resultados de cada iteração do algoritmo de ordenação que estudamos nesta aula.

Lista Original	3	7	9	2	5
Iteração 1	3	7	2	5	9
Iteração 2	3	2	5	7	9
Iteração 3	2	3	5	7	9

Questão 2
(Prova 2018/1 Tipo 2)

Questão 2

(2.0 pontos) Carla estava se preparando para estudar ... anotações de aula para um algoritmo:

1. Definir uma lista com n elementos
2. Divida a lista em duas partes: uma ordenada (com o primeiro elemento da lista) e uma não ordenada (com os demais elementos)
3. Selecione e remova o primeiro elemento da parte não ordenada (deixe um buraco nessa posição da lista)
4. Desloque para a direita os elementos da parte ordenada (do último ao primeiro) até encontrar a posição onde encaixar o elemento selecionado no item 3
5. Atualize a divisão da lista: agora a parte ordenada contém um elemento a mais
6. Repita os itens 3, 4, e 5 até que a parte ordenada contenha n elementos

Questão 2 (a)

(0.5 ponto) Qual o algoritmo de ordenação a que se referem as anotações de Carla?

Ordenação por inserção ou Insertion sort.

Questão 2 (b)

(1.5 ponto) Considere a seguinte lista de números inteiros:

3	7	9	2	5

Preencha a tabela com os resultados de cada iteração do algoritmo de ordenação que estudamos nesta aula.

Lista Original	3	7	9	2	5
Iteração 1	3	7	9	2	5
Iteração 2	3	7	9	2	5
Iteração 3	2	3	7	9	5
Iteração 4	2	3	5	7	9

Questão 2
(Prova 2018/1 Tipo 3)

Questão 2

(2.0 pontos) Carla estava se preparando para estudar ... anotações de aula para um algoritmo:

1. Definir a lista
2. Para cada elemento da lista (percorrer a lista do primeiro ao penúltimo elemento)
    - Descubra a posição do menor elemento a partir do elemento atual
    - Se o elemento encontrado for menor que o elemento atual, troque esses valores
    - Escreva na tabela de iterações o conteúdo atual da lista

Questão 2 (a)

(0.5 ponto) Qual o algoritmo de ordenação a que se referem as anotações de Carla?

Ordenação por seleção ou Selection sort.

Questão 2 (b)

(1.5 ponto) Considere a seguinte lista de números inteiros:

3	7	9	2	5

Preencha a tabela com os resultados de cada iteração do algoritmo de ordenação que estudamos nesta aula.

Lista Original	3	7	9	2	5
Iteração 1	2	7	9	3	5
Iteração 2	2	3	9	7	5
Iteração 3	2	3	5	7	9
Iteração 4	2	3	5	7	9