Algoritmos e Programação de Computadores

Ordenação: Quick Sort & Merge Sort

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

MC102, 13 Junho, 2018

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

- 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.

Quick Sort: Particionamento

Dado um valor `p` como `pivô`, como fazer o particionamento?

- Podemos “varrer” a lista do início para o fim até encontrarmos um elemento maior que o `pivô`.
- “Varremos” o vetor do fim para o início até encontrarmos um elemento menor ou igual ao `pivô`.
- Trocamos então estes elementos de posições e continuamos com o processo até termos verificado todas as posições do vetor.

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: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,9,3,7,6,2,3,8,5)`

inicio = 0
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,9,3,7,6,2,3,8,5)

inicio = 0
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,**9**,3,7,6,2,3,8,5)

inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,**9**,3,7,6,2,3,8,**5**)

inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,**9**,3,7,6,2,3,8,**5**) → (1,**5**,3,7,6,2,3,8,**9**)

inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,**5**,3,7,6,2,3,8,**9**)

inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,**5**,3,7,6,2,3,8,**9**)

inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,**3**,7,6,2,3,8,**9**)

inicio = 2
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,**7**,6,2,3,8,**9**)

inicio = 3
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,`**7**`,6,2,3,8,`**9**`)`

inicio = 3
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,**7**,6,2,3,**8**,9)

inicio = 3
fim = 7
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,**7**,6,2,**3**,8,9)

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,**7**,6,2,**3**,8,9) → (1,5,3,**3**,6,2,**7**,8,9)

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,**3**,6,2,**7**,8,9)

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,**3**,6,2,**7**,8,9)

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,3,**6**,2,**7**,8,9)

inicio = 4
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,3,**6**,2,**7**,8,9)

inicio = 4
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,3,**6**,**2**,7,8,9)

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
     pivo = v[fim]
     while (inicio < fim):
          while (inicio < fim) and (v[inicio] <= pivo):
               inicio = inicio + 1
          while (inicio < fim) and (v[fim] > pivo) :
               fim = fim - 1
          v[inicio], v[fim] = v[fim], v[inicio]
     return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,3,**6**,**2**,7,8,9) → (1,5,3,3,**2**,**6**,7,8,9)

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
     pivo = v[fim]
     while (inicio < fim):
          while (inicio < fim) and (v[inicio] <= pivo):
               inicio = inicio + 1
          while (inicio < fim) and (v[fim] > pivo) :
               fim = fim - 1
          v[inicio], v[fim] = v[fim], v[inicio]
     return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,3,**2**,**6**,7,8,9)

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
     pivo = v[fim]
     while (inicio < fim):
          while (inicio < fim) and (v[inicio] <= pivo):
               inicio = inicio + 1
          while (inicio < fim) and (v[fim] > pivo) :
               fim = fim - 1
          v[inicio], v[fim] = v[fim], v[inicio]
     return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- (1,5,3,3,**2**,**6**,7,8,9)

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,3,2,**6**,7,8,9)

inicio = 5
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: (1,9,3,7,6,2,3,8,5) e pivô=5.

- (1,5,3,3,2,**6**,7,8,9) → Retorna posição 5 (inicio=5).

inicio = 5
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <= pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,`**`9`**`,3,7,6,2,3,8,`**`5`**`)` → `(1,`**`5`**`,3,7,6,2,3,8,`**`9`**`)`
- `(1,5,3,`**`7`**`,6,2,`**`3`**`,8,9)` → `(1,5,3,`**`3`**`,6,2,`**`7`**`,8,9)`
- `(1,5,3,3,`**`6`**`,`**`2`**`,7,8,9)` → `(1,5,3,3,`**`2`**`,`**`6`**`,7,8,9)`
- `(1,5,3,3,2,`**`6`**`,7,8,9)` → Retorna posição `5`.

Quick Sort

```
def quickSort(v, inicio, fim):
    if (inicio < fim):
    # tem pelo menos 2 elementos a serem ordenados
        pos = particiona(v, inicio, fim)
        quickSort(v, inicio, pos-1)
        quickSort(v, pos, fim)
```

Quick Sort

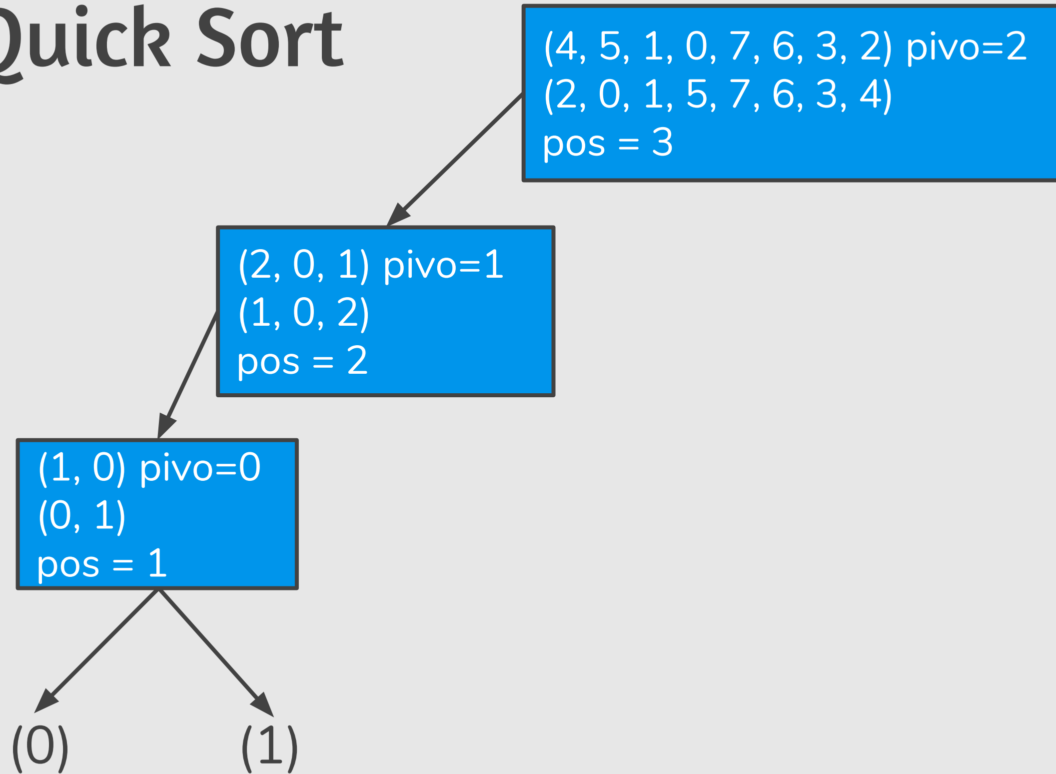
(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3

Quick Sort

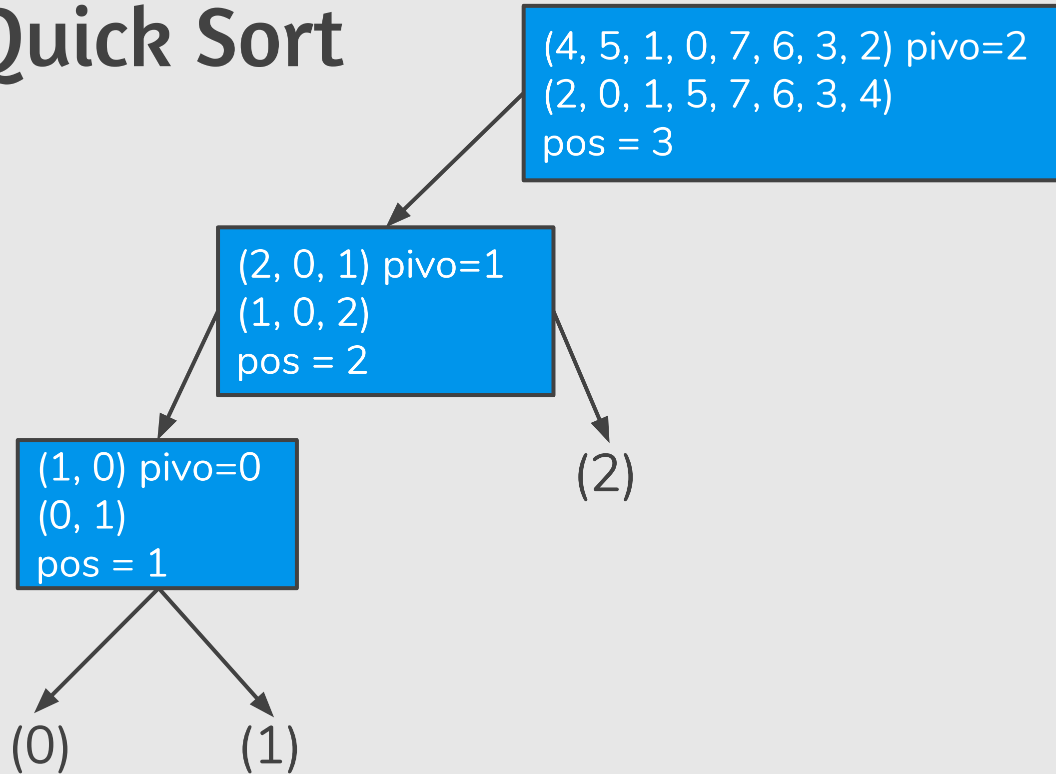
(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3

(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2

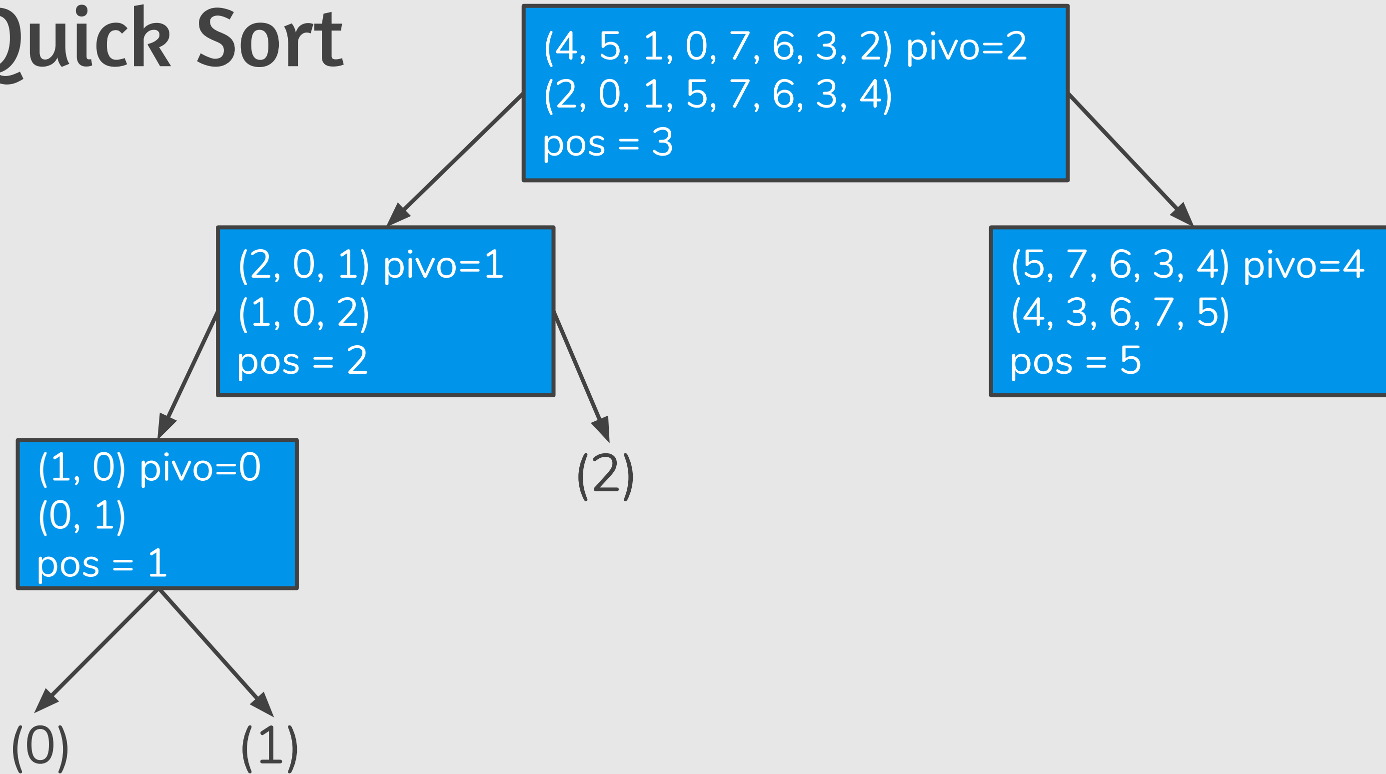
Quick Sort



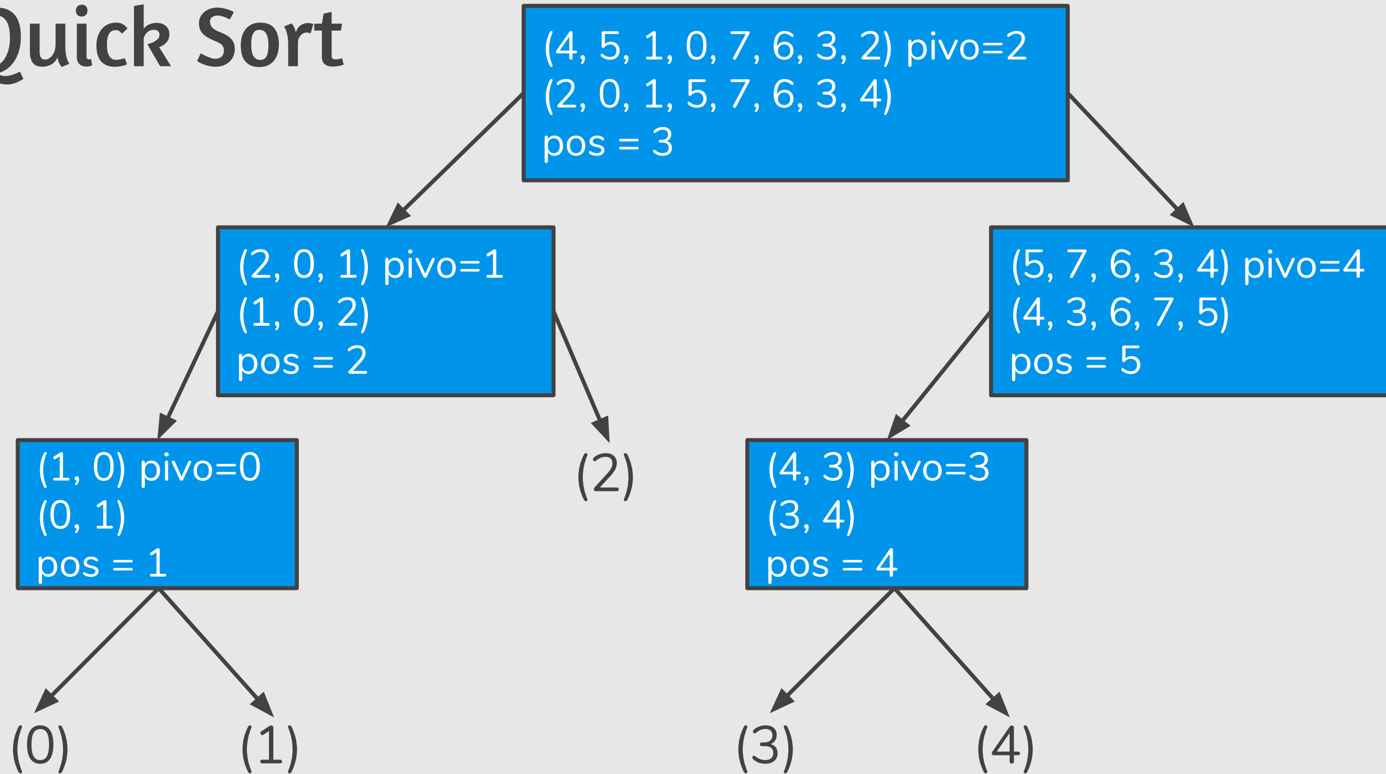
Quick Sort



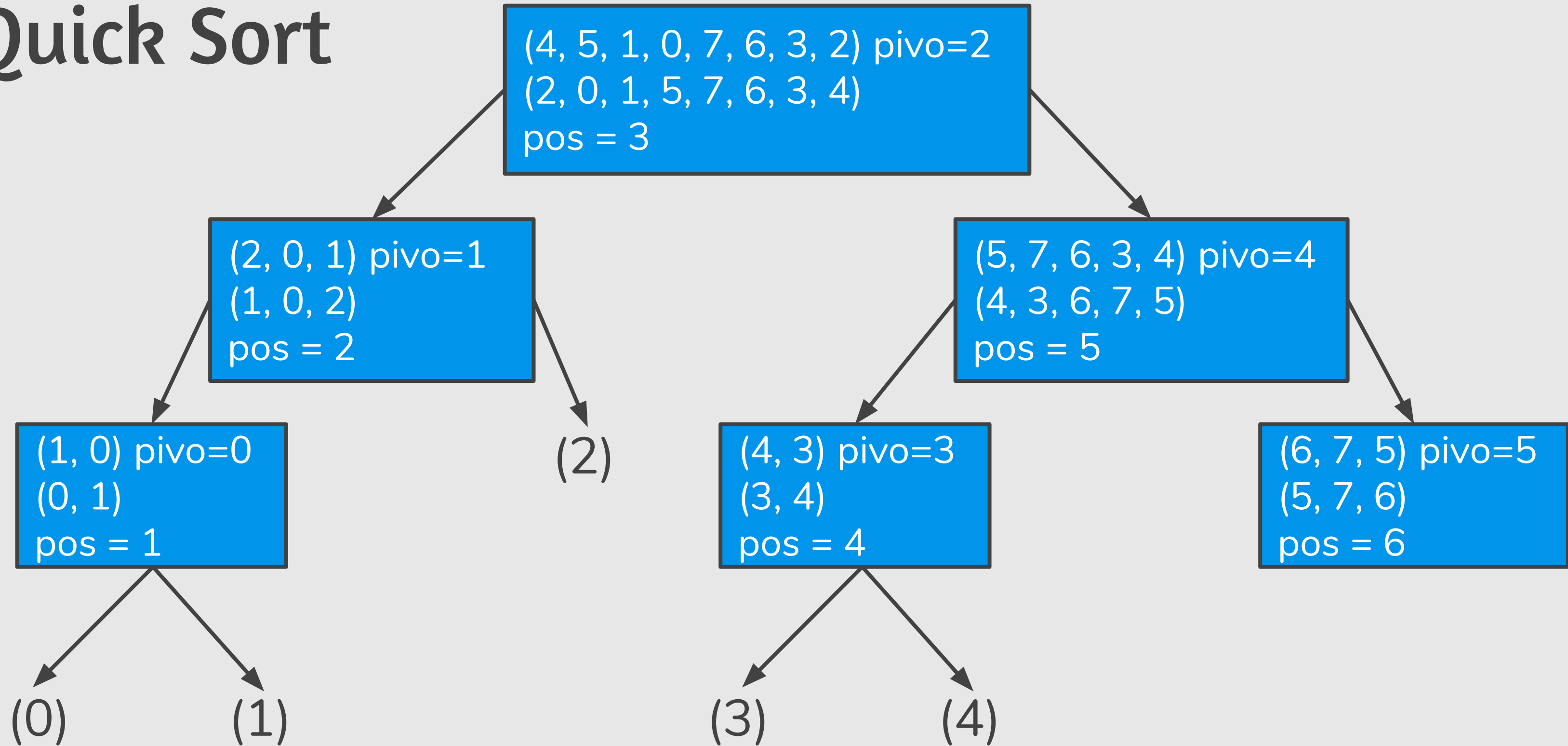
Quick Sort



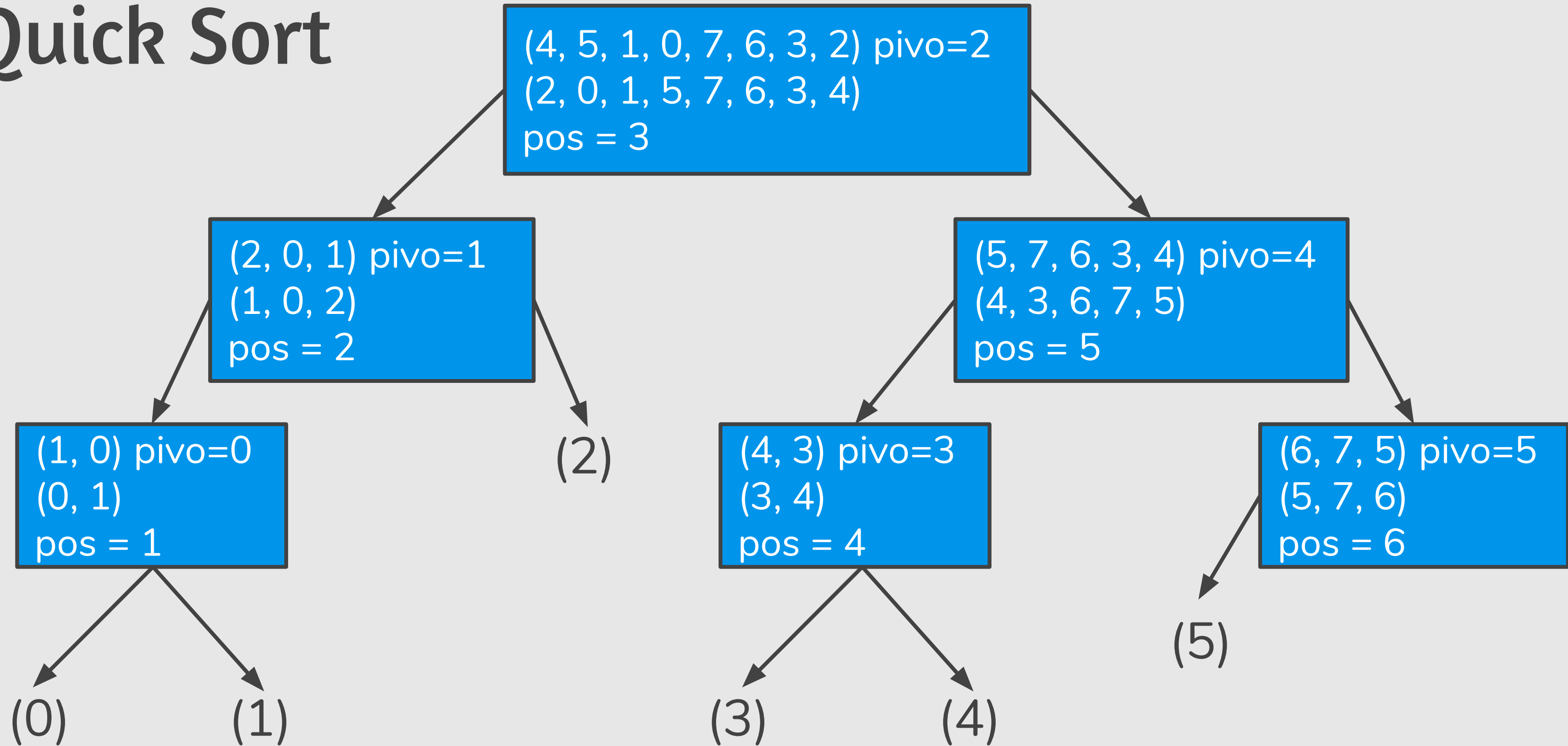
Quick Sort



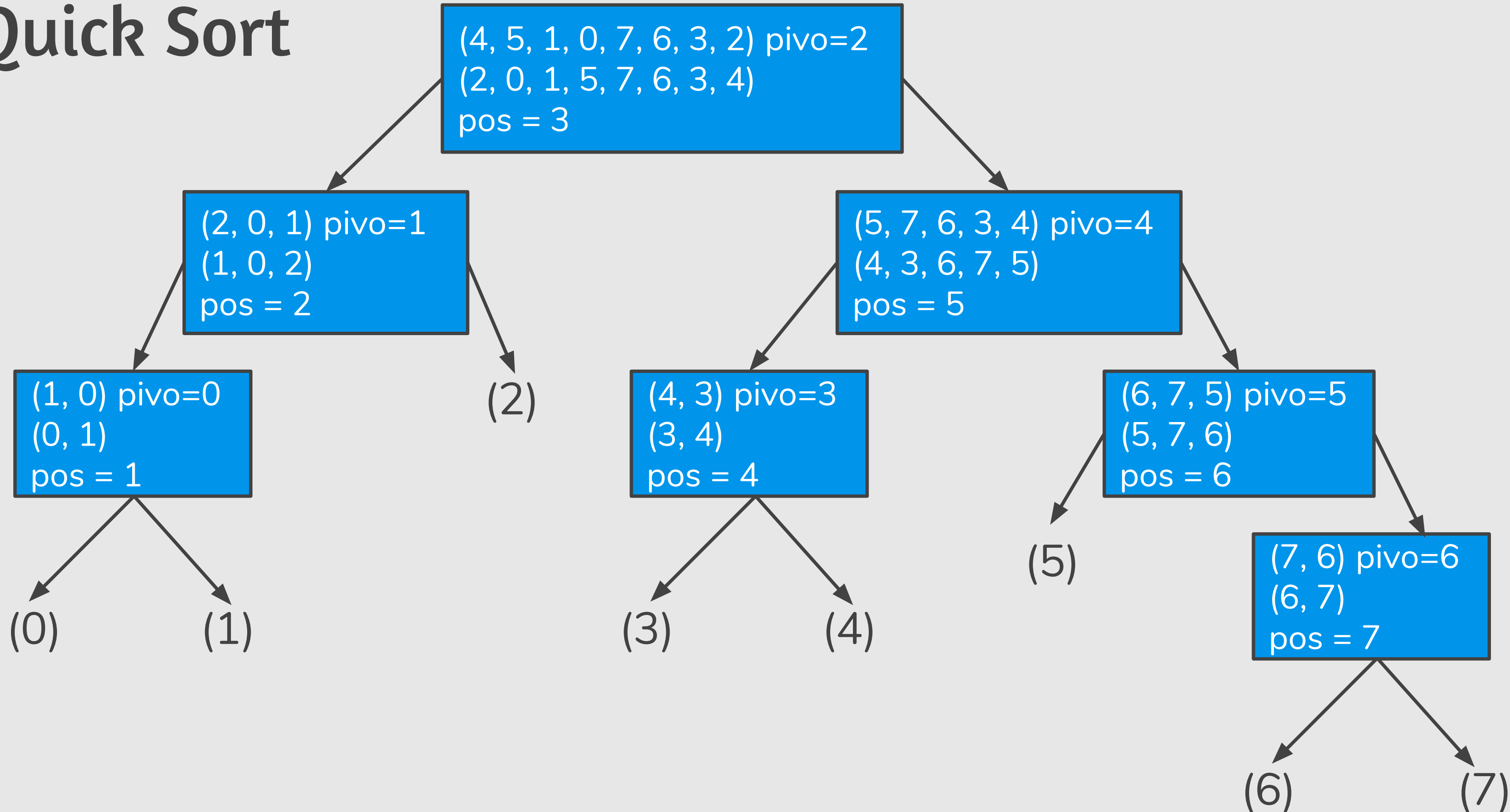
Quick Sort



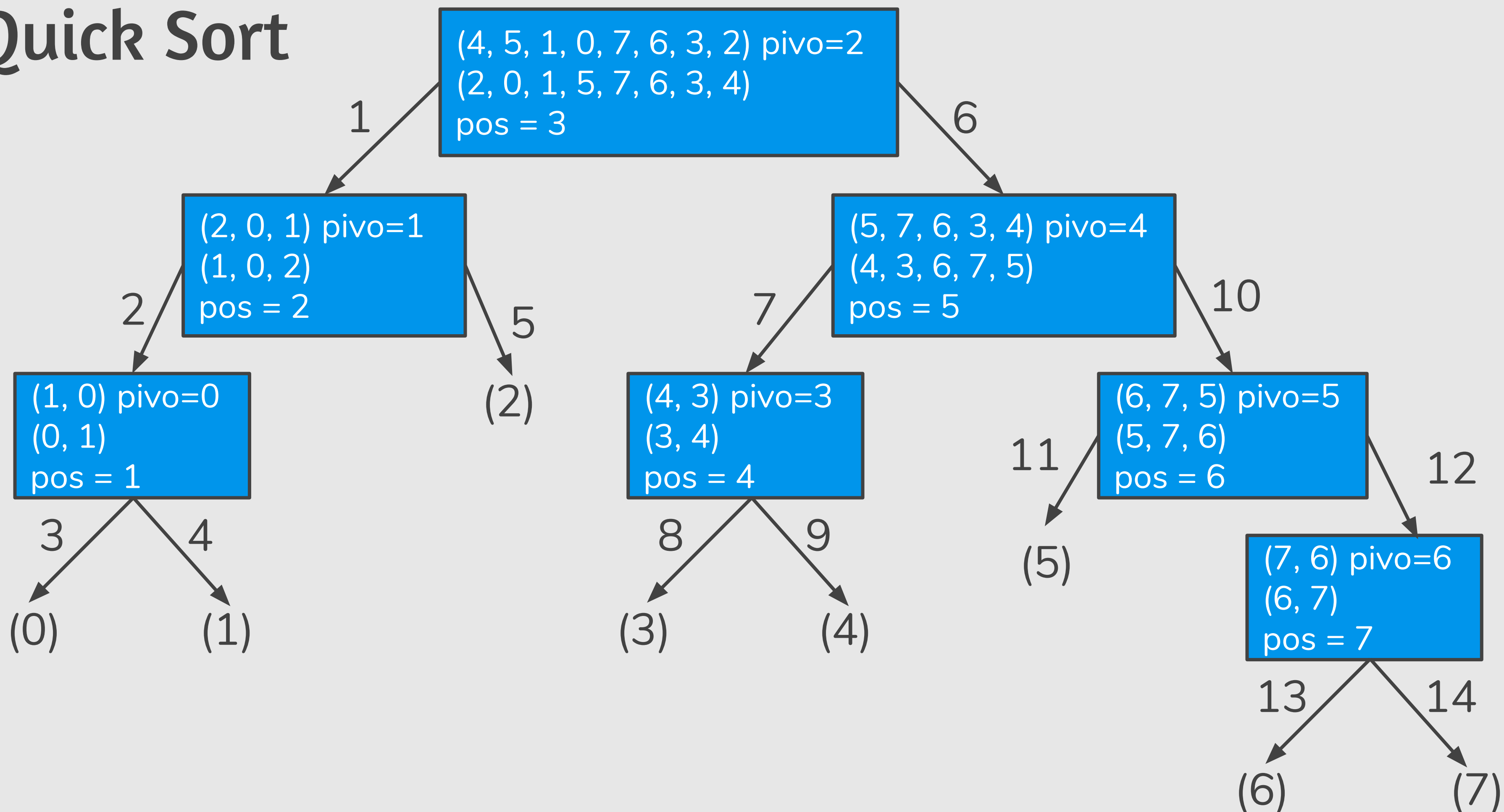
Quick Sort



Quick Sort



Quick Sort



Quick Sort

- Se o QuickSort particionar o vetor de tal forma que cada partição tenha mais ou menos o mesmo tamanho ele é muito eficiente.
- Porém se a partição for muito desigual ($n - 1$ de um lado e 1 de outro) ele é ineficiente.
- Quando um vetor já está ordenado ou quase ordenado, ocorre este caso ruim. Por que?

Quick Sort: Tratando o pior caso

- Podemos implementar o QuickSort de tal forma a diminuirmos a chance de ocorrência do pior caso.
- Ao invés de escolhermos o pivô como um elemento de uma posição fixa, podemos escolher como pivô o elemento de uma posição aleatória.
- Podemos usar a função `random.randint(a,b)` da biblioteca `random` que retorna um número de forma aleatória entre `a` e `b`.

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
```

```python
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.

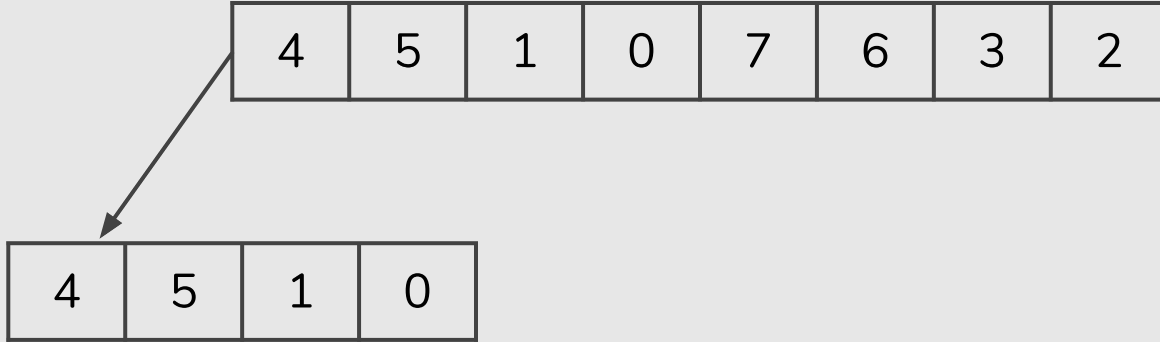
Merge Sort

```python
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)
```

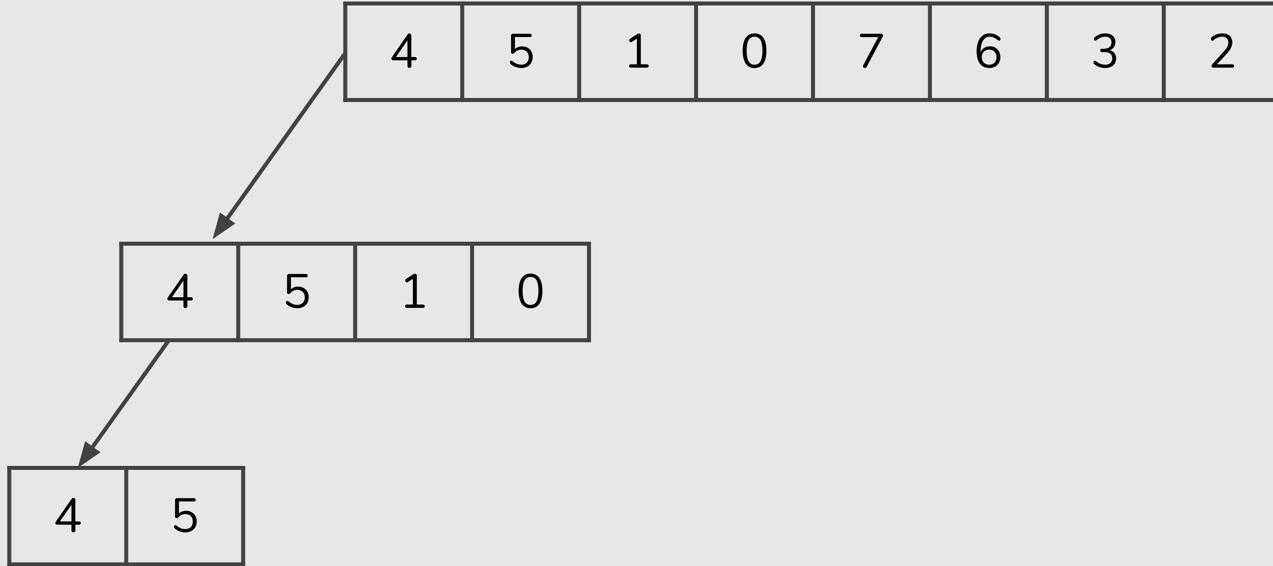
Merge Sort

4	5	1	0	7	6	3	2

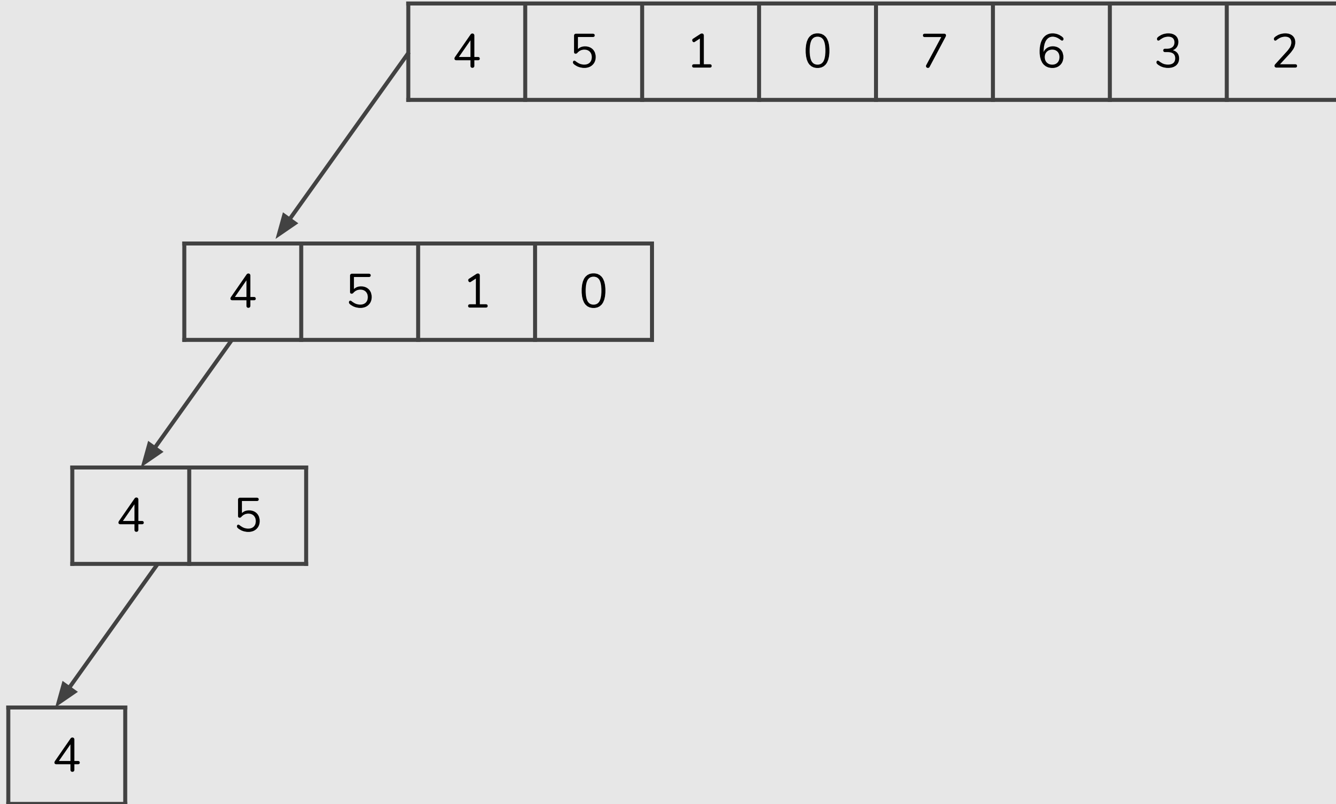
Merge Sort



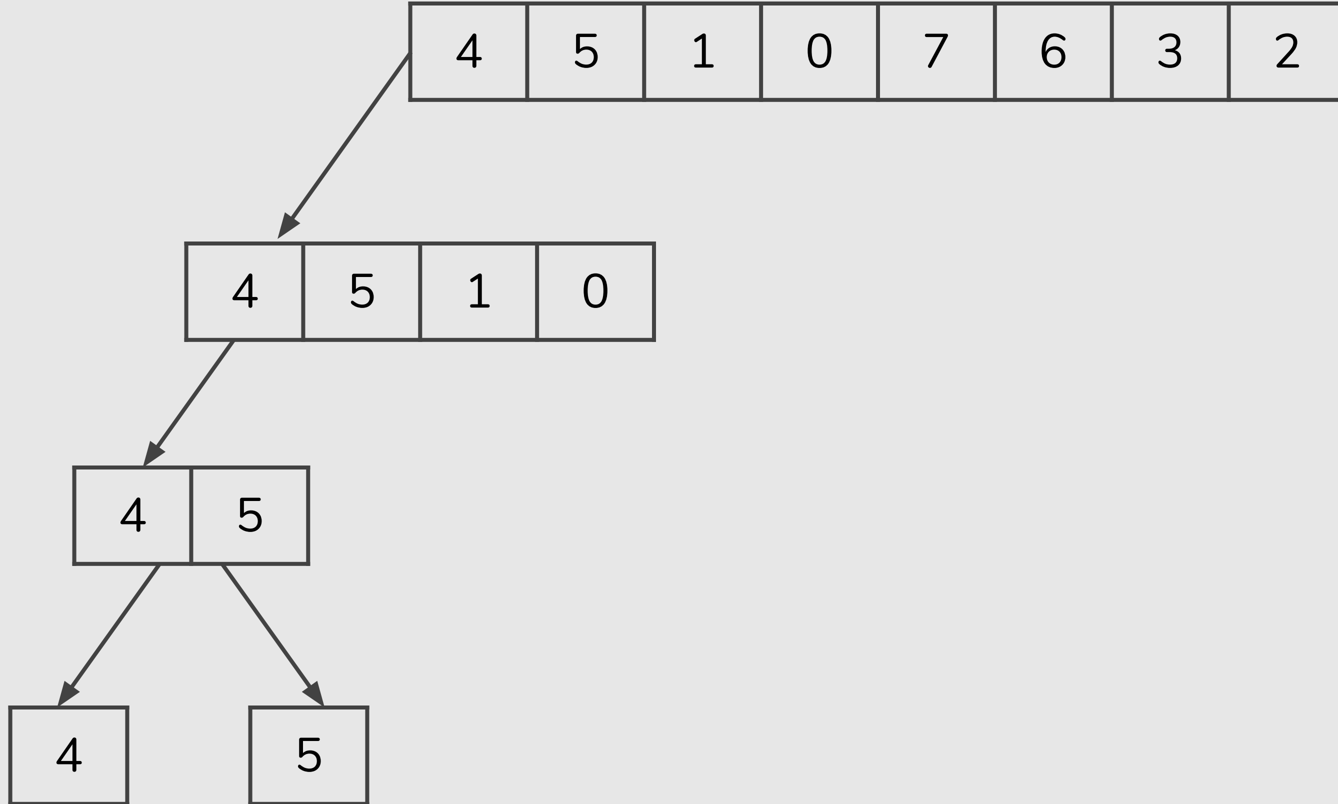
Merge Sort



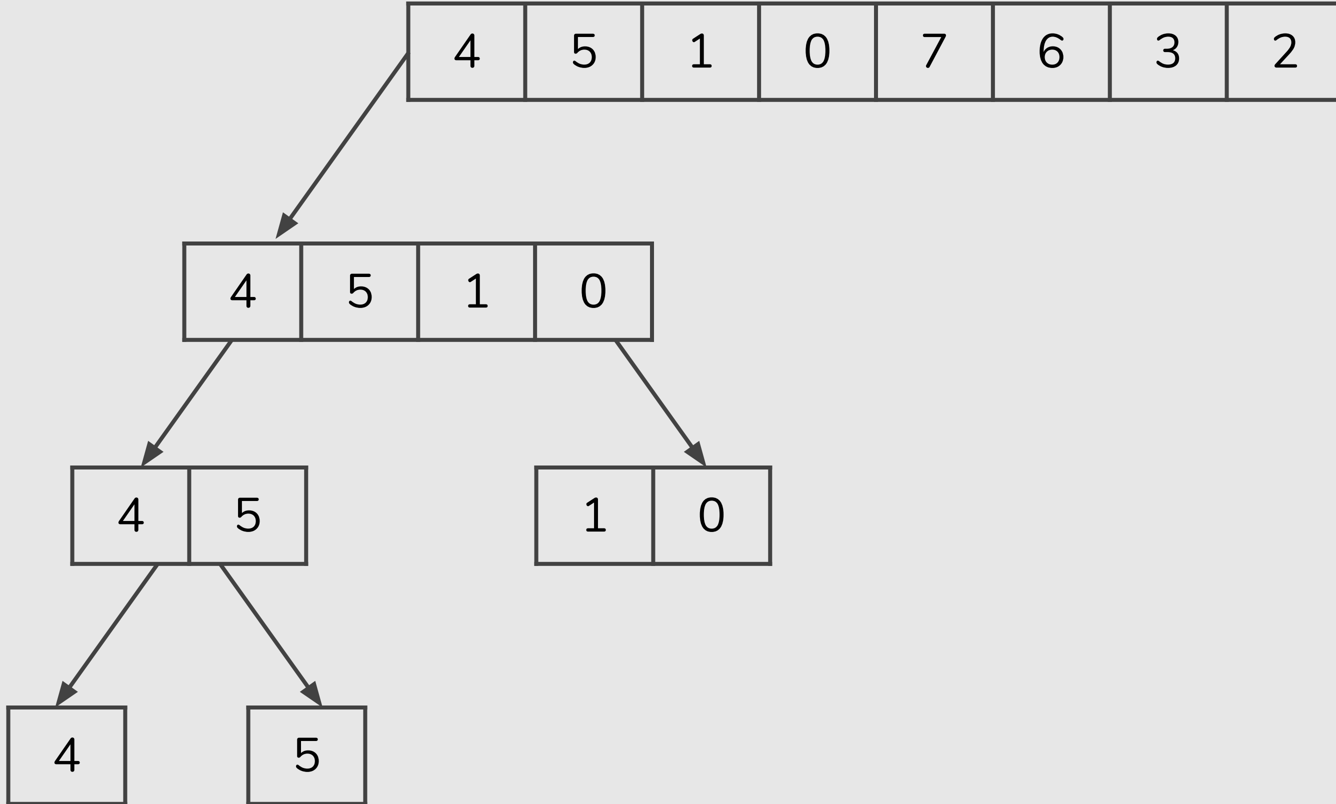
Merge Sort



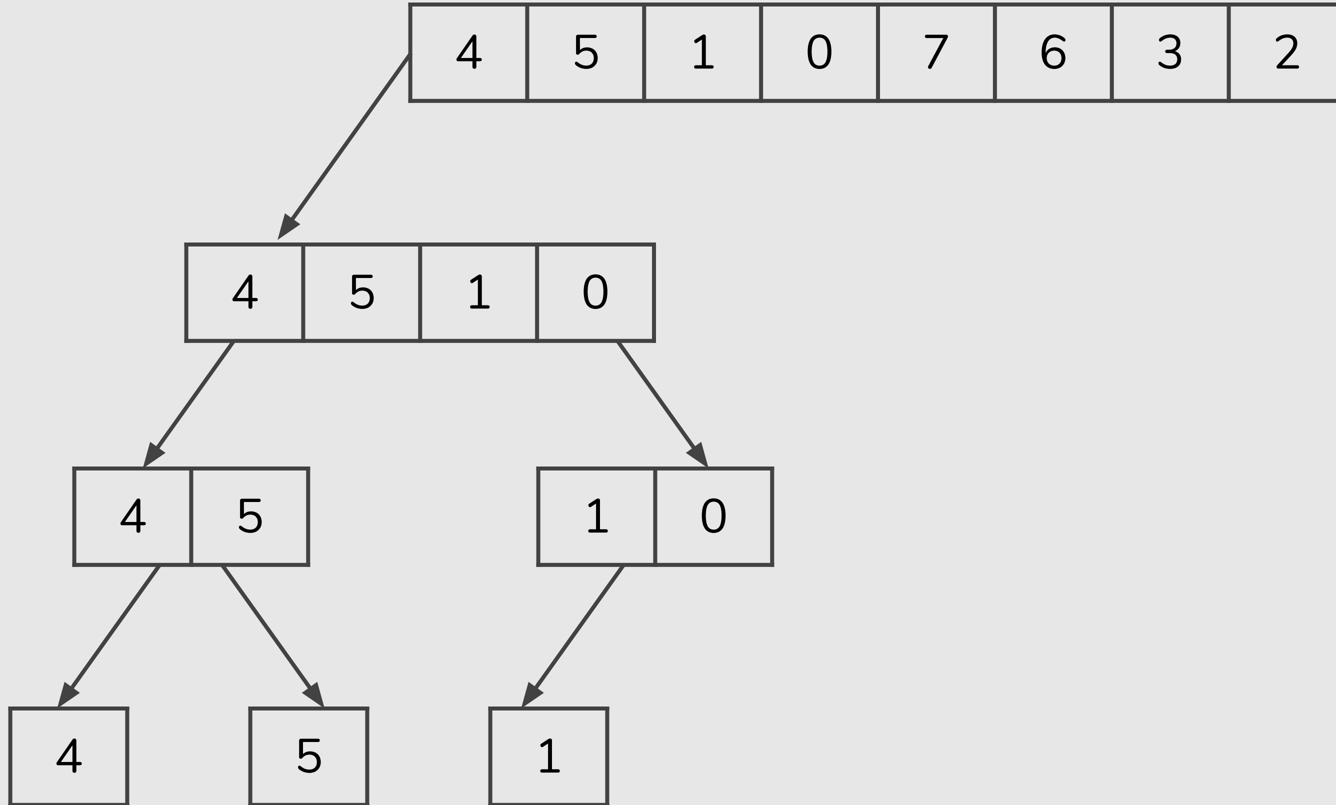
Merge Sort



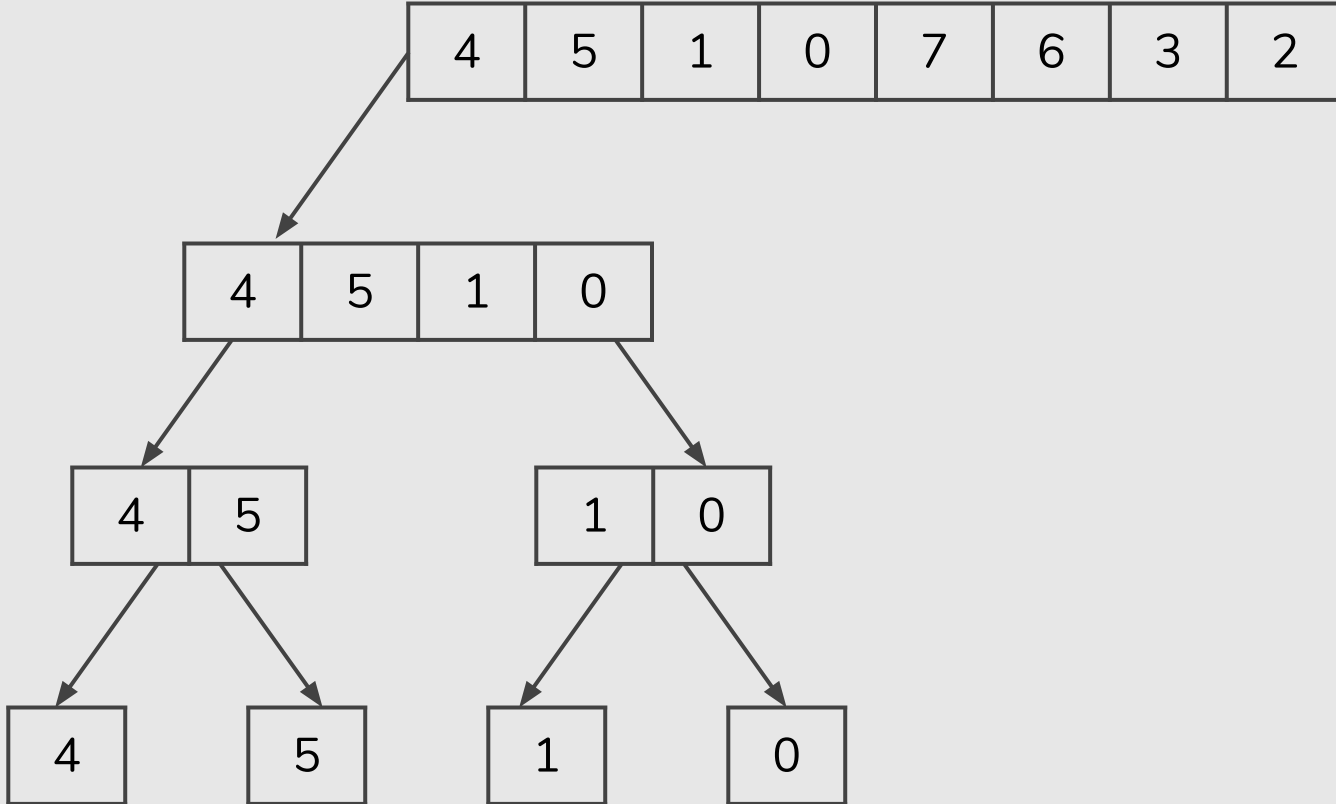
Merge Sort



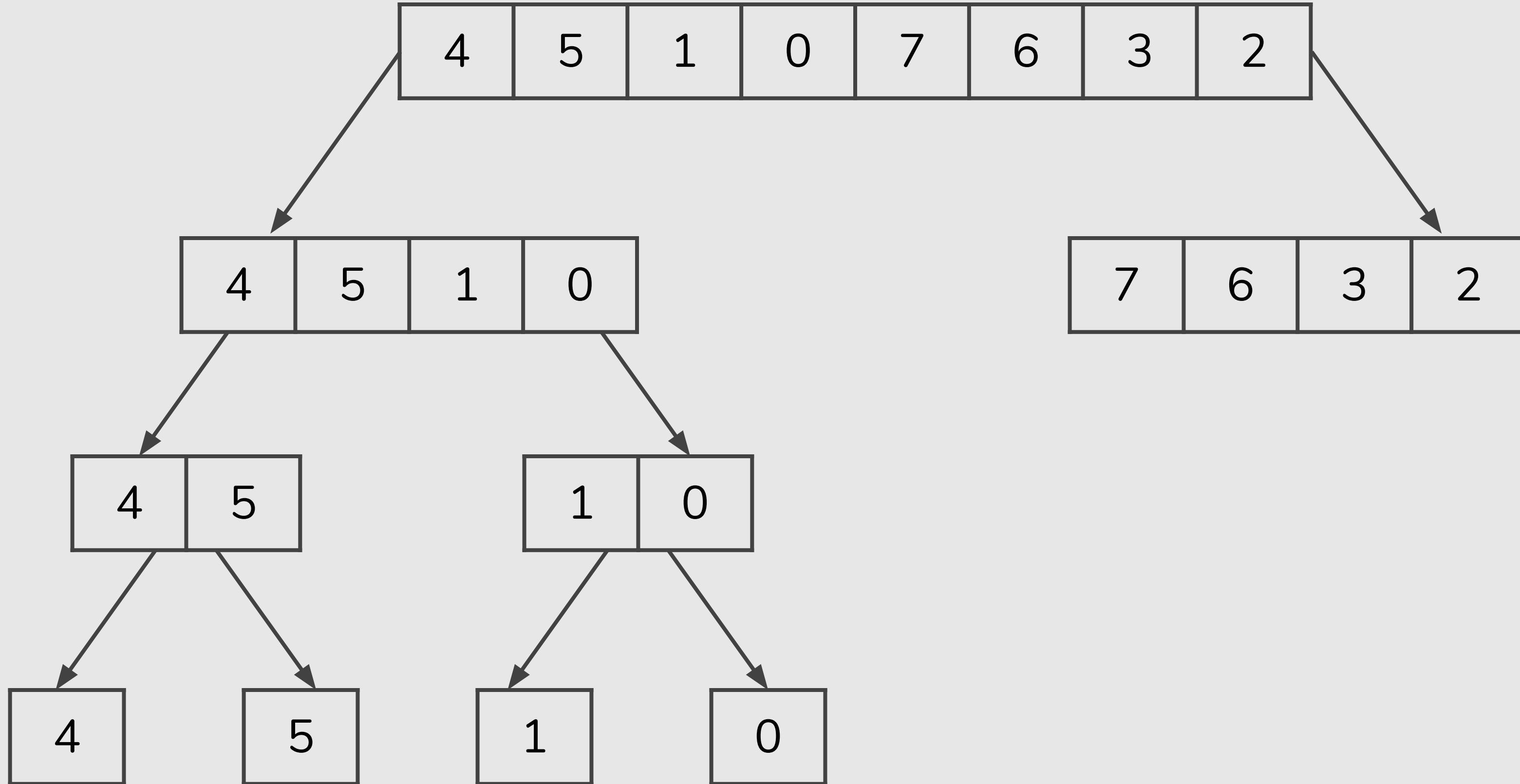
Merge Sort



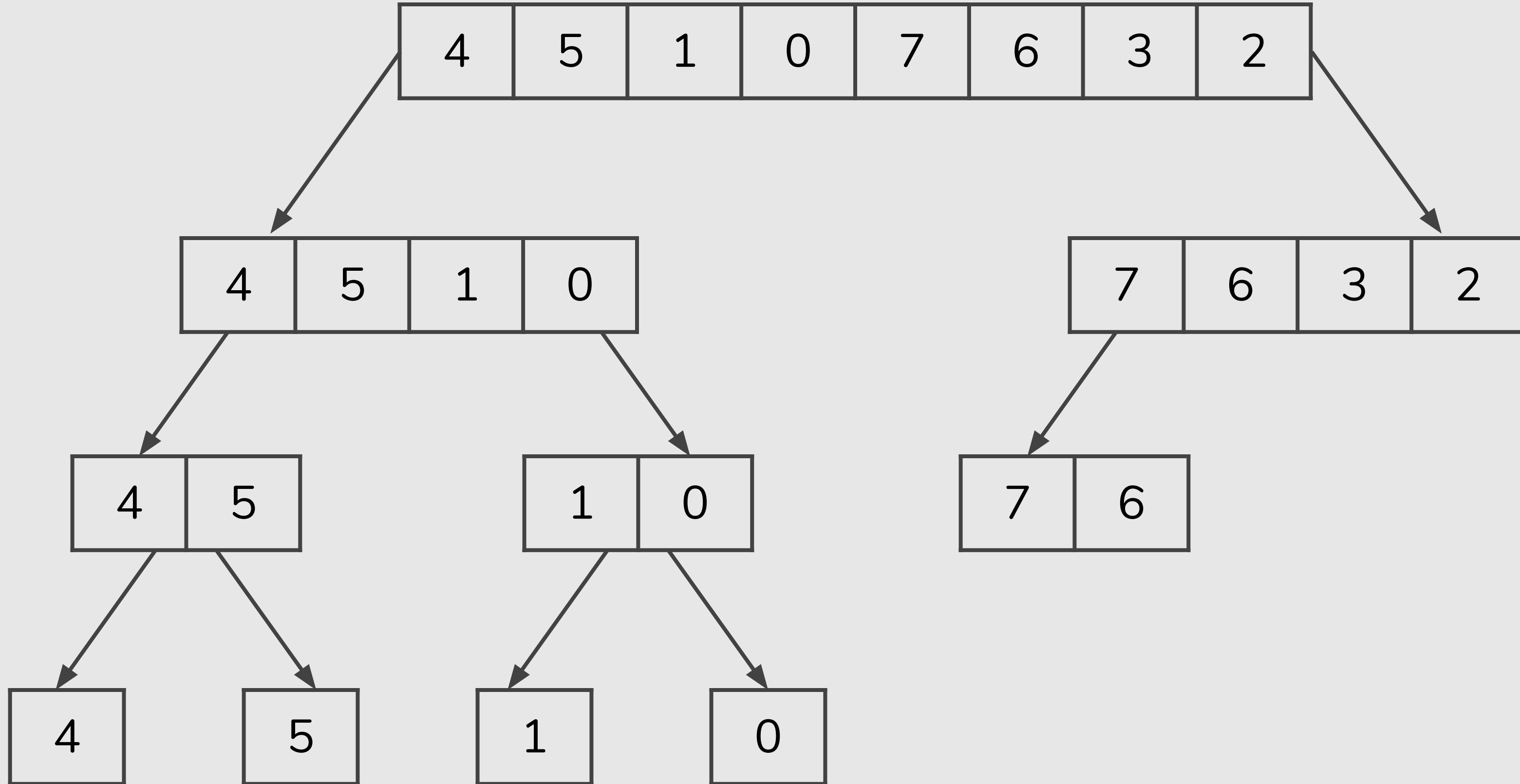
Merge Sort



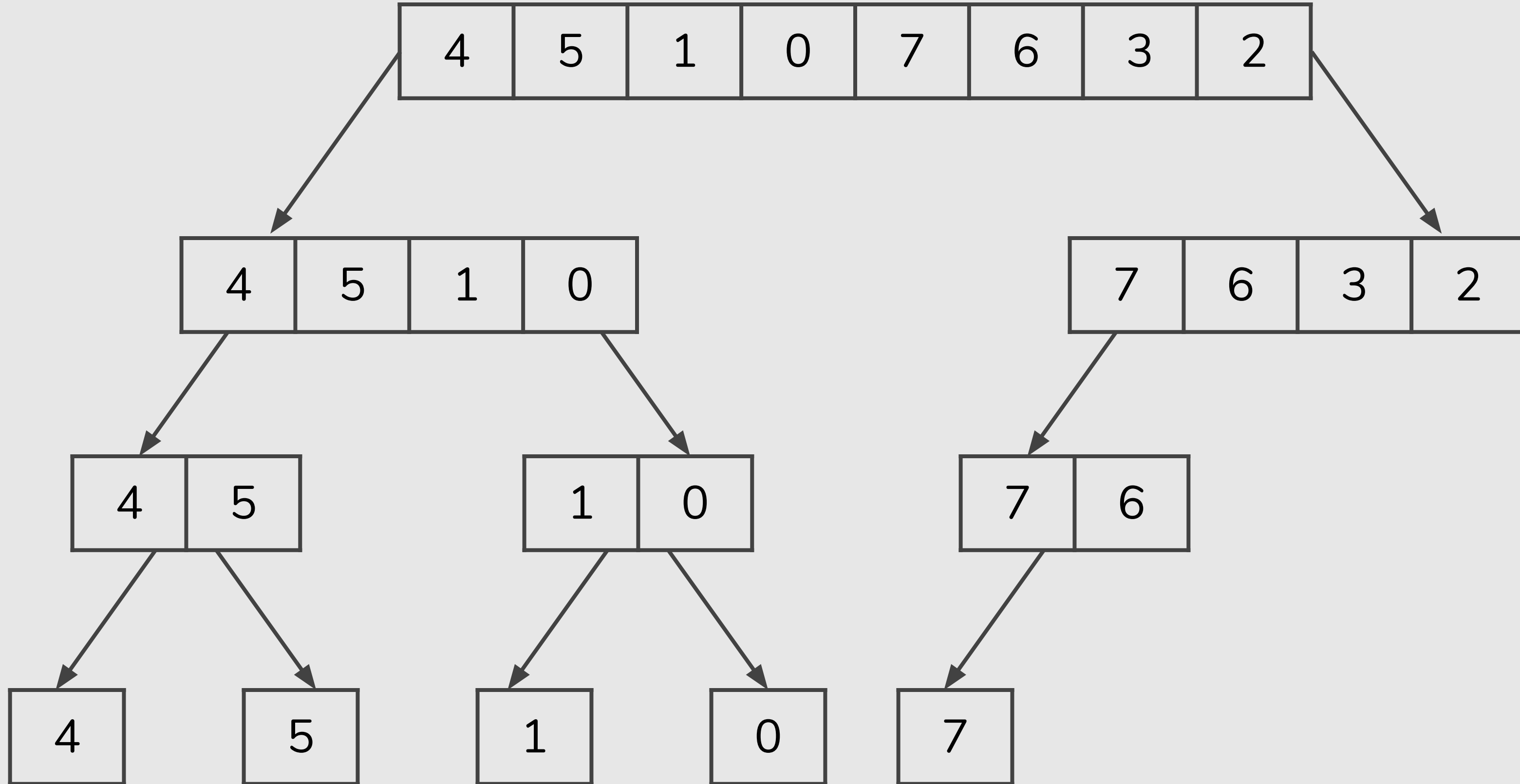
Merge Sort



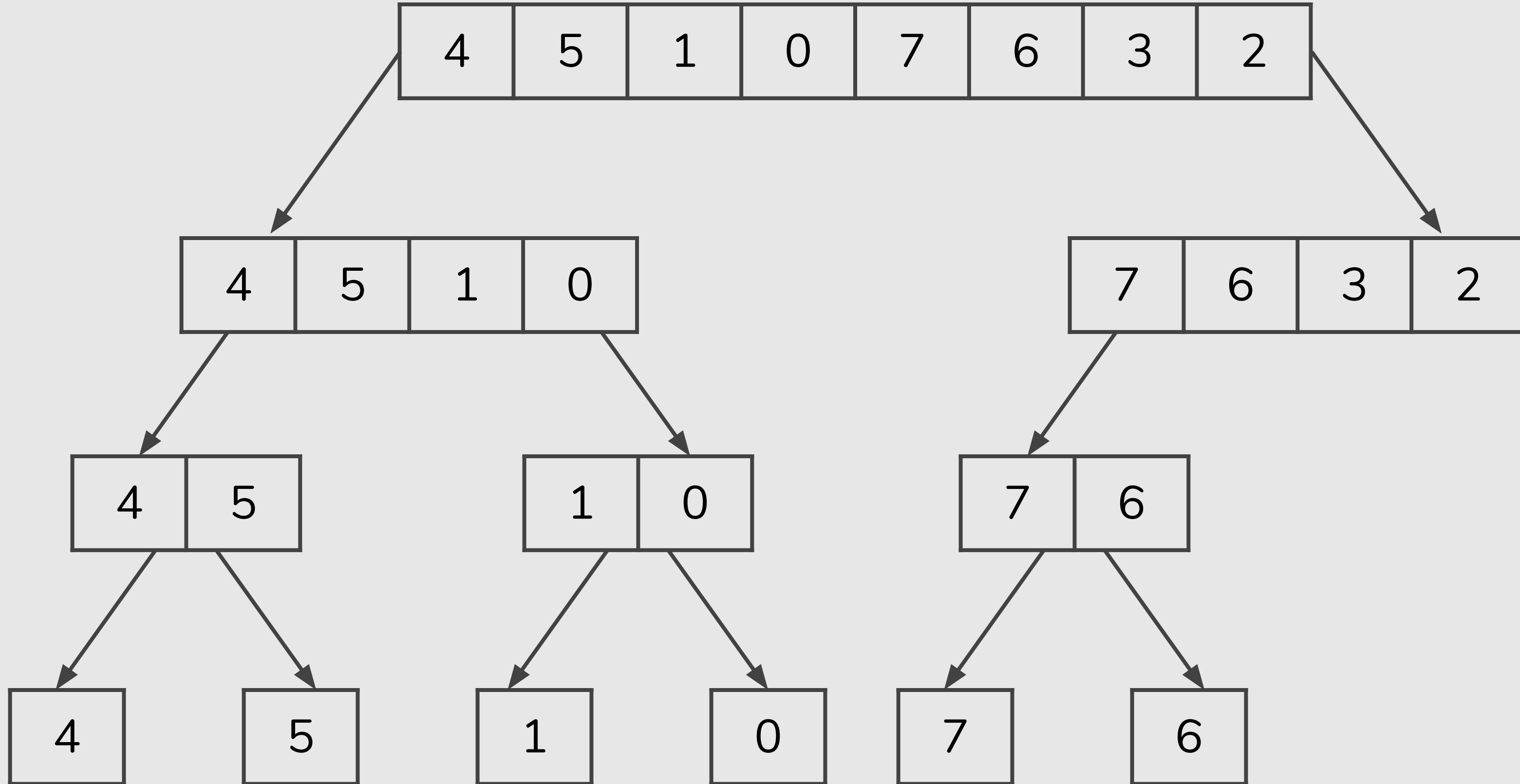
Merge Sort



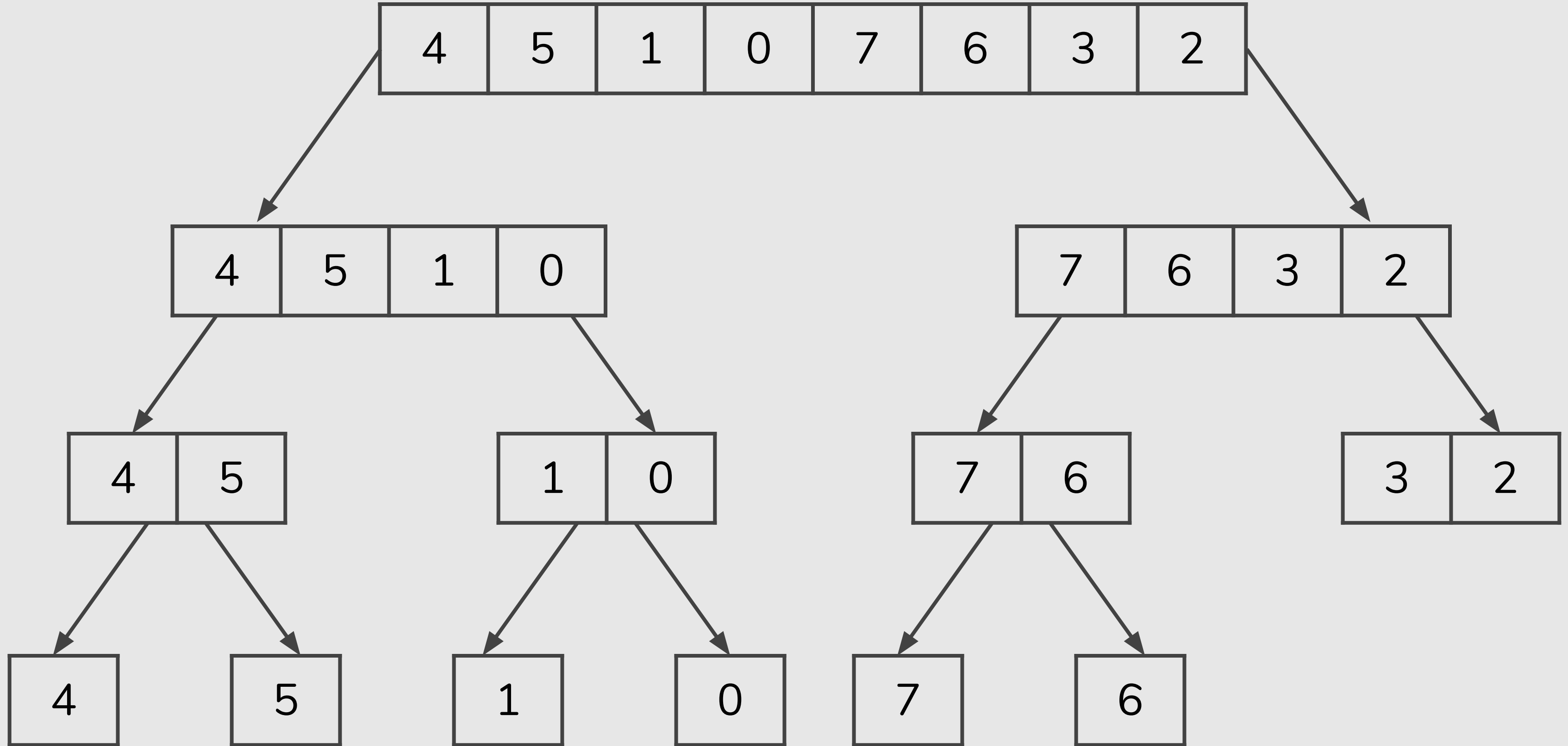
Merge Sort



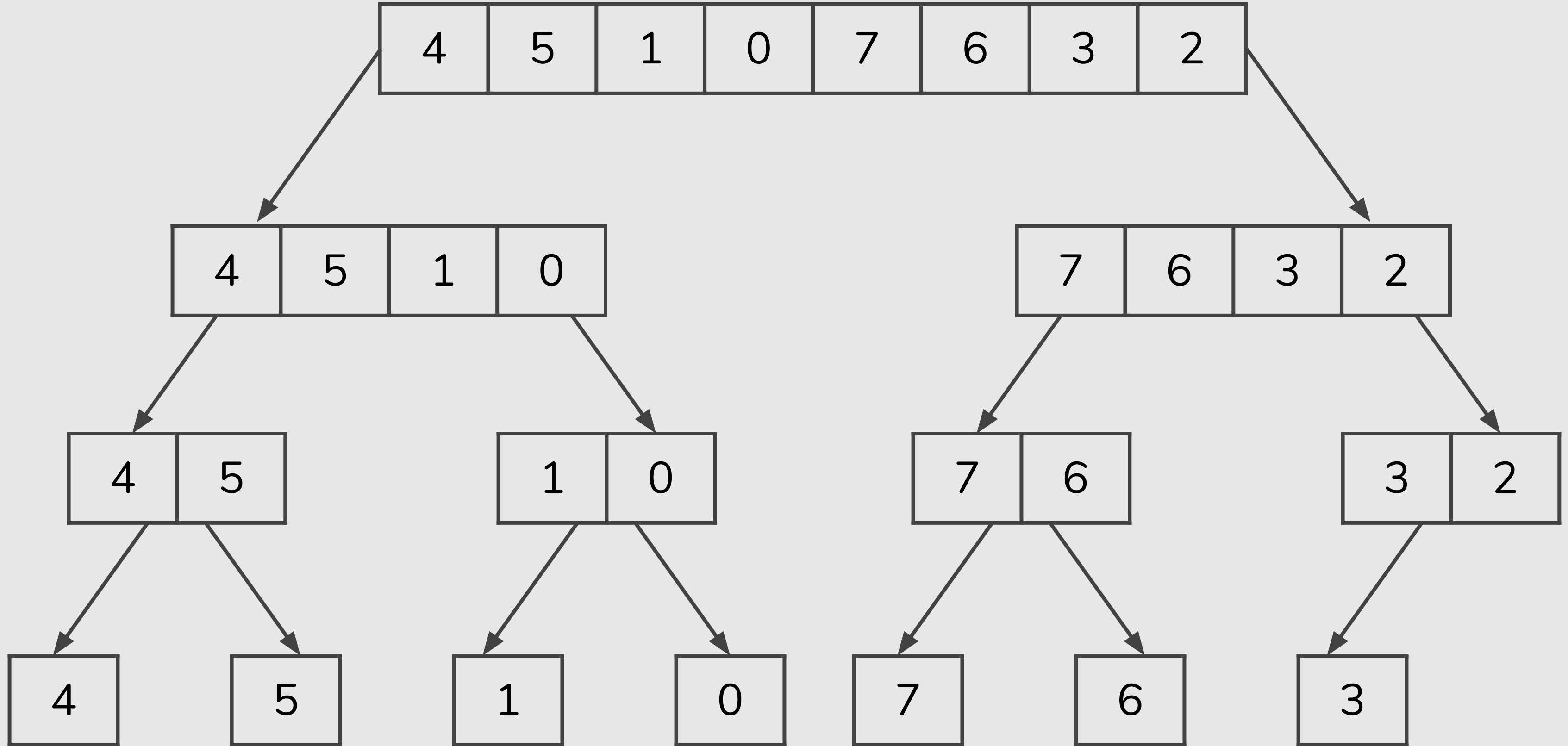
Merge Sort



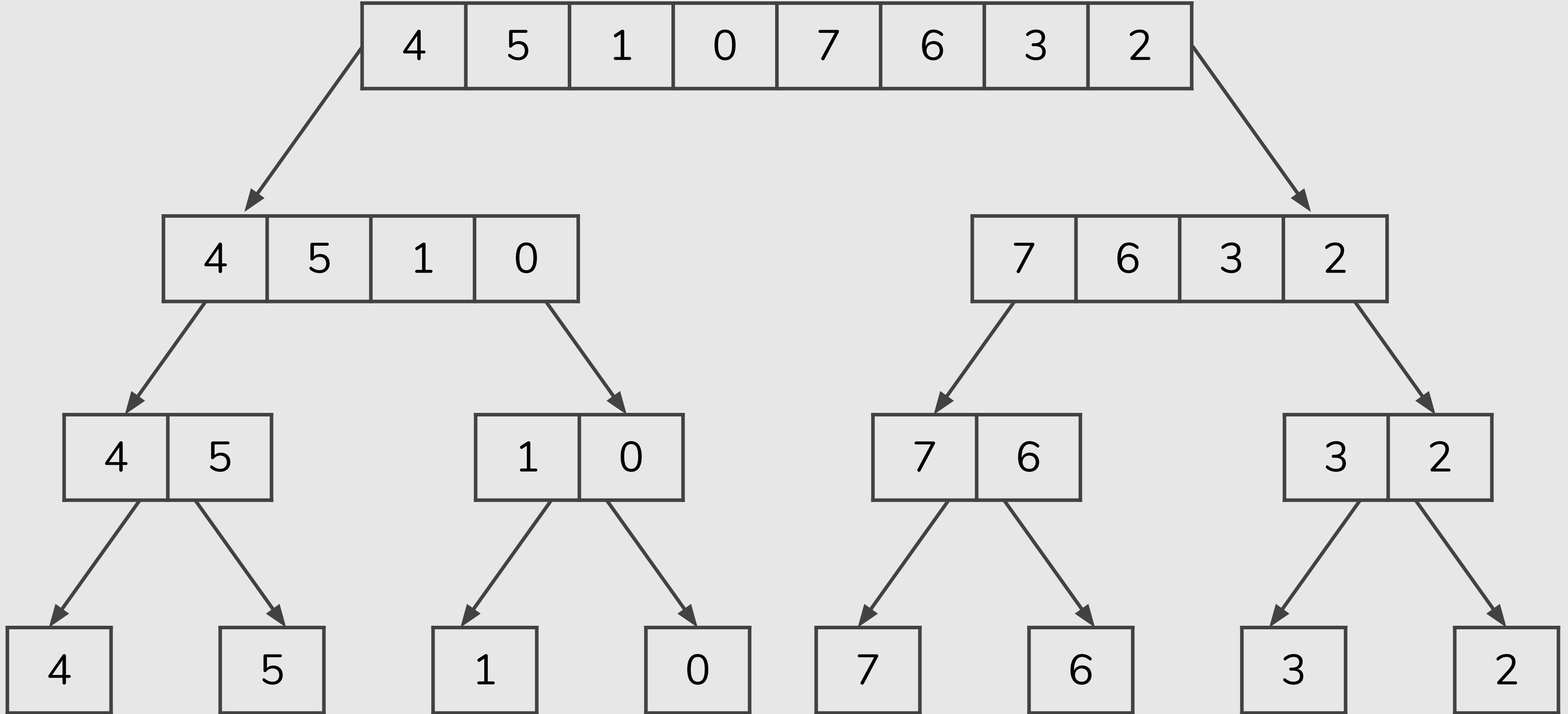
Merge Sort



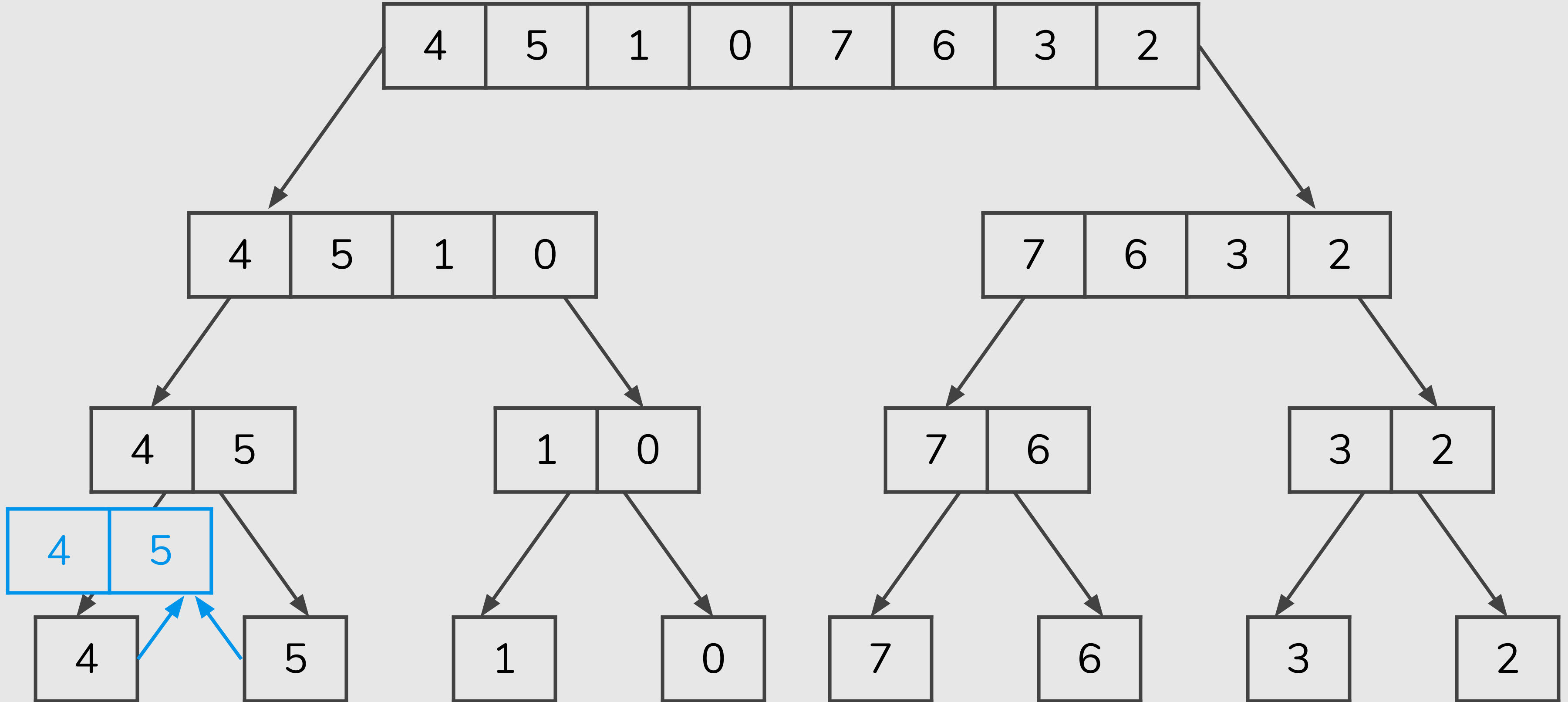
Merge Sort



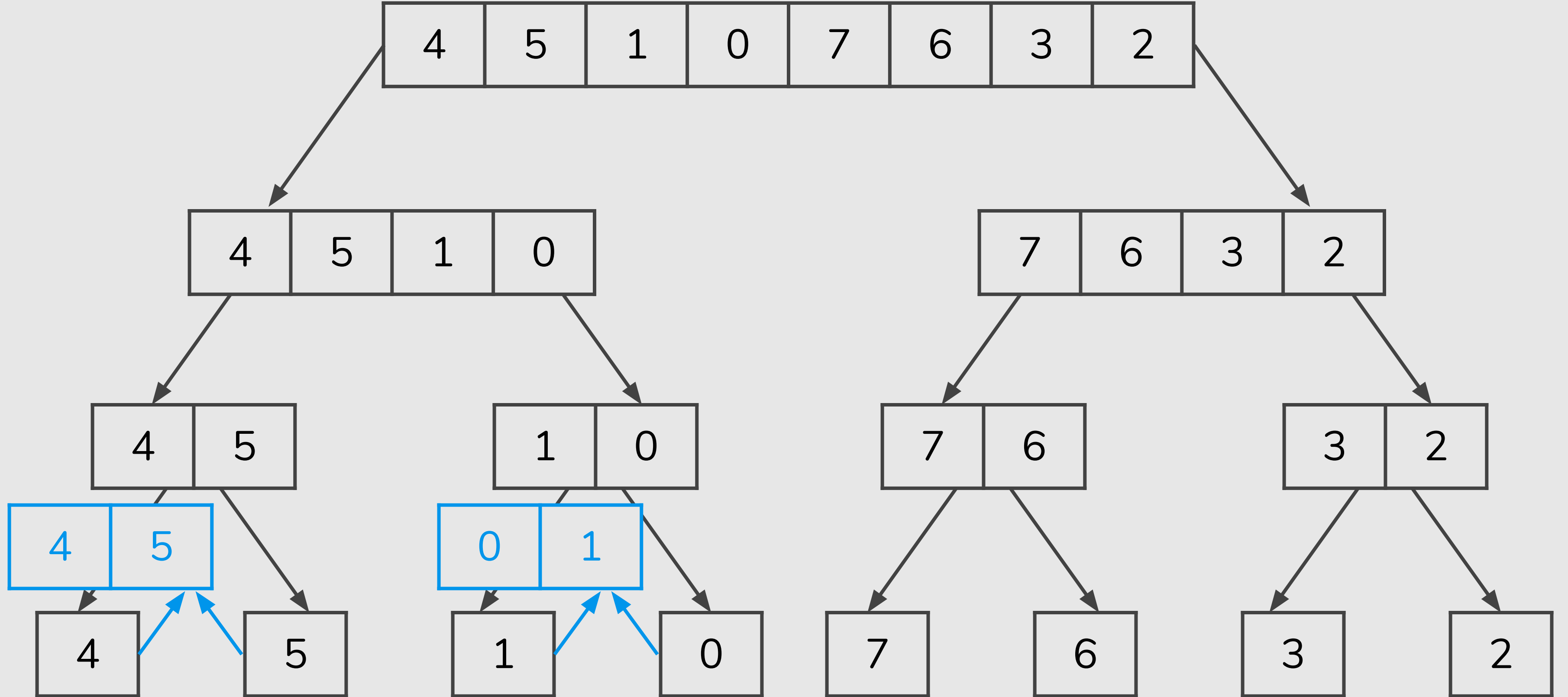
Merge Sort



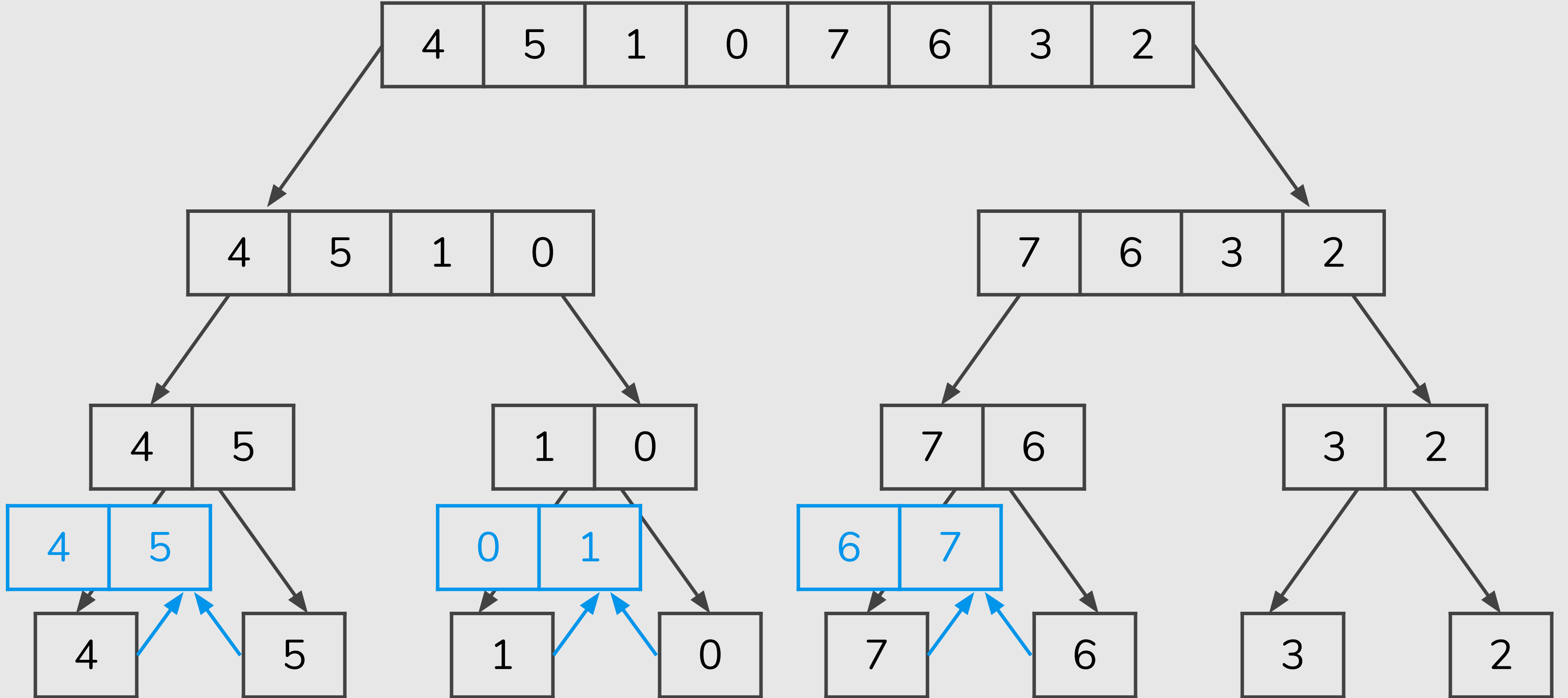
Merge Sort



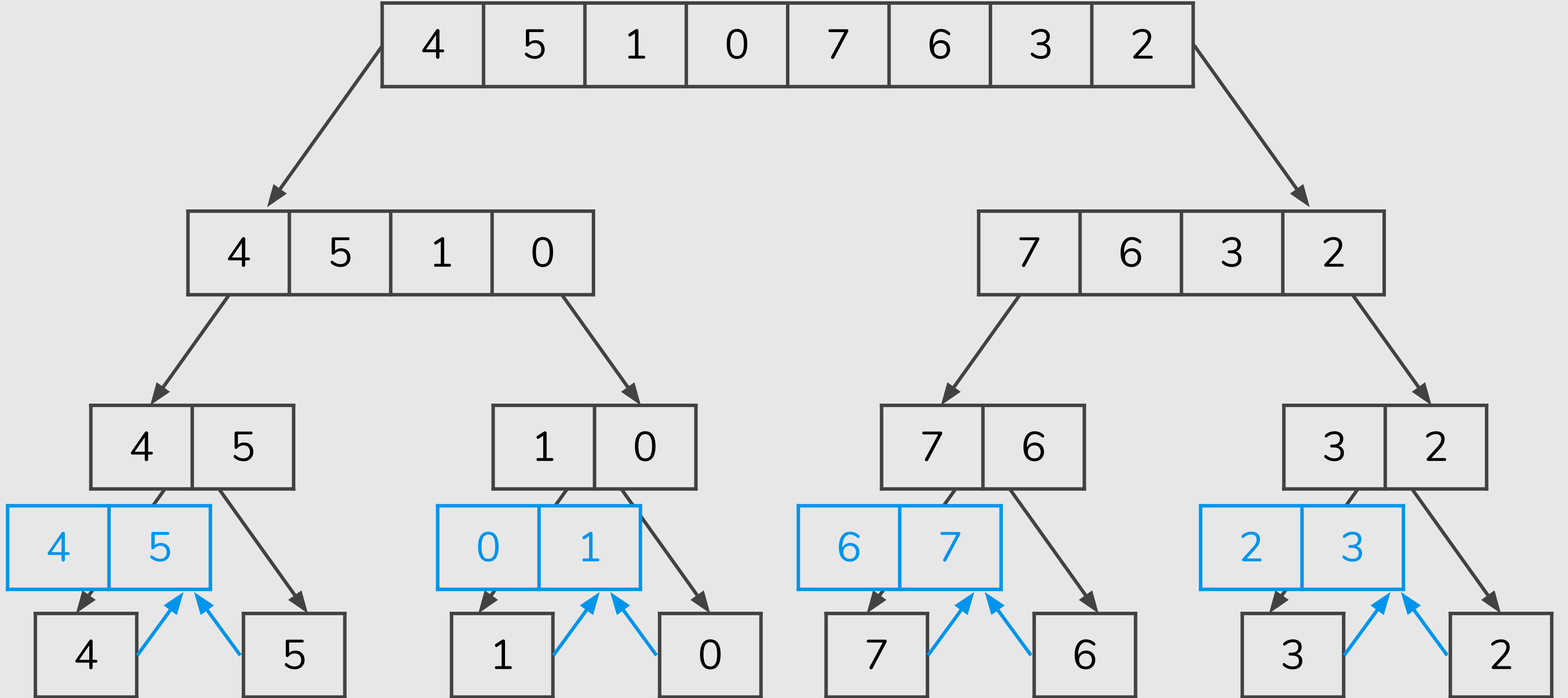
Merge Sort



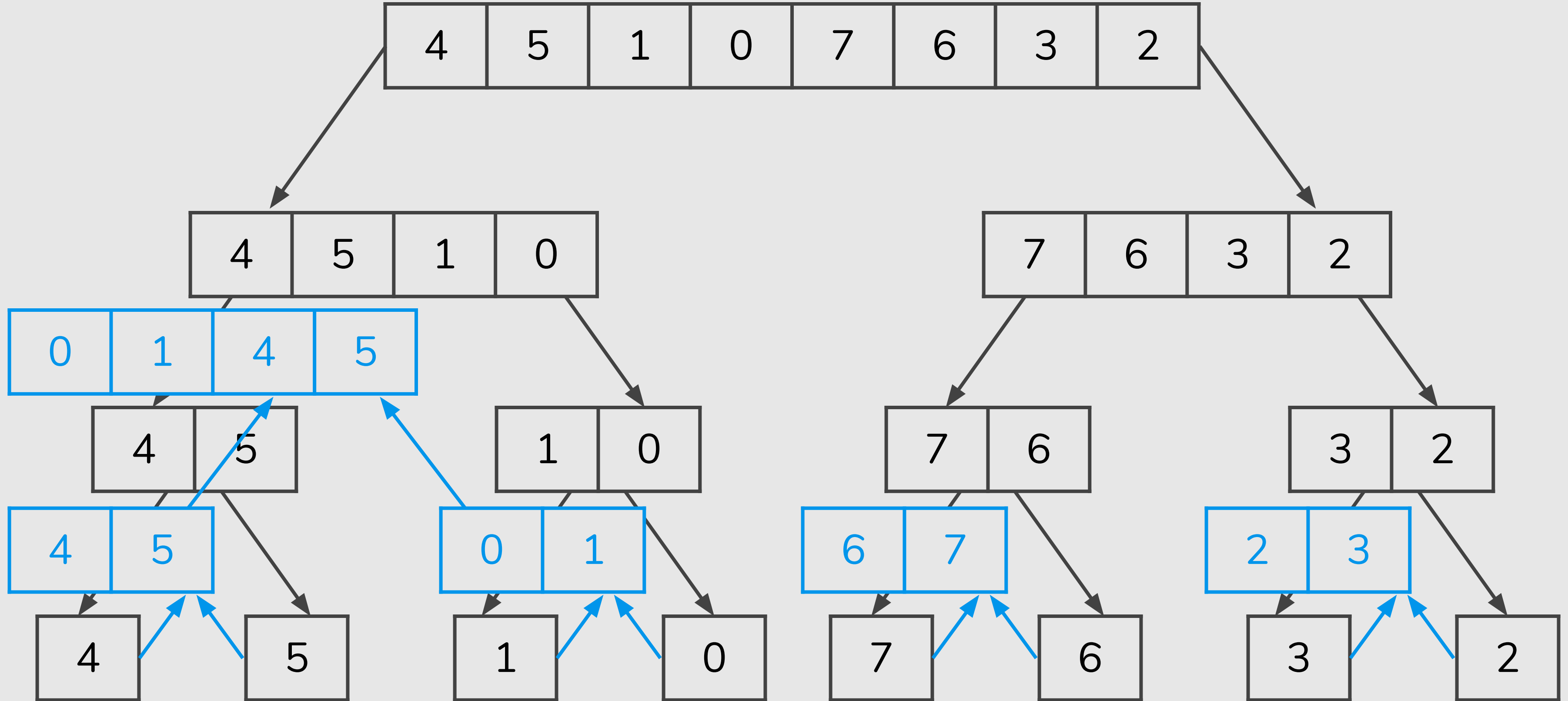
Merge Sort



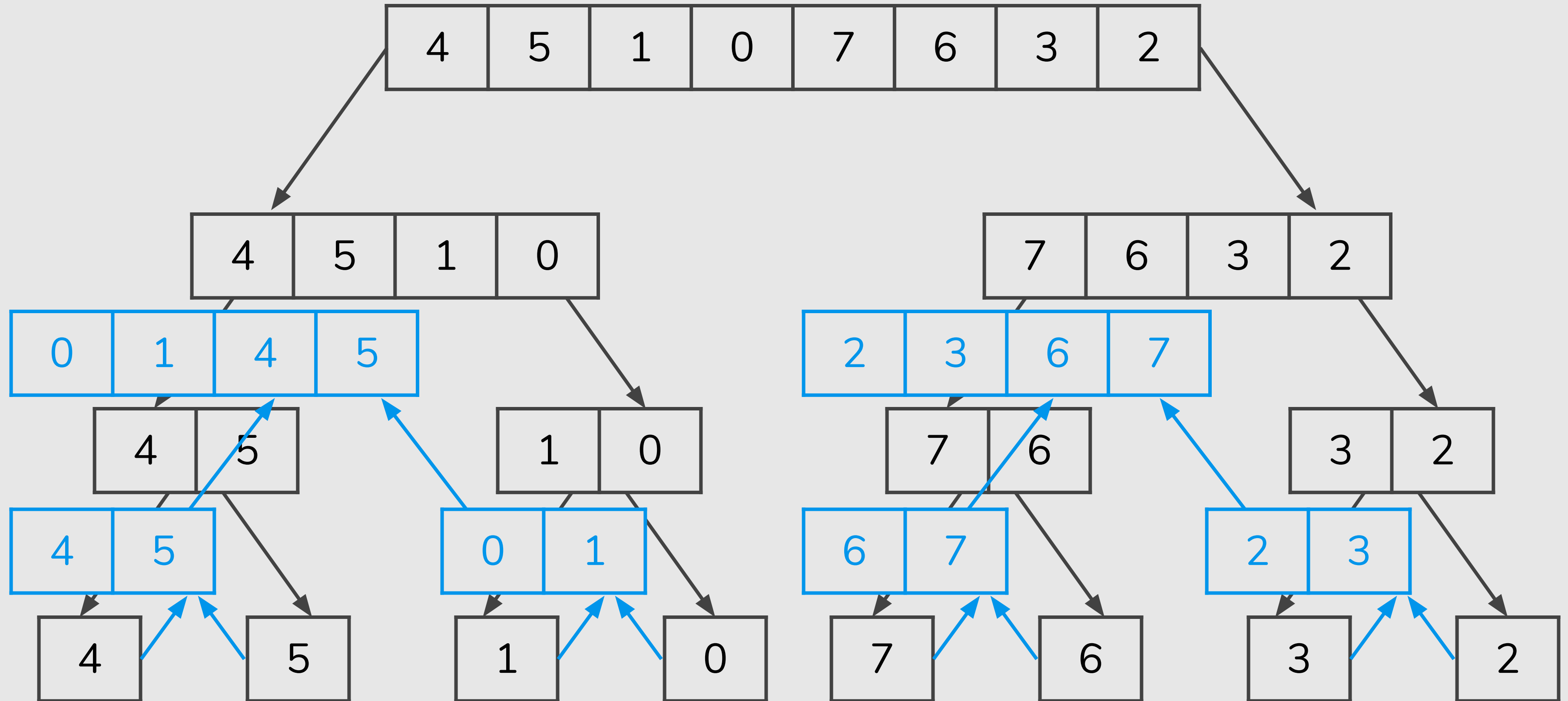
Merge Sort



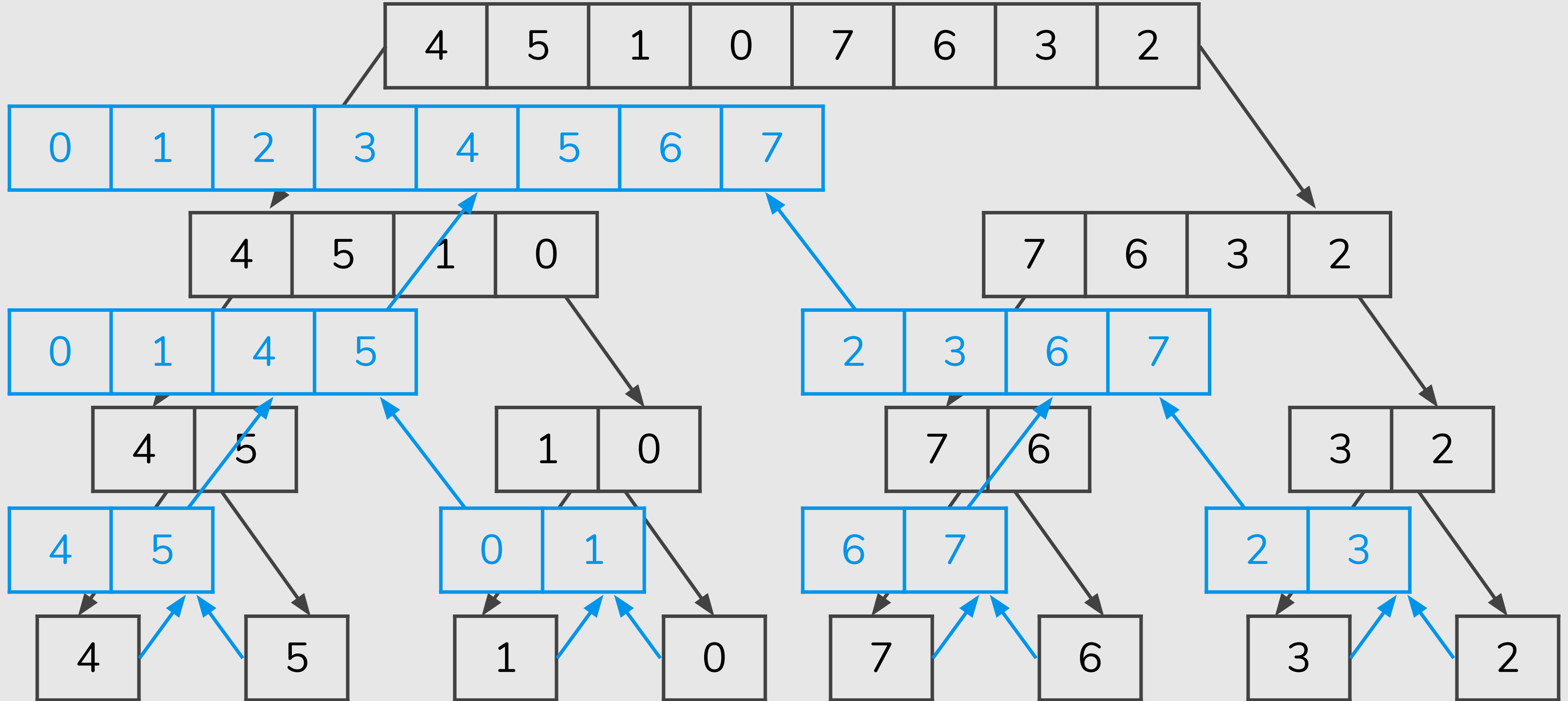
Merge Sort



Merge Sort



Merge Sort



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.

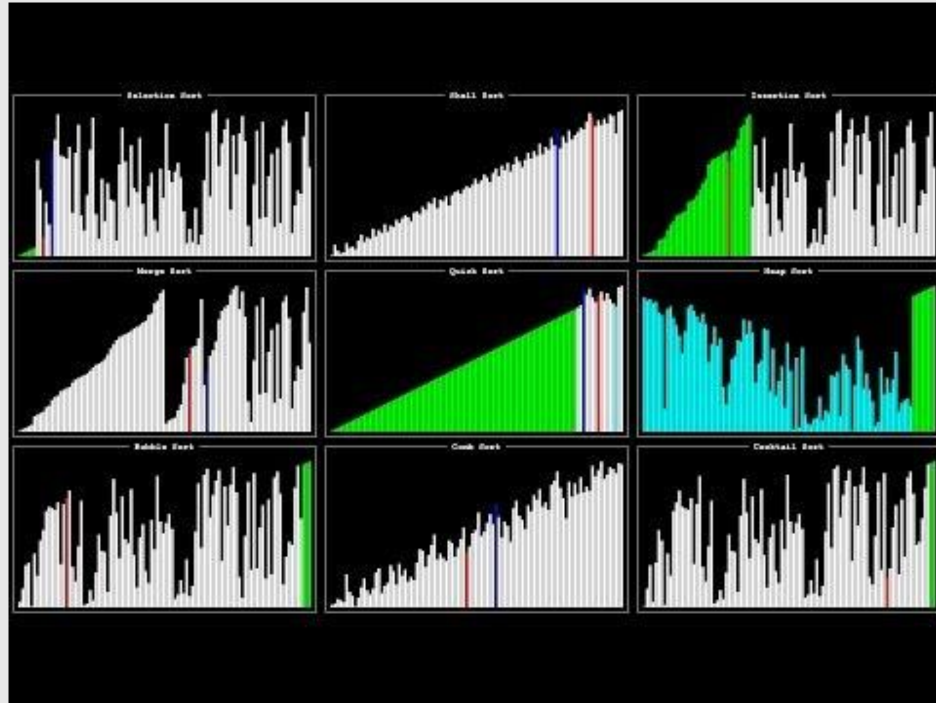
```
v = [12, 90, 47, -9, 78, 45, 78, 3323, 1, 2, 34, 20]
aux = [0 for i in range(12)] # tem o mesmo tamanho de v
print(v)
mergeSort(v, 0, 11, aux)
print(v)
```

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



Referências

- Os slides dessa aula foram baseados no material de MC102 do Prof. Eduardo Xavier (IC/Unicamp).