Merge Sort

Prof. Dr. Eleandro Maschio
Tecnologia em Sistemas para Internet
Câmpus Guarapuava
Universidade Tecnológica Federal do Paraná (UTFPR)

Problema

(**1**) Considere o conjunto de cartas de um baralho, excetuando-se os curingas. São 52 cartas.

(**2**) Assuma:

> > >

(**3**) Escolha e aplique um dos métodos de ordenação vistos:

- Selection Sort (por **seleção**).
- Bubble Sort (por **troca**).
- Insertion Sort (por **inserção**).

(**4**) A sequência ordenada será:

- A , ..., K , A , ..., K , A , ..., K , A , ..., K .

Problema

(**5**) Perceba, então, a **inviabilidade** dos métodos anteriormente vistos tratarem grandes conjuntos de dados.

- Grande quantidade de comparações.
- Grande quantidade de trocas.

Ordenação por Intercalação

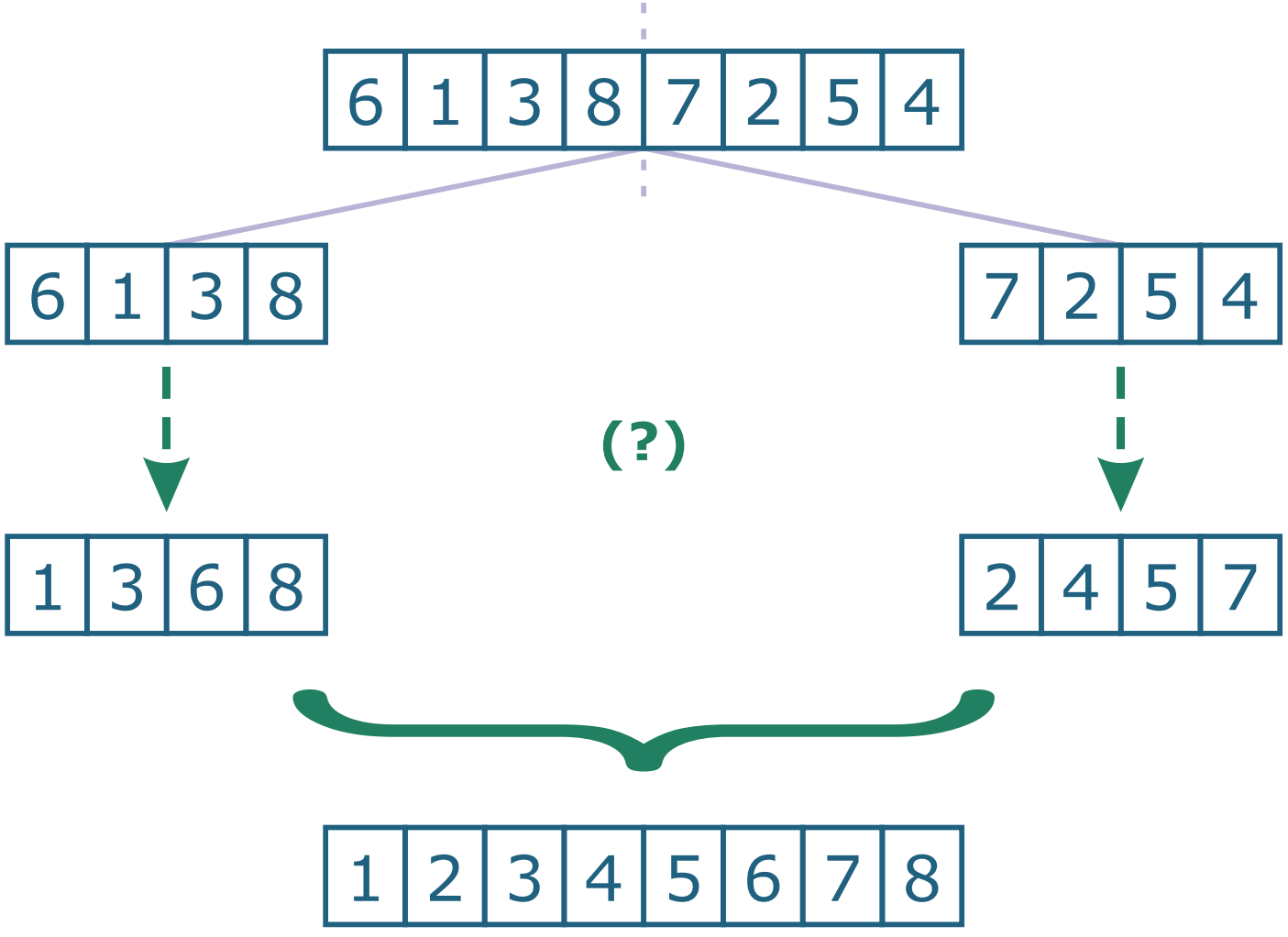
(**1**) **Divida** o monte de cartas em dois outros, proporcionais.

(**2**) **Ordene** cada um dos dois montes **separadamente**.

(**3**) **Intercale** os dois montes obedecendo a ordem.

Embora tenhamos pensado na divisão do baralho em dois montes, pode-se abordar o problema mediante a **divisão em mais montes**.

Ordenação por Intercalação



Ordenação por Intercalação

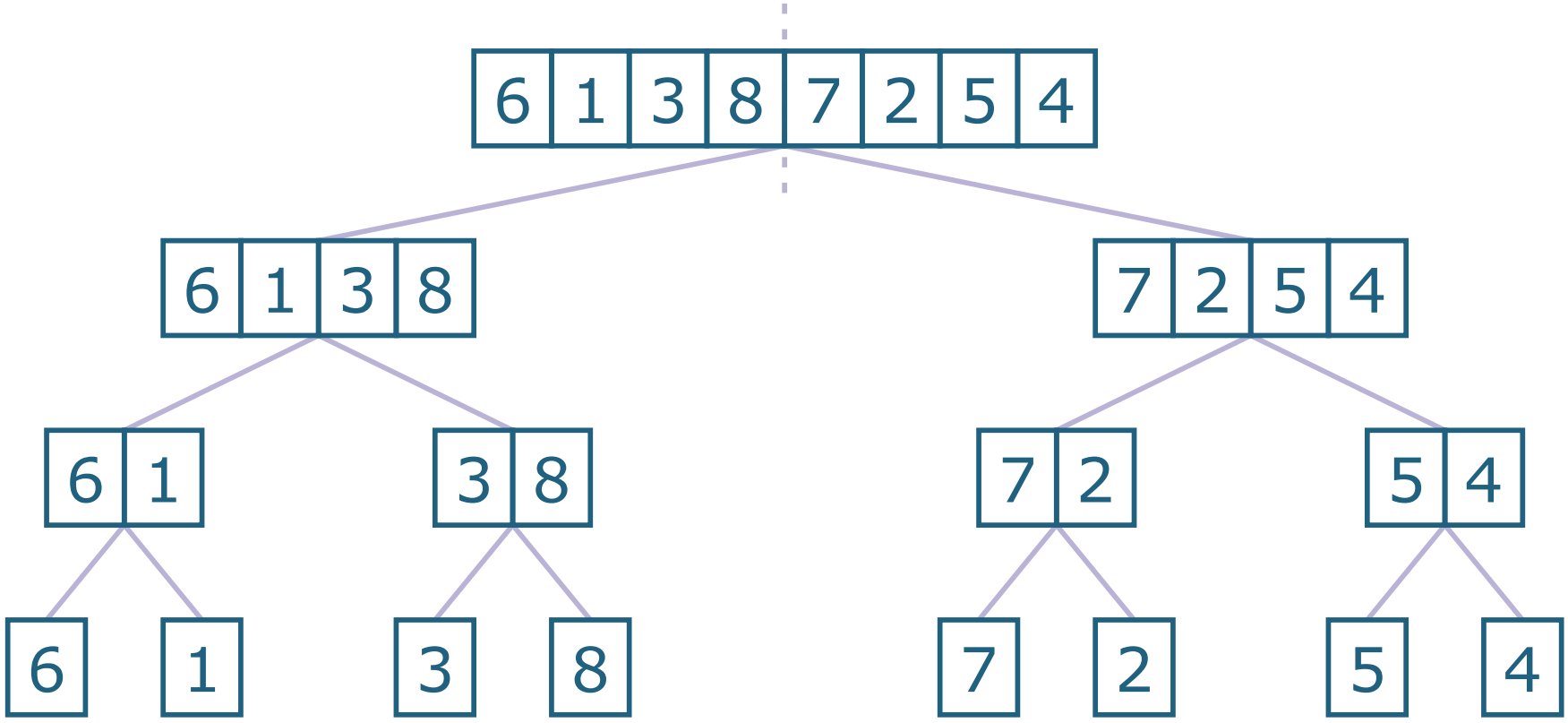
(**1**) **Divida** o monte de cartas em dois outros, proporcionais.

(**2**) **Ordene por intercalação** cada um dos dois montes **separadamente**.

Note que a ordenação agora ocorre de forma **recursiva** pelos passos aqui descritos.

(**3**) **Intercale** os dois montes obedecendo a ordem.

Ordenação por Intercalação



(?)

Técnica da Divisão e Conquista

Trata-se de uma técnica recursiva que envolve três passos em cada nível da recursão.

(**1**) **Divida** o problema em um certo número de subproblemas.

(**2**) **Conquiste** os subproblemas solucionando-os recursivamente pela técnica.

Se os subproblemas tiverem tamanhos **suficientemente pequenos**, solucioná-los de forma simples (como **caso base** da recursão).

(**3**) **Combine** as **soluções dos subproblemas** na solução de problema original.

Merge Sort

Lógica de Implementação

Proposto por Von Neumann em 1945, aplica a Técnica da **Divisão e Conquista** da seguinte forma:

(**1**) **Divida** a sequência de *n* elementos a serem ordenados em duas subsequências de *n/2* elementos cada.

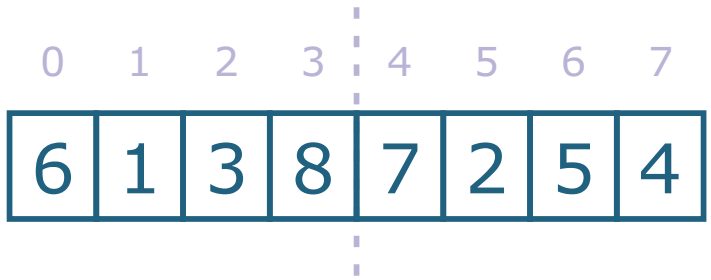
(**2**) **Conquiste** ordenando as duas subsequências recursivamente.

Considere ordenada uma subsequência com um único elemento.

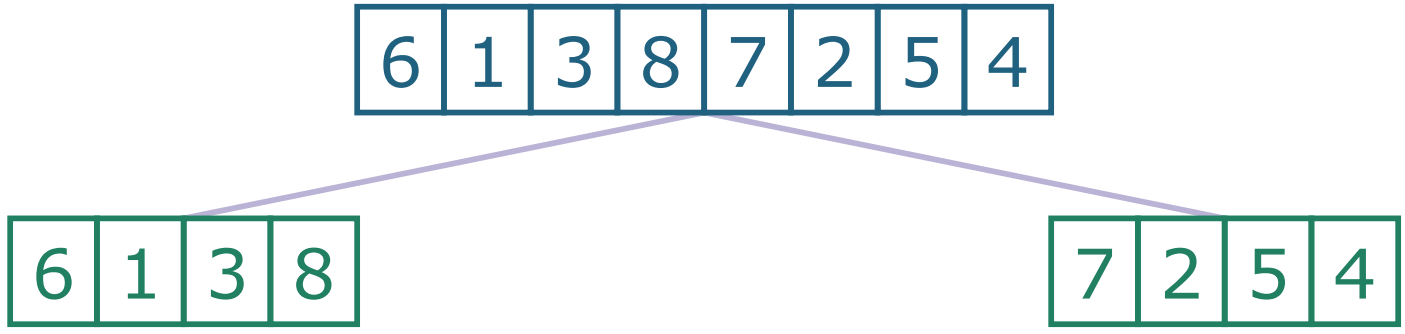
(**3**) **Combine**, através da intercalação, as duas subsequências ordenadas pelo passo anterior.

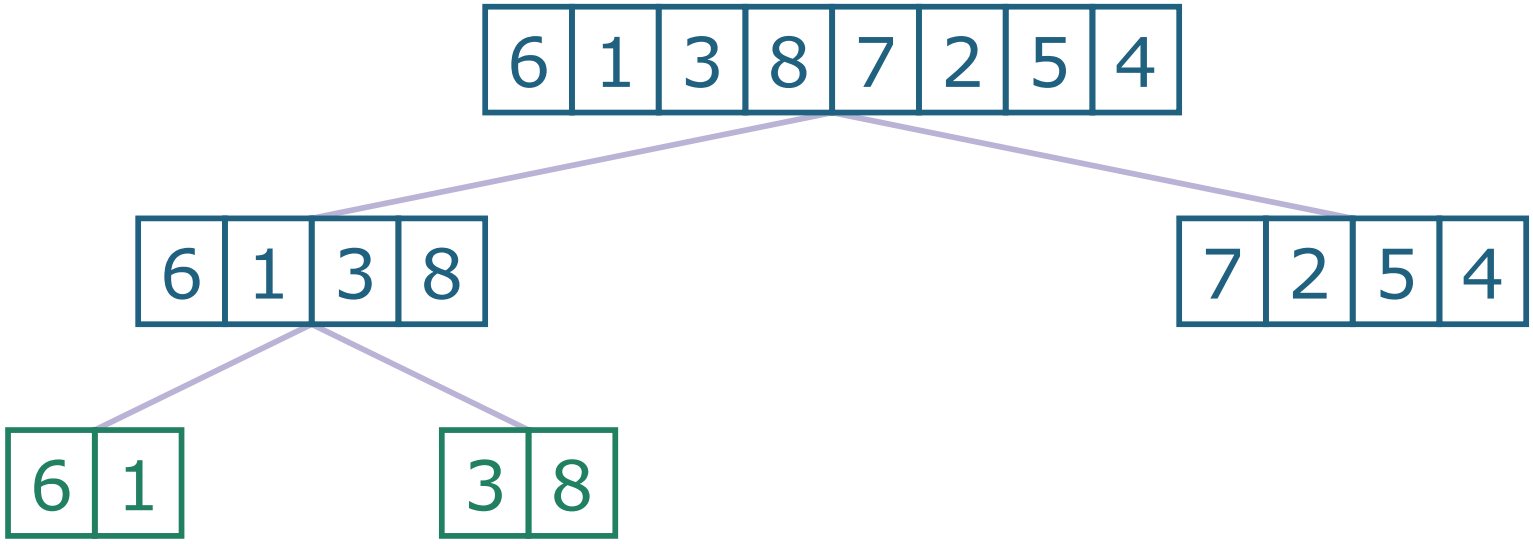
Dinâmica de Funcionamento

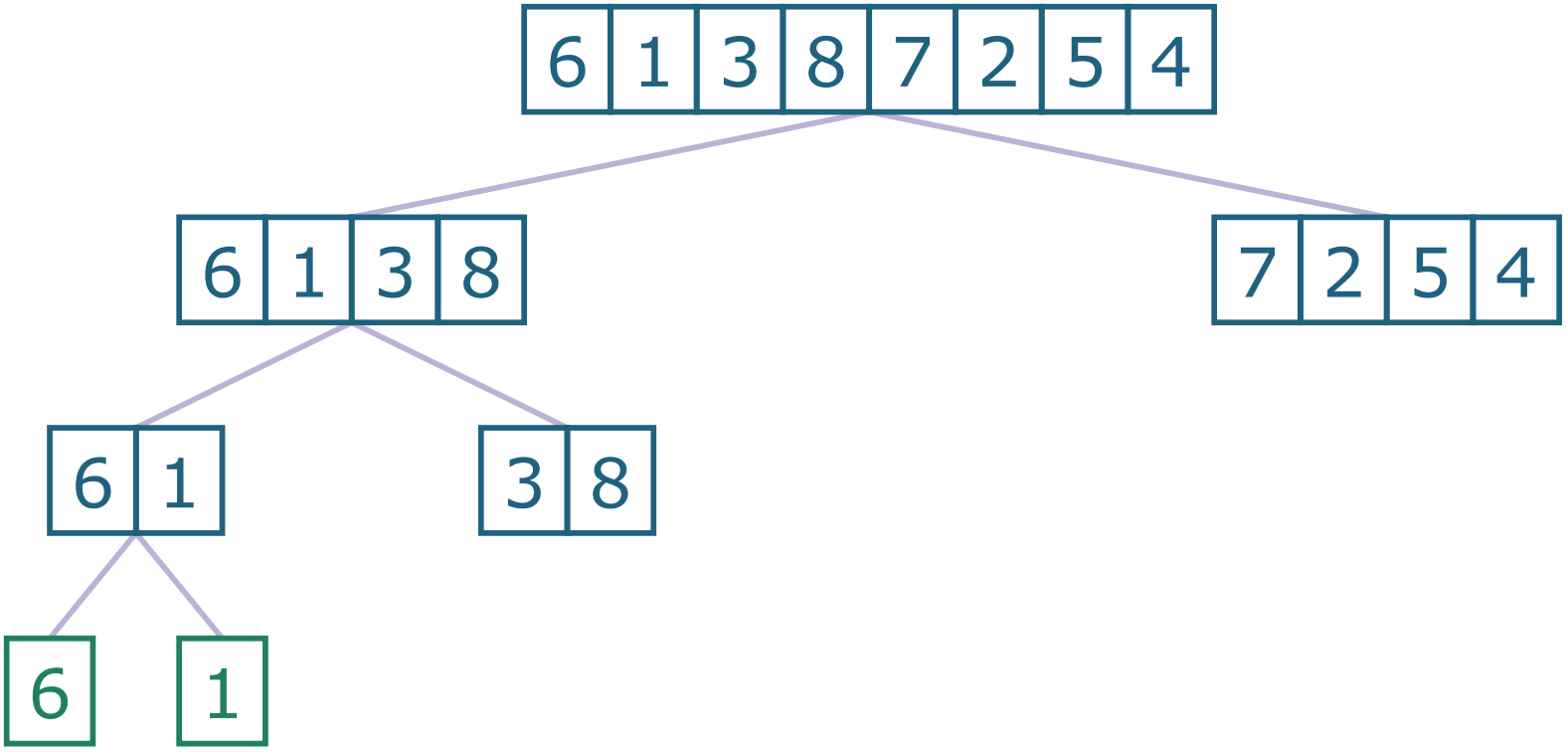
Considere a matriz unidimensional.

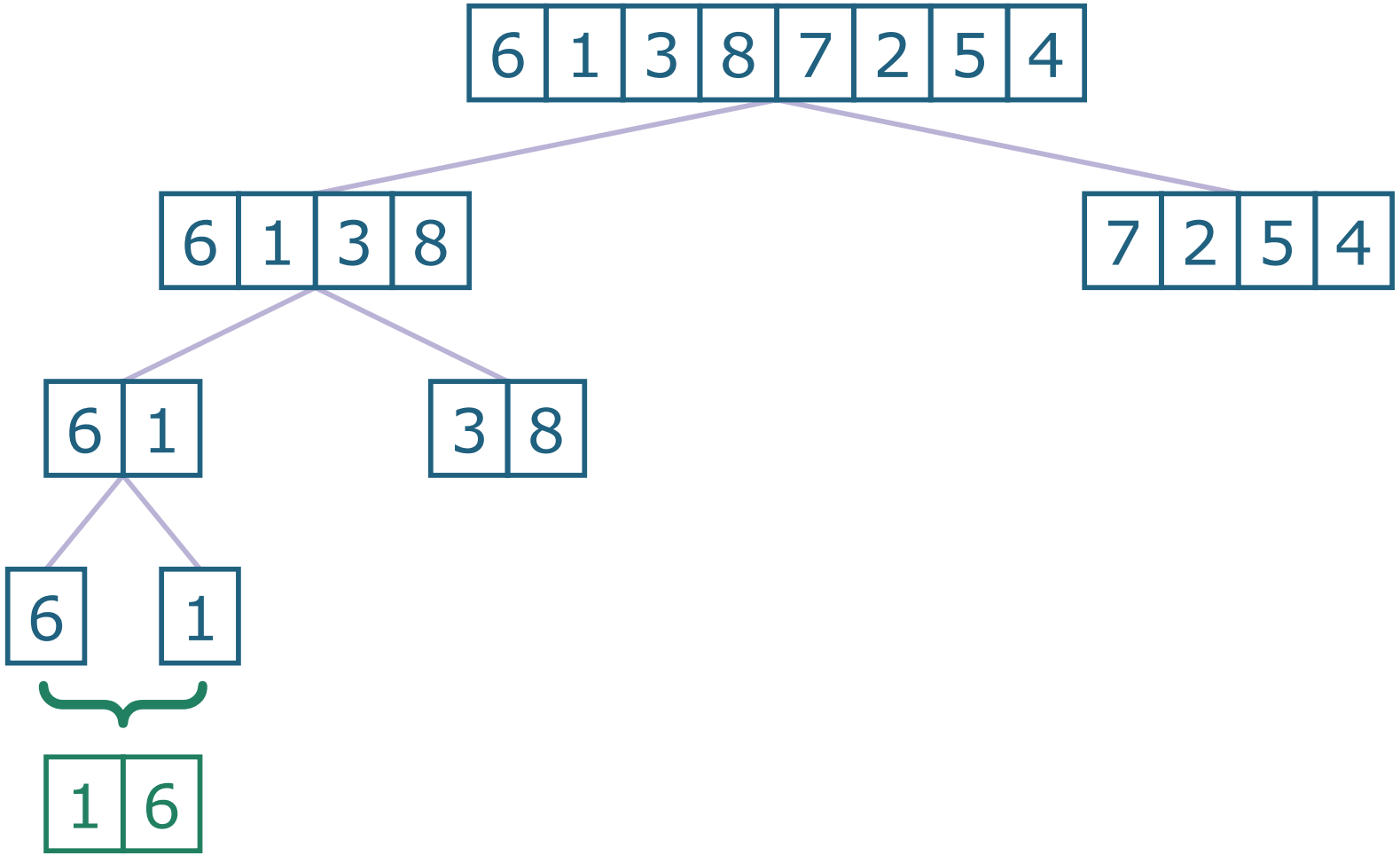


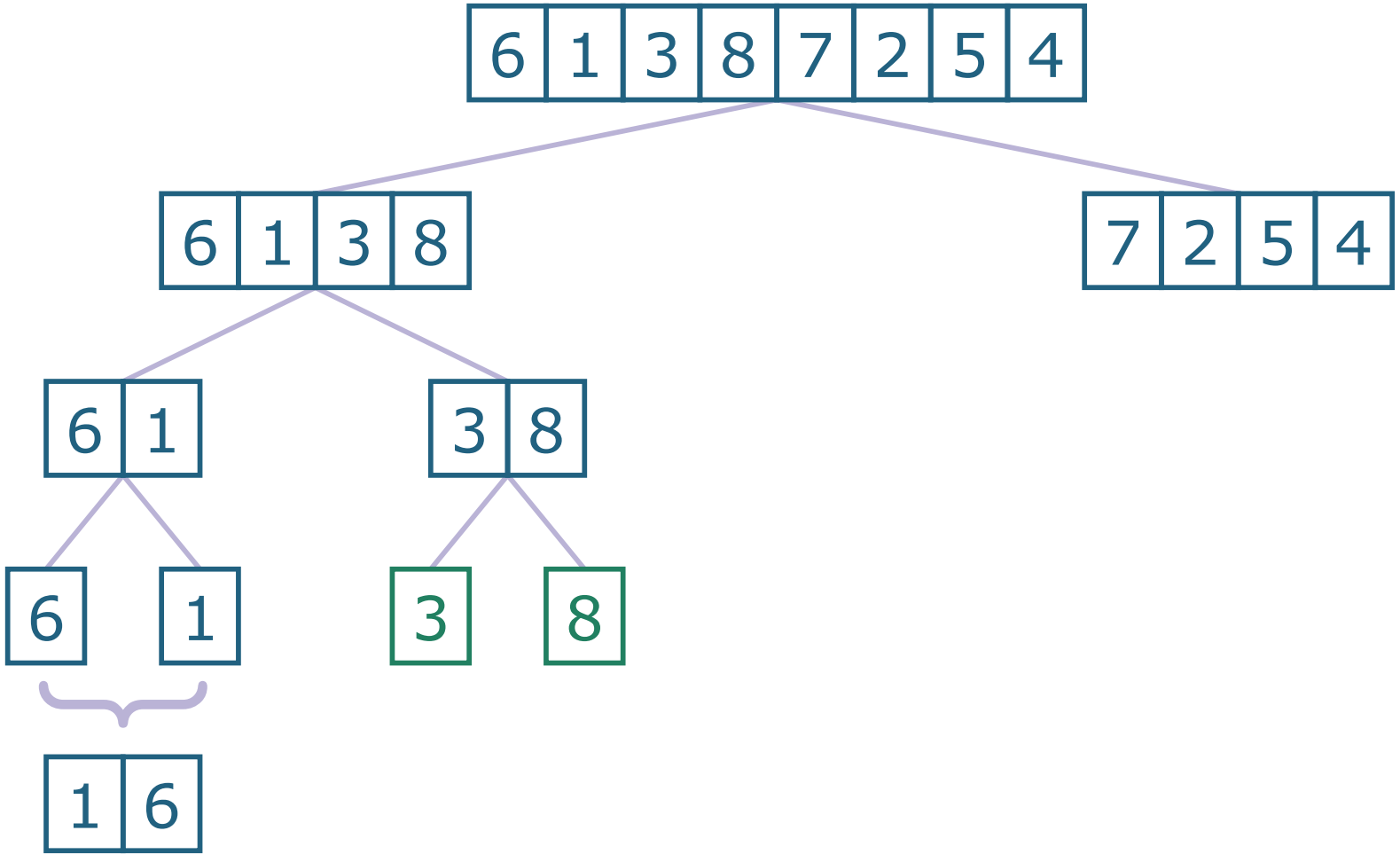

6 1 3 8 7 2 5 4

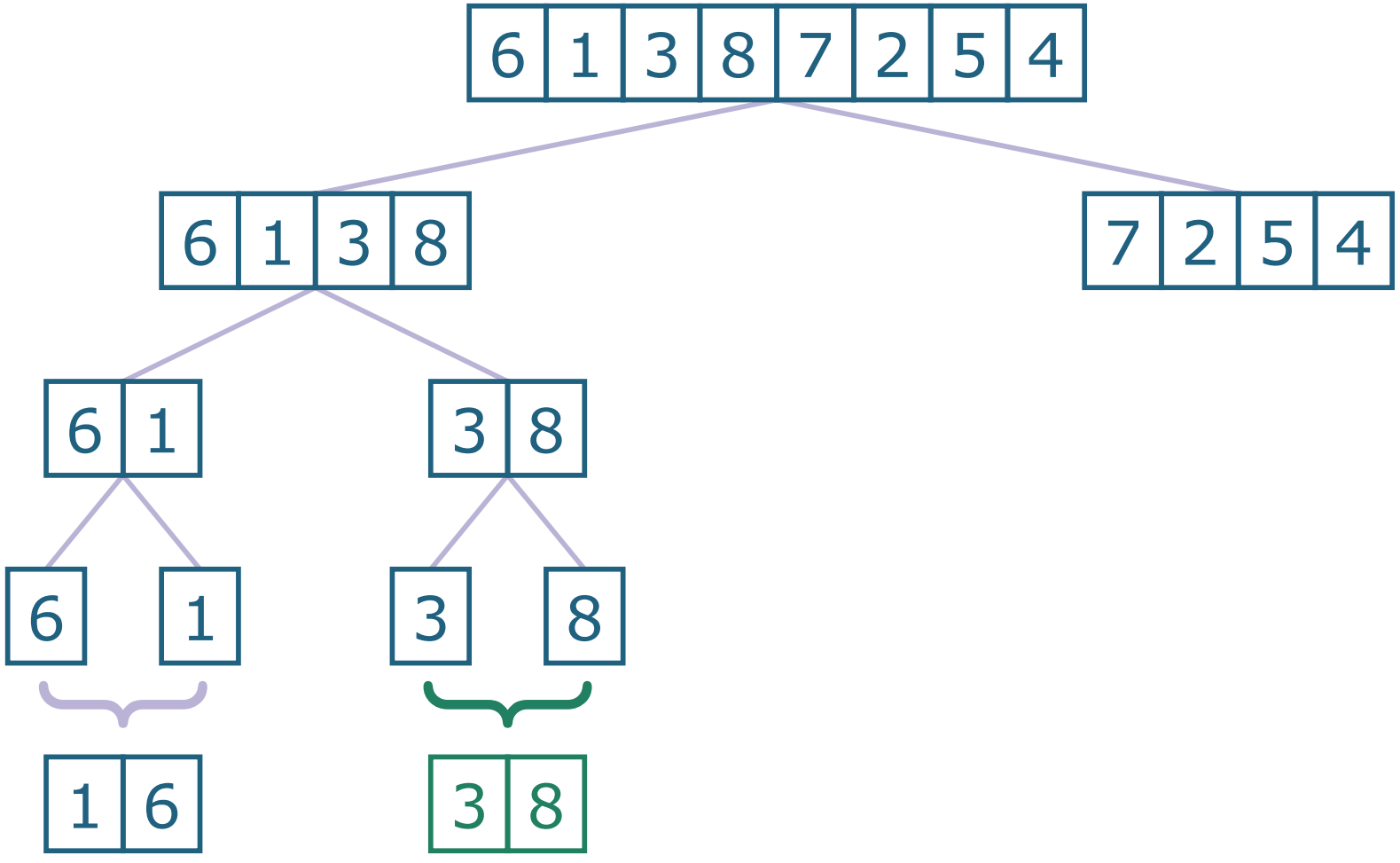

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4

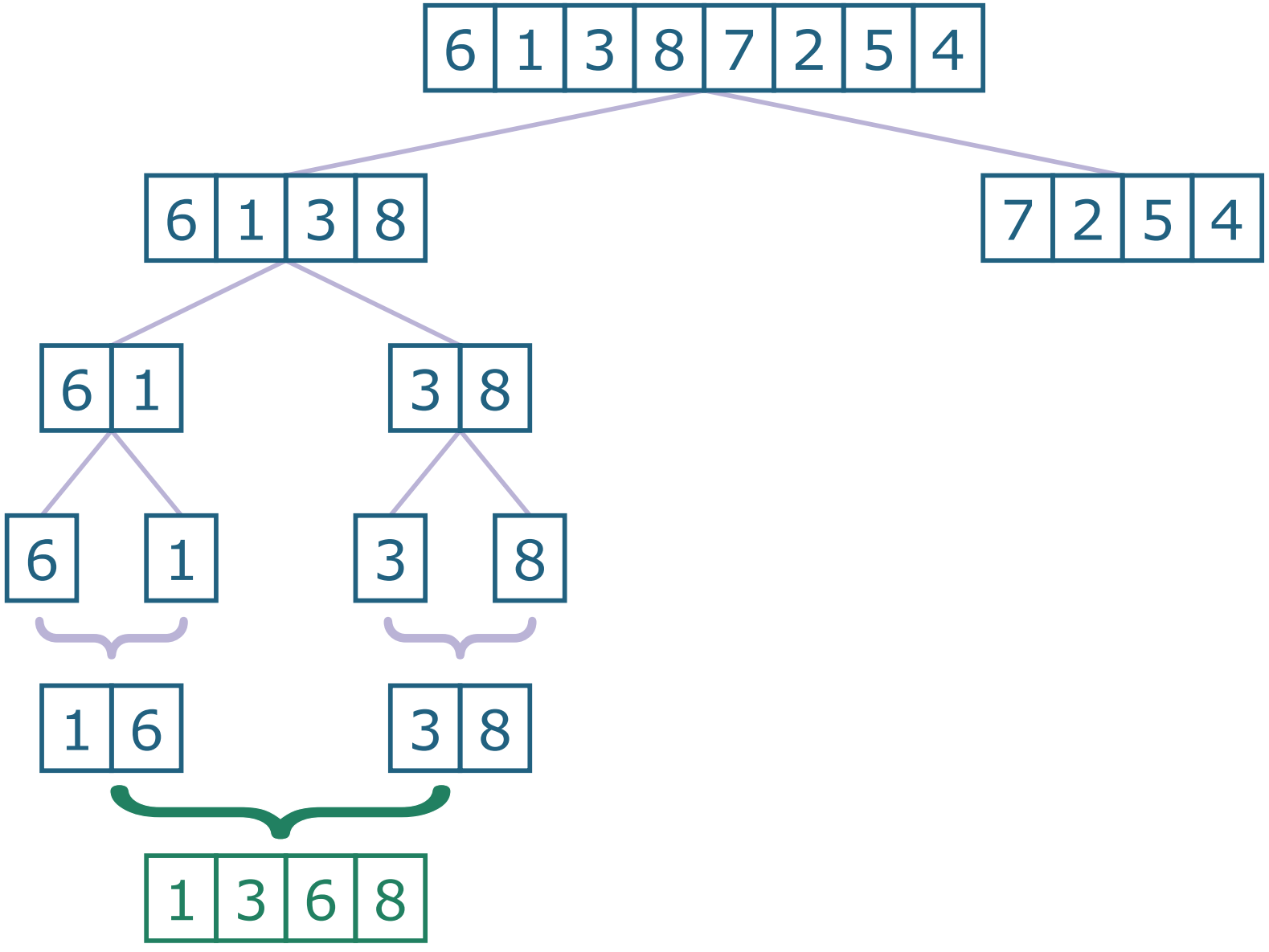

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8

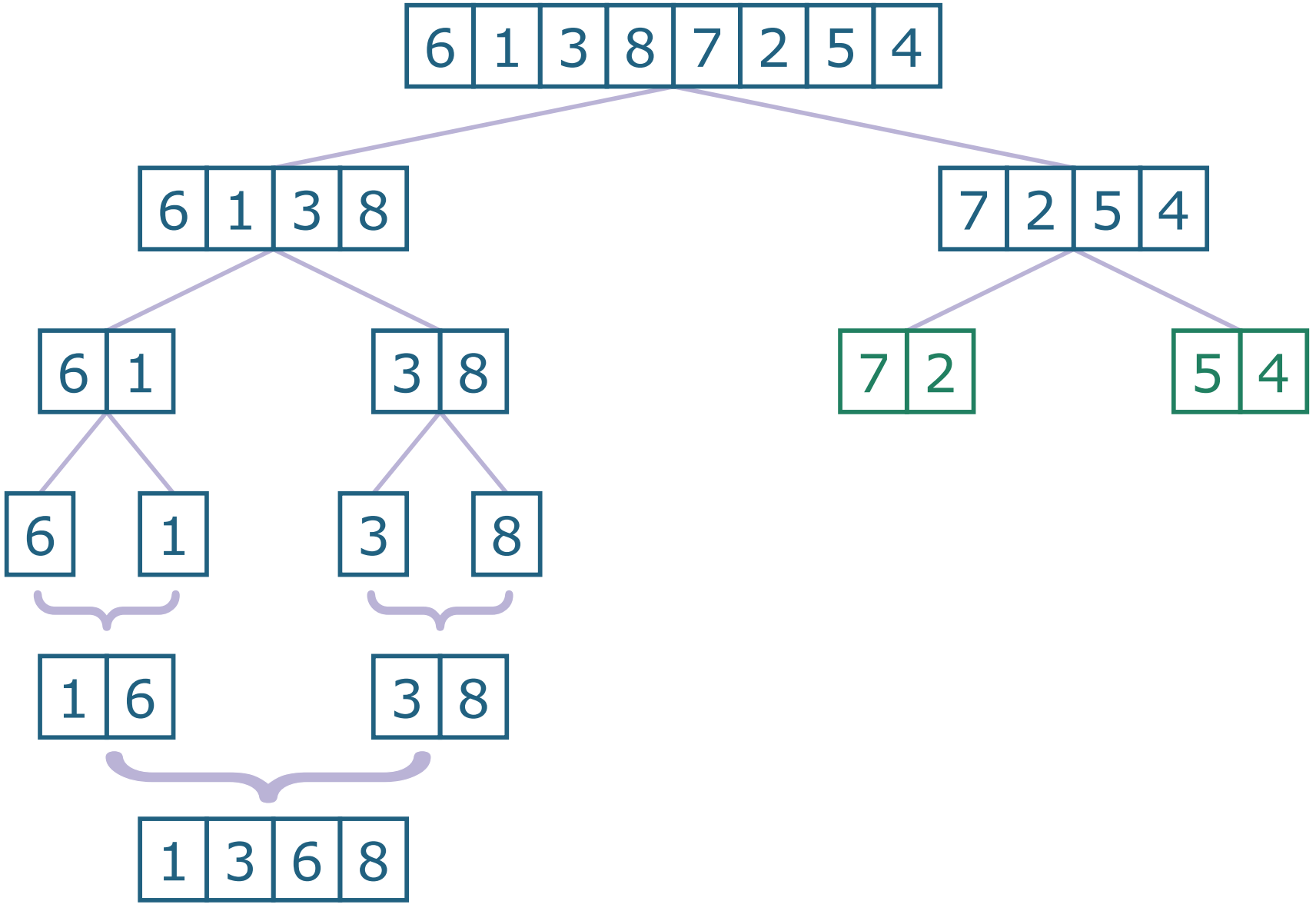

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
6
1

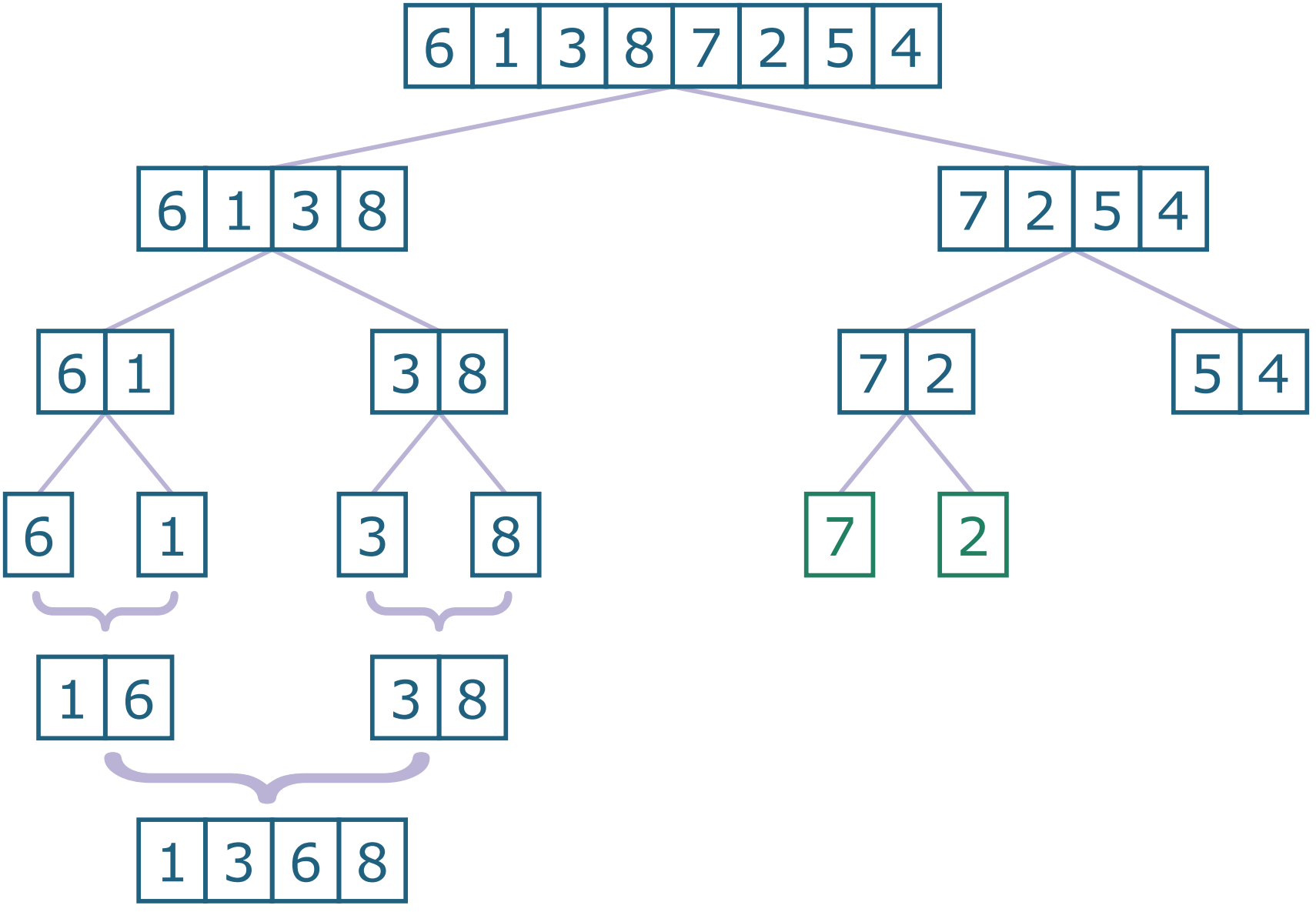

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
6
1
1 6

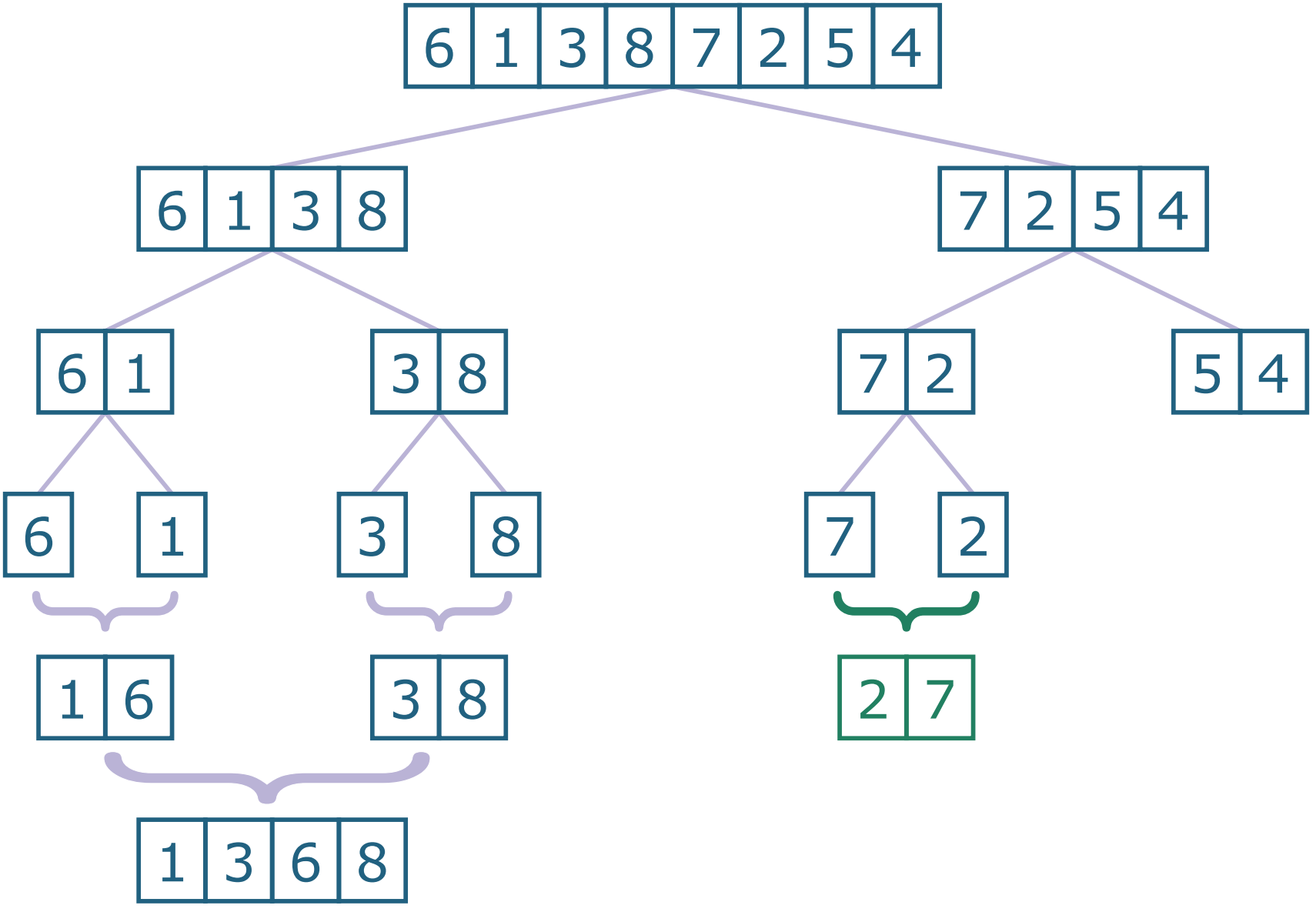

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
6
1
3
8
1 6

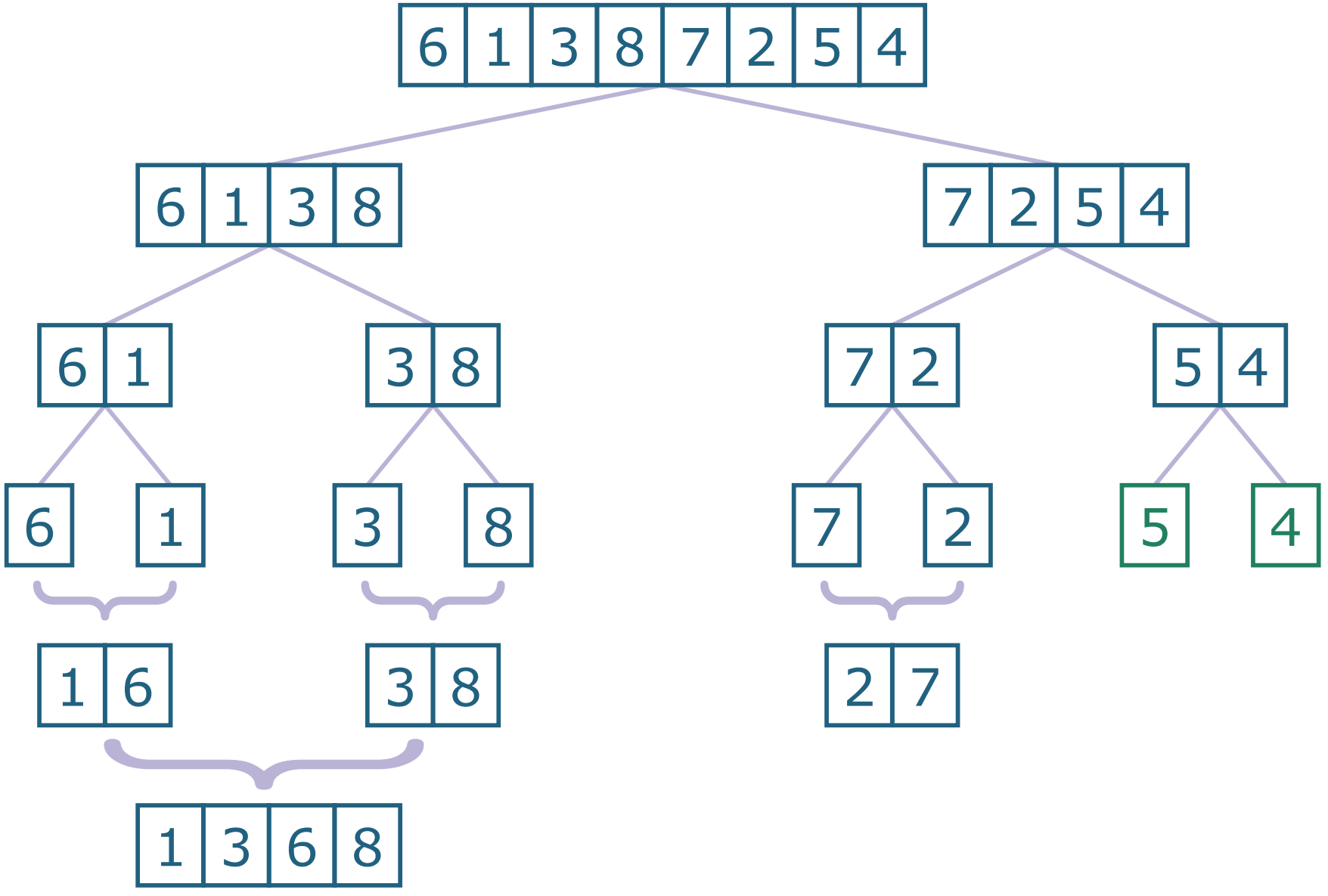

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
6
1
3
8
1 6
3 8

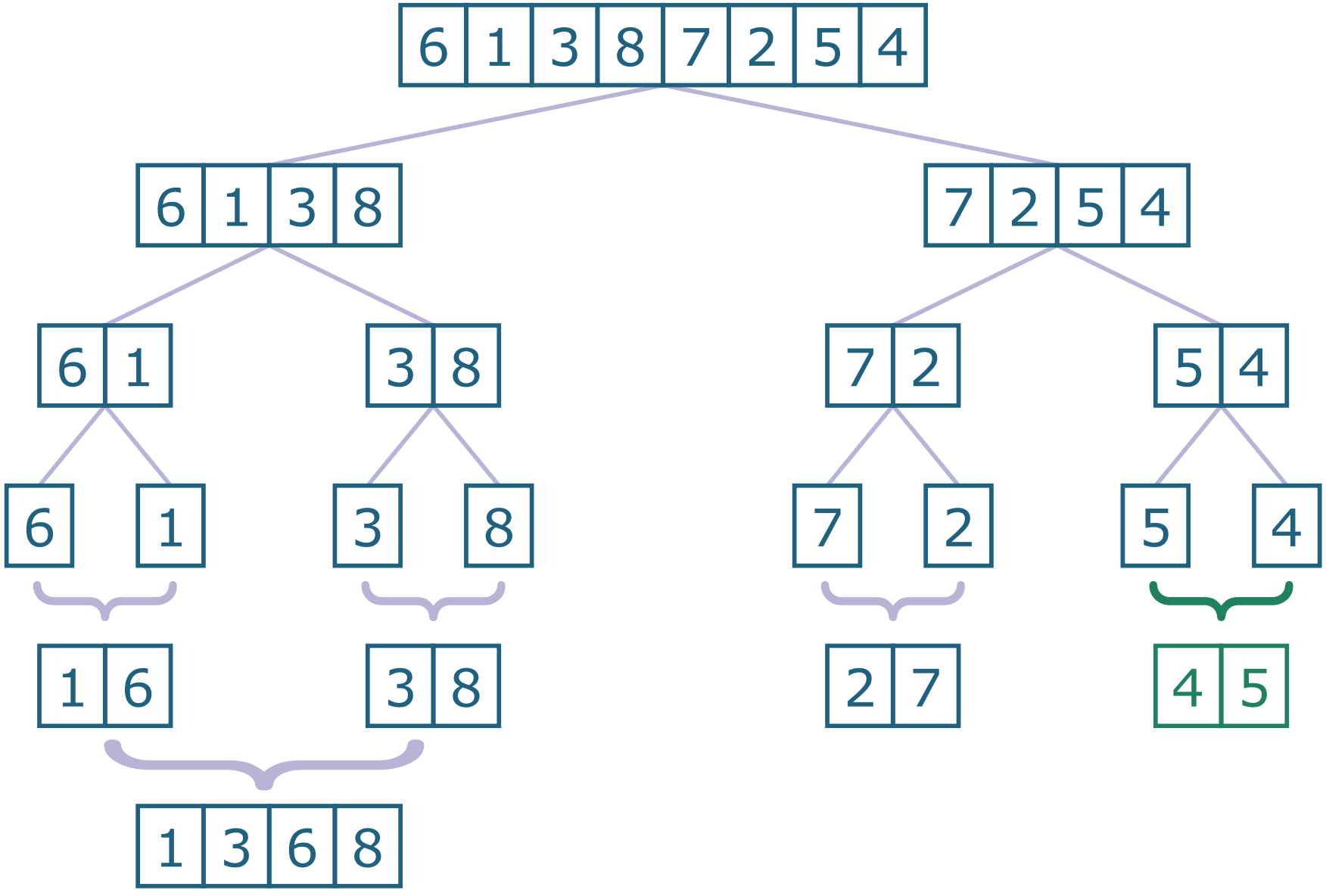

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
6
1
3
8
1 6
3 8
1 3 6 8

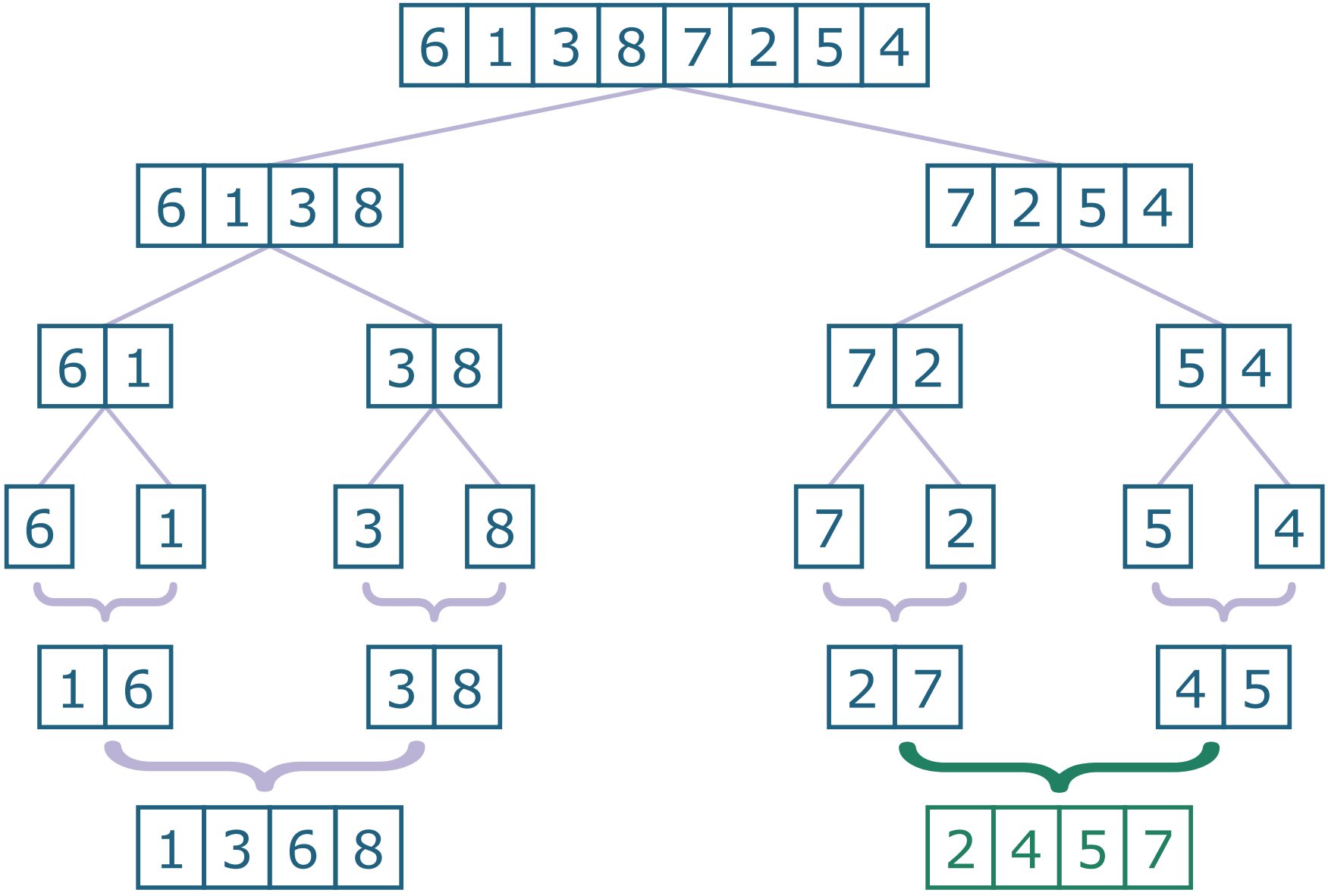

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
1 6
3 8
1 3 6 8

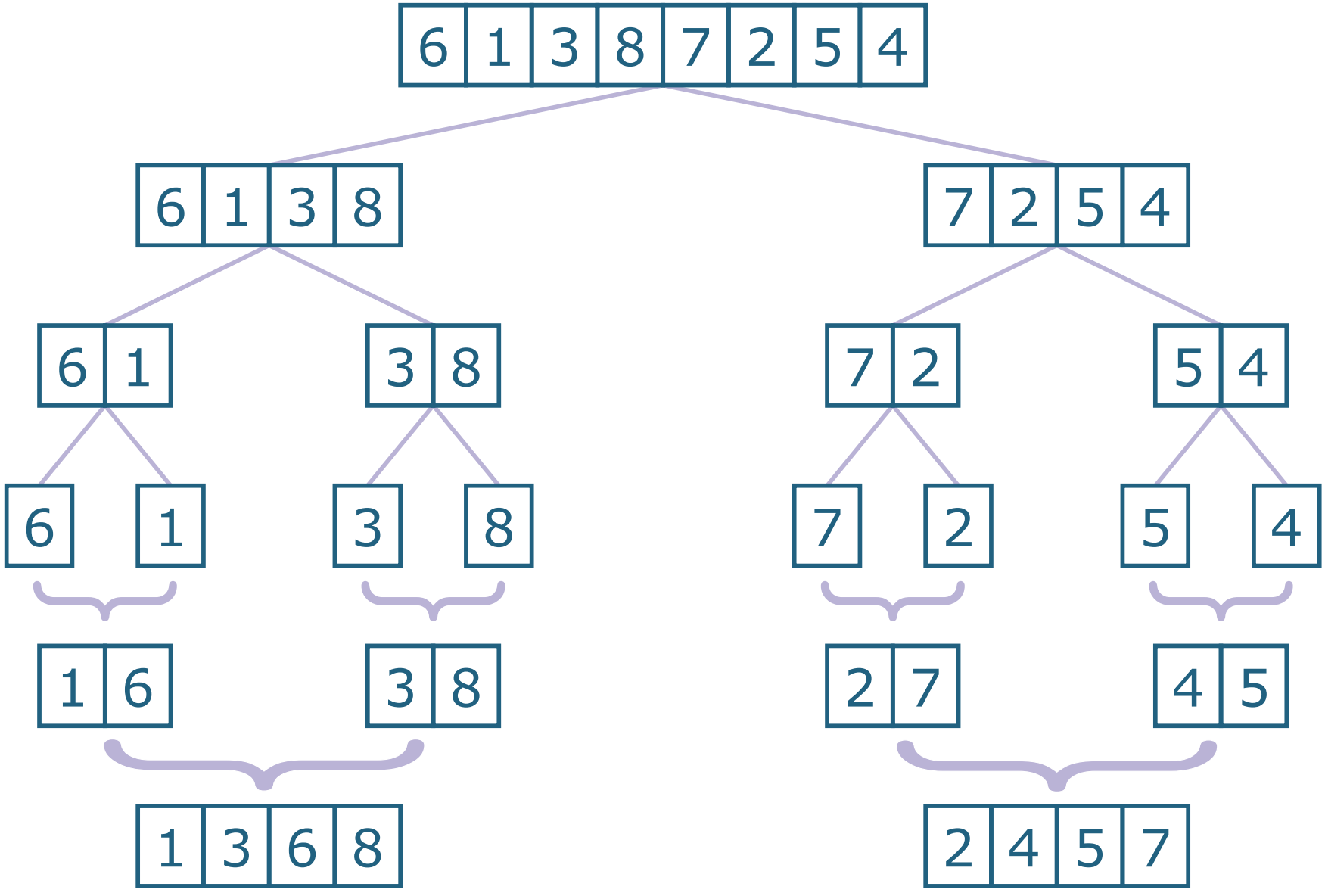

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
1 6
3 8
1 3 6 8

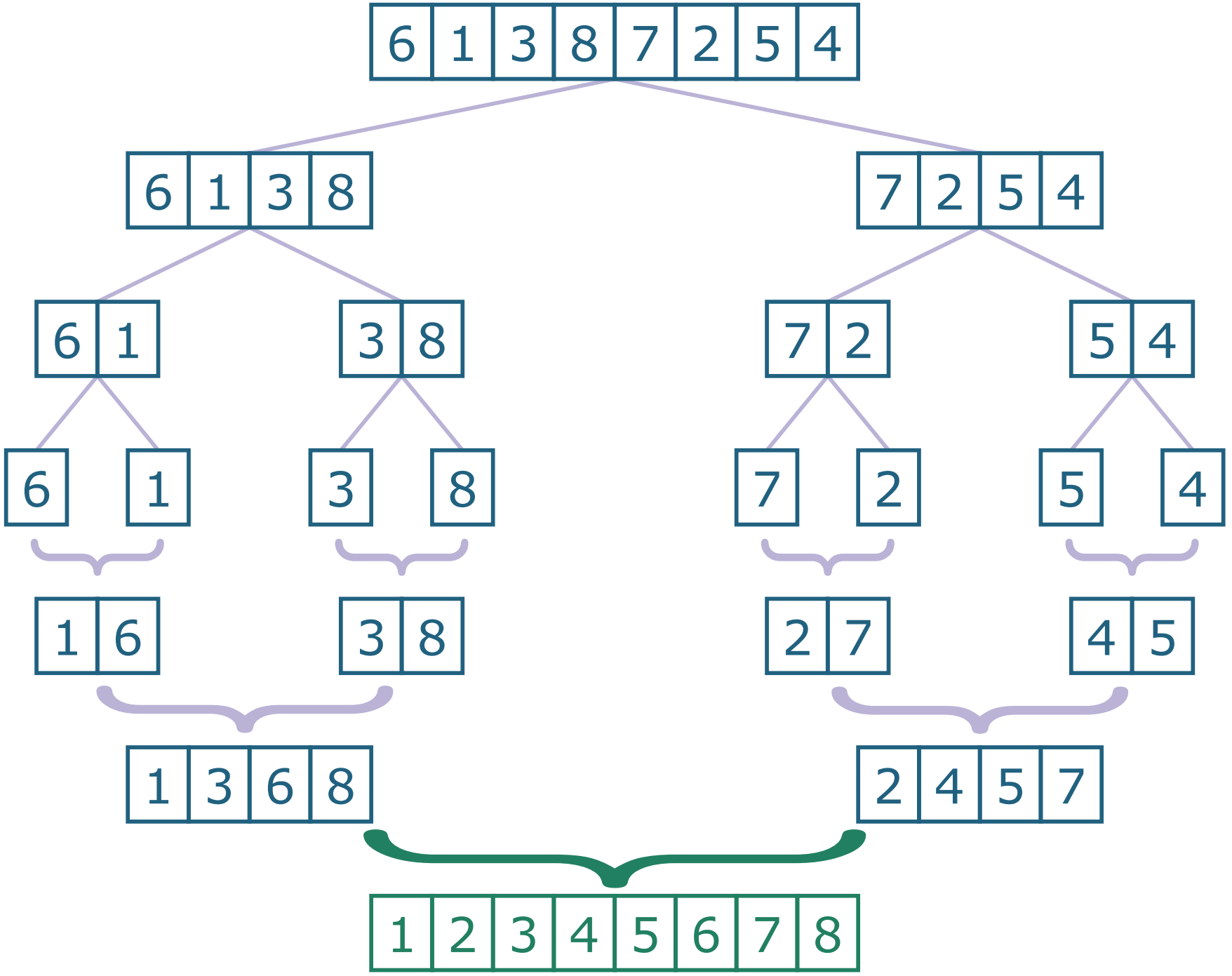

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
1 6
3 8
2 7
1 3 6 8

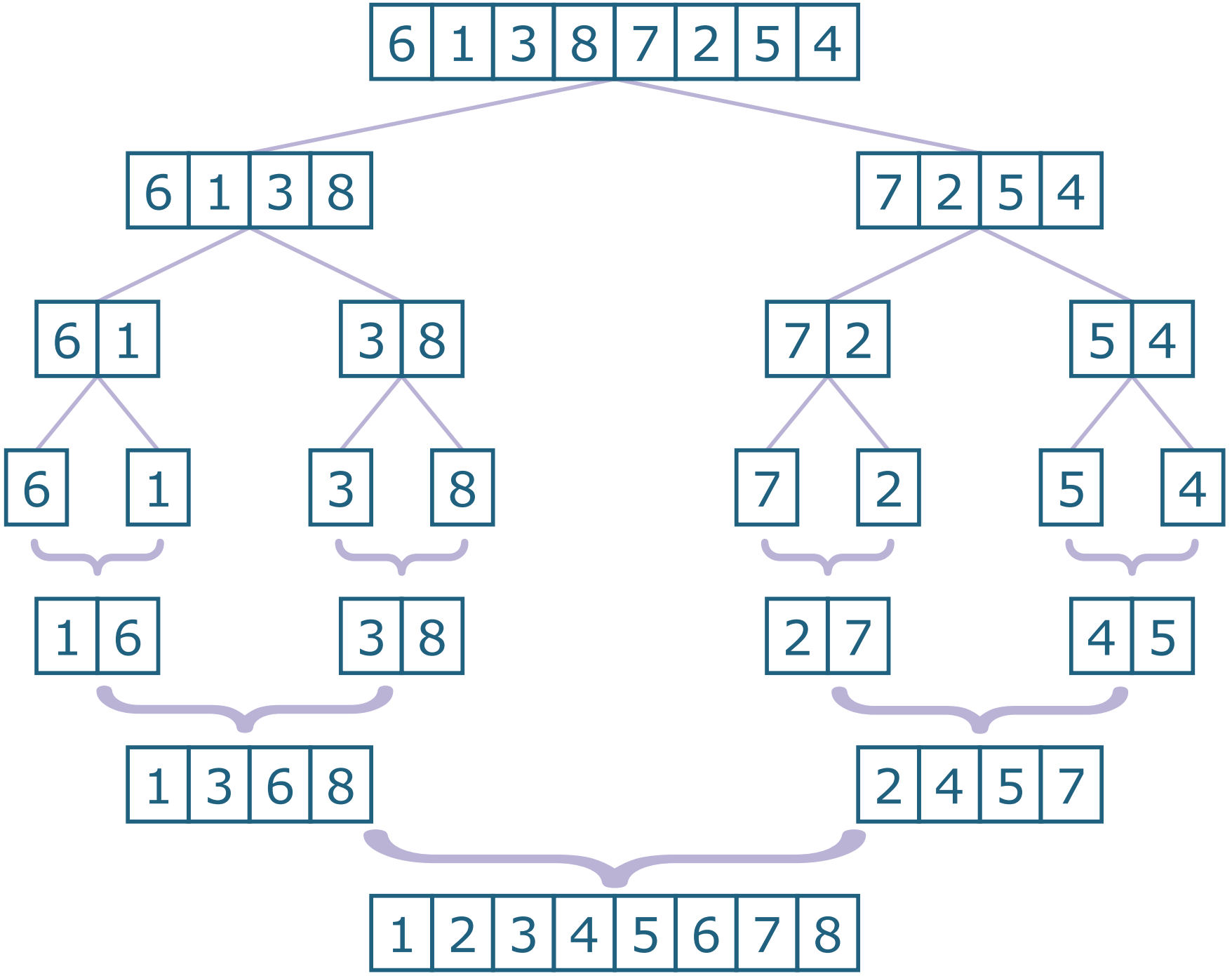

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
5
4
1 6
3 8
2 7
1 3 6 8


6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
5
4
1 6
3 8
2 7
4 5
1 3 6 8


6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
5
4
1 6
3 8
2 7
4 5
1 3 6 8
2 4 5 7

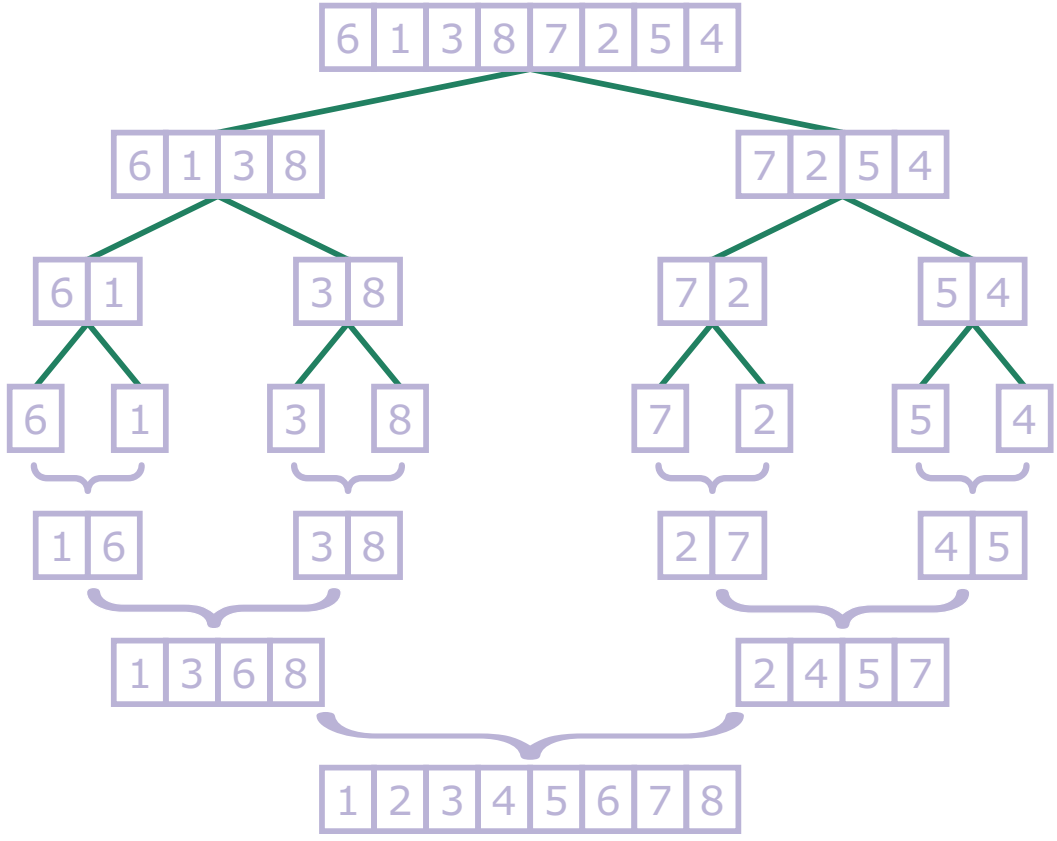

6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
5
4
1 6
3 8
2 7
4 5
1 3 6 8
2 4 5 7


6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
5
4
1 6
3 8
2 7
4 5
1 3 6 8
2 4 5 7
1 2 3 4 5 6 7 8


6 1 3 8 7 2 5 4
6 1 3 8
7 2 5 4
6 1
3 8
7 2
5 4
6
1
3
8
7
2
5
4
1 6
3 8
2 7
4 5
1 3 6 8
2 4 5 7
1 2 3 4 5 6 7 8

Dinâmica de Funcionamento

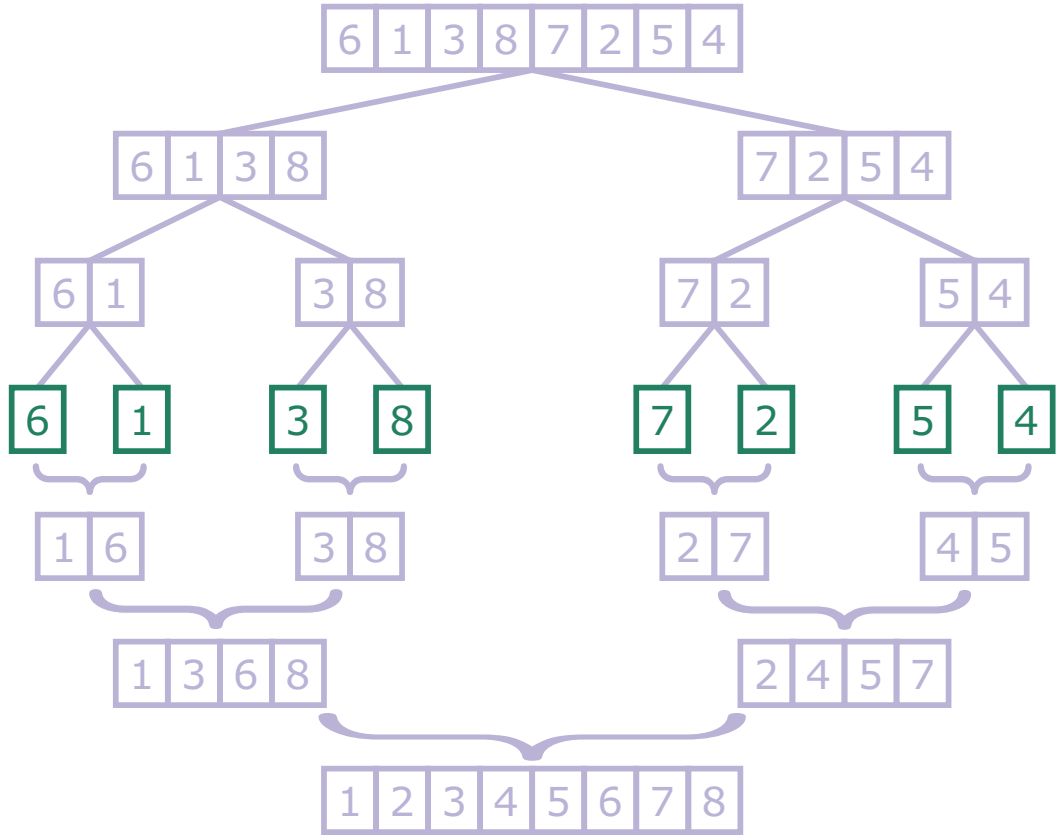
Divida a sequência de *n* elementos a serem ordenados em duas subsequências de *n/2* elementos cada.



Dinâmica de Funcionamento

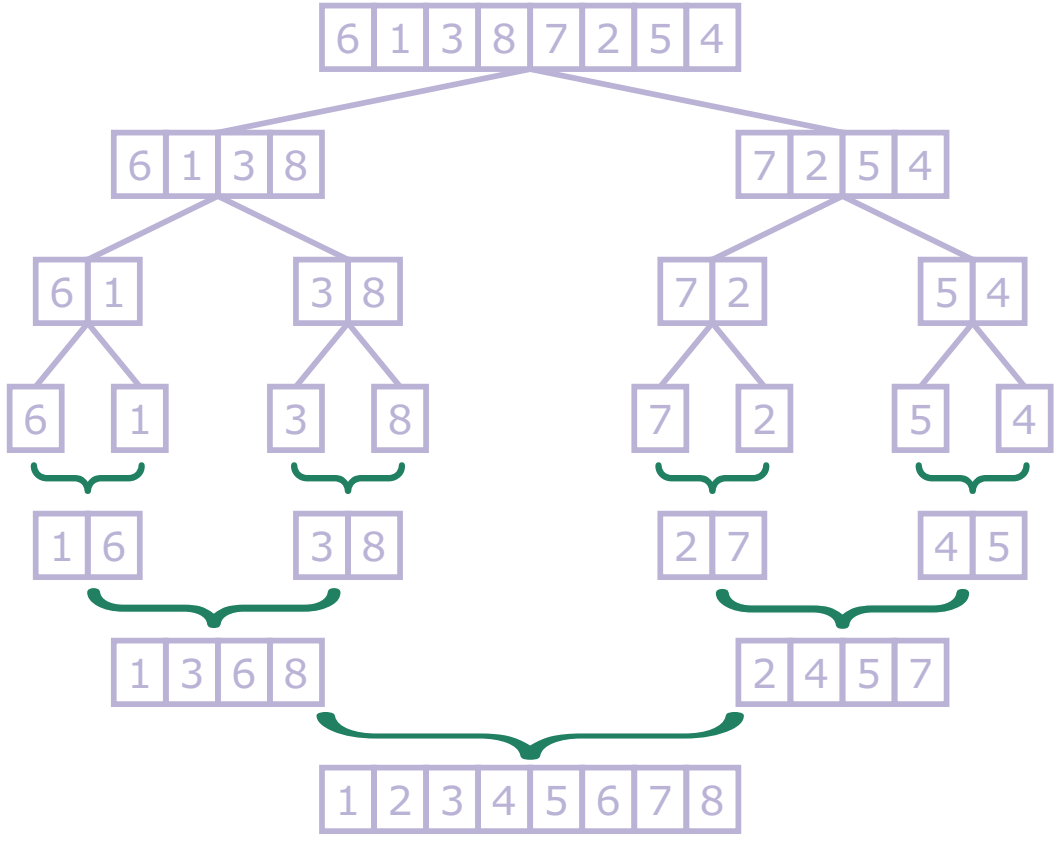
Conquiste ordenando as duas subsequências recursivamente.

Considere ordenada uma subsequência com um único elemento.



Dinâmica de Funcionamento

Combine, através da intercalação, as duas subsequências ordenadas pelo passo anterior.



Implementação

(**1**) Desenvolver um **intercalador de sequências**. Dados duas matrizes unidimensionais ordenadas e de mesmo tamanho, **A** e **B**, intercalá-las em uma matriz resultante **C**.

(**2**) Identificar características e dificuldades de implementação do método de ordenação completo.

(**3**) Pesquisar por implementações prontas e compreender os artifícios utilizados frente às dificuldades anteriormente apresentadas.

Implementação

Discussão sobre as características das implementações encontradas:

(**1**) Iterativas ou recursivas.

(**2**) Com alocação de memória estática ou dinâmica.

Merge Sort

Implementação Sugerida

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define TAM 20

// Solução proposta por Sedgewick.

void merge(int v[TAM], int inicio, int meio, int fim);
void mergesort(int v[TAM], int inicio, int fim);
int aux[TAM];
```

Merge Sort

```
int main()
{
    srand(time(NULL));

    printf("Merge Sort\n");
    printf("==========\n\n\n");

    int dados[TAM], i = 0;
    printf("Sequencia Gerada\n");
    printf("----------------\n");
    for (i = 0; i < TAM; i++)
    {
        dados[i] = rand() % 100;
        printf("%2d ", dados[i]);
    }

    // Método de ordenação
    mergesort(dados, 0, TAM-1);
```

Merge Sort

```
    printf("\n\n");
    printf("Sequencia Ordenada\n");
    printf("------------------\n");
    for (i = 0; i < TAM; i++)
    {
        printf("%2d ", dados[i]);
    }

    return(0);
}
```

Merge Sort

```
void mergesort(int v[TAM], int inicio, int fim)
{
    if (inicio < fim)
    {
        int meio = (inicio + fim) / 2;

        mergesort(v, inicio, meio);
        mergesort(v, meio+1, fim);

        merge(v, inicio, meio, fim);
    }
}
```

Merge Sort

```
// Recebe as sequencias crescentes v[inicio..meio] e v[meio+1..fim],
// rearranjando-as na sequencia v[inicio..fim] de modo que a ordem
// crescente se mantenha.

void merge(int v[TAM], int inicio, int meio, int fim)
{
    int i, j, k;
    for (i = meio+1; i > inicio; i--)
       aux[i-1] = v[i-1];

    for (j = meio; j < fim; j++)
       aux[fim + meio-j] = v[j+1];

    // agora i == inicio e j == fim

    for (k = inicio; k <= fim; k++)
       if (aux[j] < aux[i])
          v[k] = aux[j--];
       else
          v[k] = aux[i++];
}
```