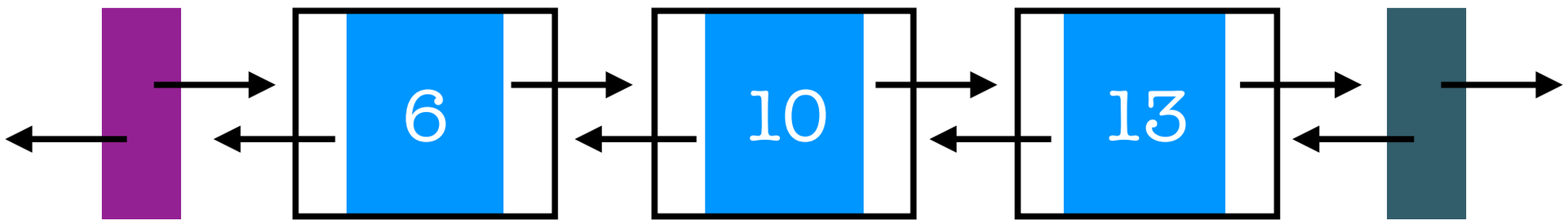
listas **duplamente** encadeadas


6
10
13

linkedList.h

```
template <typename T>
class LinkedList {
    private:
        struct Node {
            public:
                T chave;
                Node * next, * prev;
                Node () = default;
                Node (T _chave) :
                        chave(_chave),
                        next(nullptr),
                        prev(nullptr) {}
        };
        Node * inicio, * fim;
        unsigned tamanho;
    public:
...
```

```
    public:
        LinkedList (void) {
            fim = new Node;
            inicio = new Node;
            inicio->next = fim;
            fim->prev = inicio;
        }
        unsigned size (void) {
            return tamanho;
        }
        void print (void) {
            Node * itr = inicio->next;
            while (itr != fim) {
                std::cout << itr->chave << " ";
                itr = itr->next;
            }
            std::cout << std::endl;
        }
};
```

linkedList.h

```
template <typename T>
class LinkedList {
    private:
        struct Node {
            public:
                T chave;
                Node * next, * prev;
                Node () = default;
                Node (T _chave) :
                        chave(_chave),
                        next(nullptr),
                        prev(nullptr) {}
        };
        Node * inicio, * fim;
        unsigned tamanho;
    public:
...
```

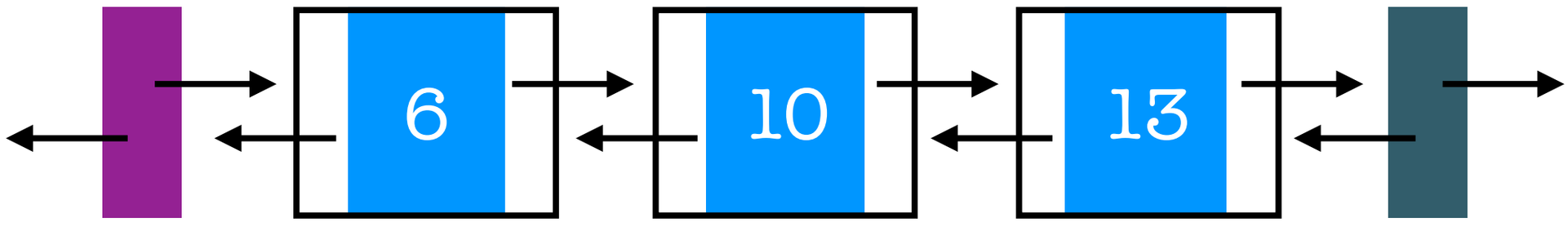
```
public:
        ~LinkedList (void) {
            Node  * itr = inicio->next;
            while (itr != fim) {
                itr = itr->next;
                delete itr->prev;
            }
            delete inicio;
            delete  fim;
        }
};
```

Operação	Vetor	Lista Encadeada
acesso	O(1)	
busca	O(n)	
tamanho	O(1)	
inserção	O(n)	
remoção	O(n)	
particionar	O(n)	
duplicar	O(n)	
ordenar	O(nlogn)	

acesso

linkedList.h

```
template <typename T>
T LinkedList<T>::get(size_t idx) {
    if (idx > tamanho) abort();
    Node * itr = inicio->next;
    for (int i = 1; i < idx; i++)
        itr = itr->next;
    return itr->chave;
}
```

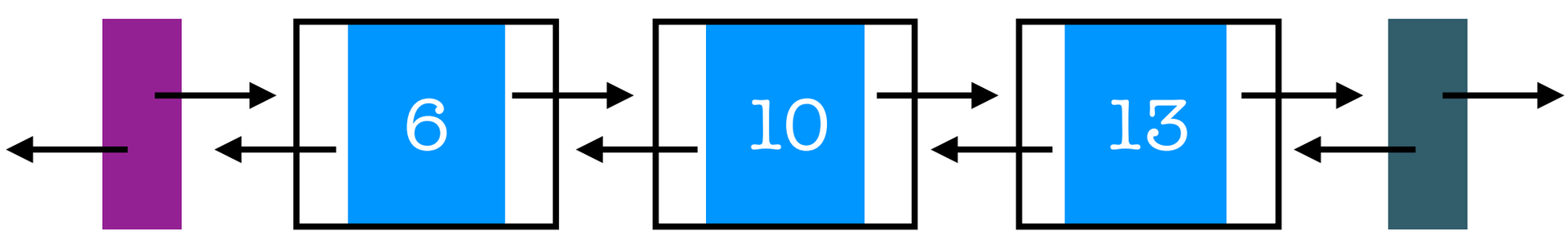


Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	
tamanho	O(1)	
inserção	O(n)	
remoção	O(n)	
particionar	O(n)	
duplicar	O(n)	
ordenar	O(nlogn)	

busca

linkedList.h

```
template <typename T>
typename LinkedList<T>::Node * LinkedList<T>::find(T chave) {
    Node * itr = inicio->next;
    while (itr != fim && itr->chave != chave)
        itr = itr->next;
    return itr;
}
```



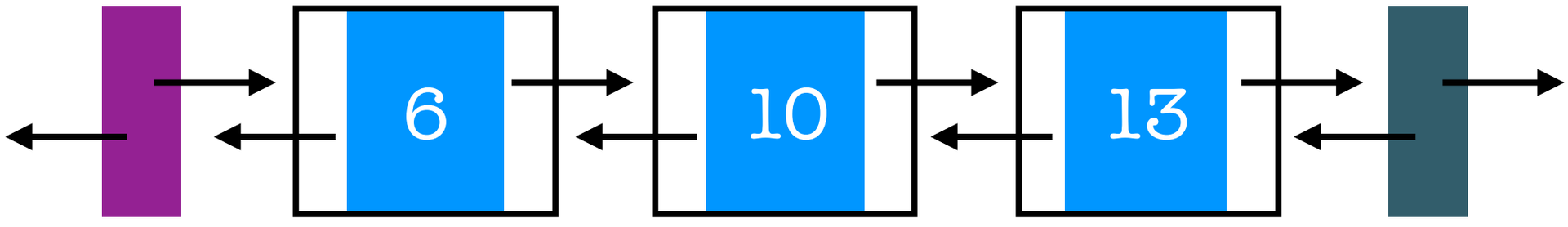
Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	O(n)
tamanho	O(1)	
inserção	O(n)	
remoção	O(n)	
particionar	O(n)	
duplicar	O(n)	
ordenar	O(nlogn)	

Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	O(n)
tamanho	O(1)	O(1)
inserção	O(n)	
remoção	O(n)	
particionar	O(n)	
duplicar	O(n)	
ordenar	O(nlogn)	

inserção

linkedList.h

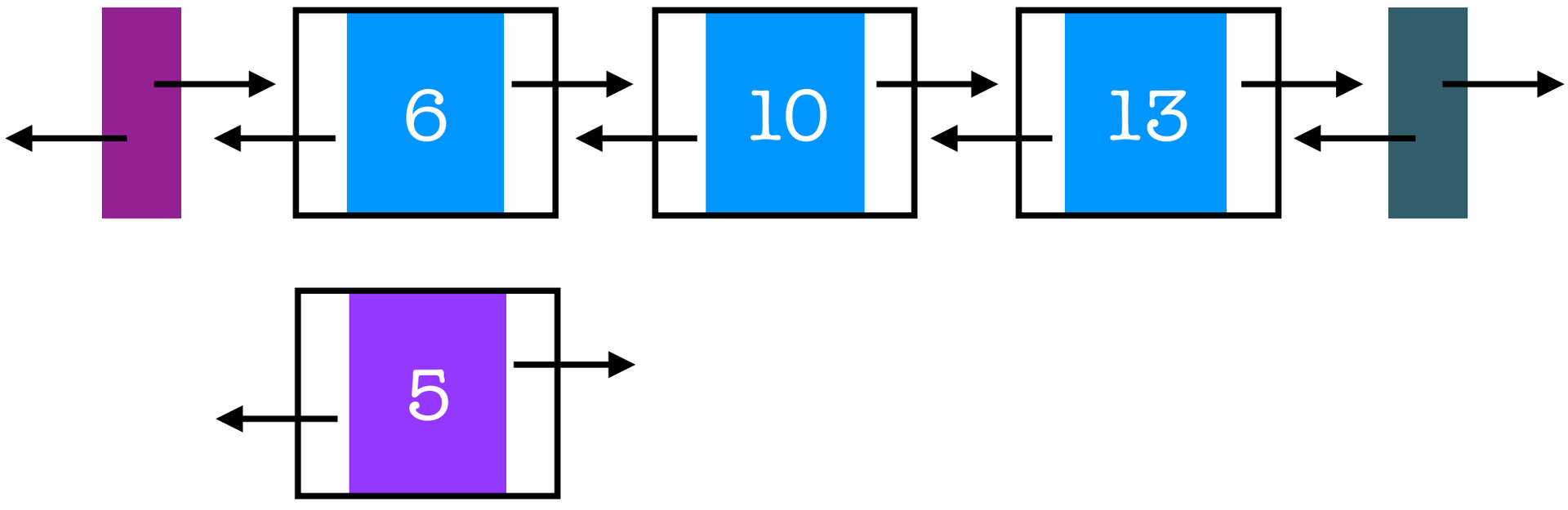
```
template <typename T>
void LinkedList<T>::insert(LinkedList<T>::Node * pos, T chave) {
    Node * novo = new Node (chave);
    Node * anterior = pos->prev;
    anterior->next = novo;
    pos->prev = novo;
    novo->next = pos;
    novo->prev = anterior;
    tamanho++;
}
```



inserção

linkedList.h

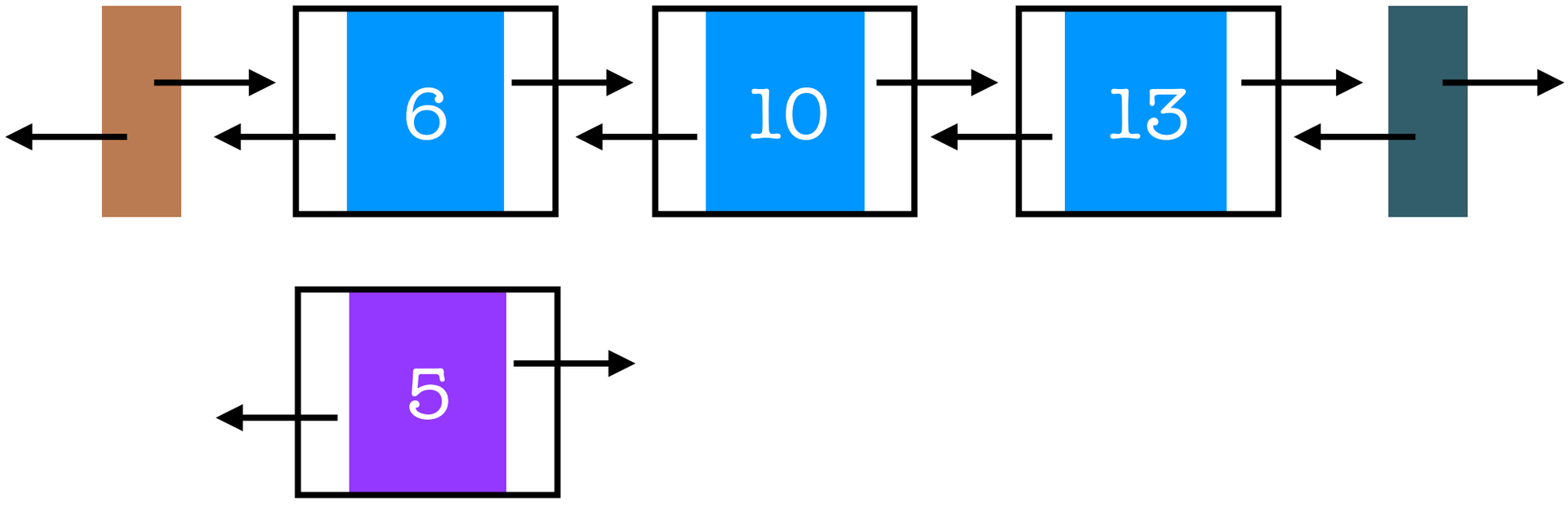
```
template <typename T>
void LinkedList<T>::insert(LinkedList<T>::Node * pos, T chave) {
    Node * novo = new Node (chave);
    Node * anterior = pos->prev;
    anterior->next = novo;
    pos->prev = novo;
    novo->next = pos;
    novo->prev = anterior;
    tamanho++;
}
```



inserção

linkedList.h

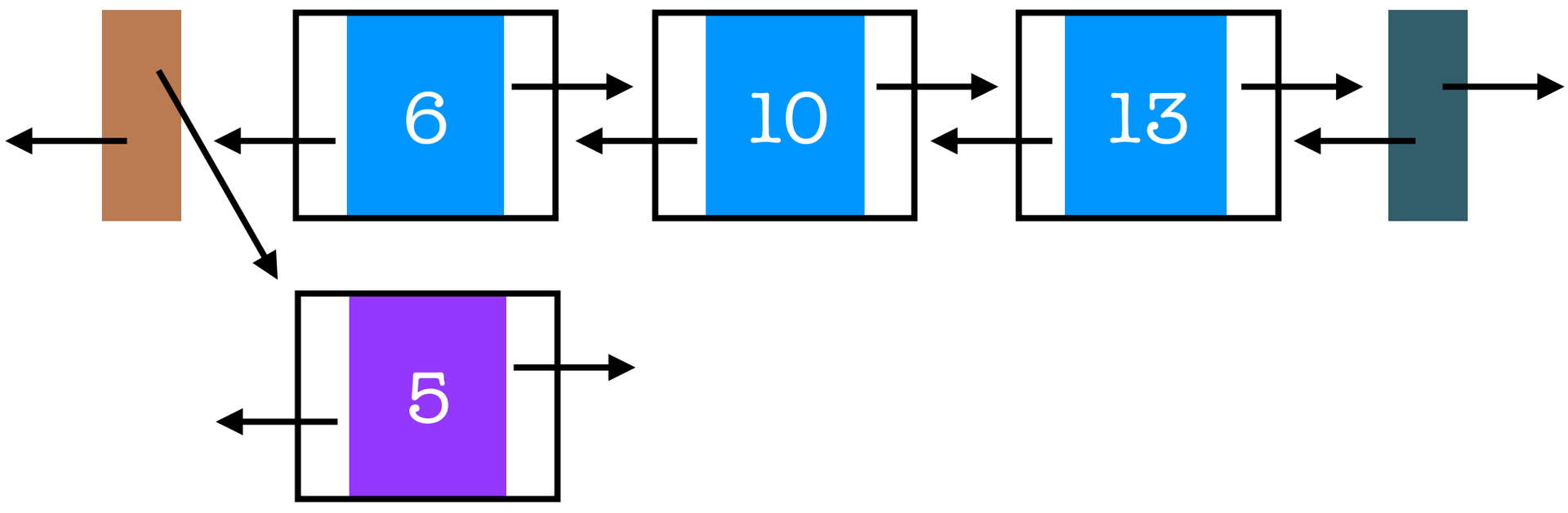
```
template <typename T>
void LinkedList<T>::insert(LinkedList<T>::Node * pos, T chave) {
    Node * novo = new Node (chave);
    Node * anterior = pos->prev;
    anterior->next = novo;
    pos->prev = novo;
    novo->next = pos;
    novo->prev = anterior;
    tamanho++;
}
```



inserção

linkedList.h

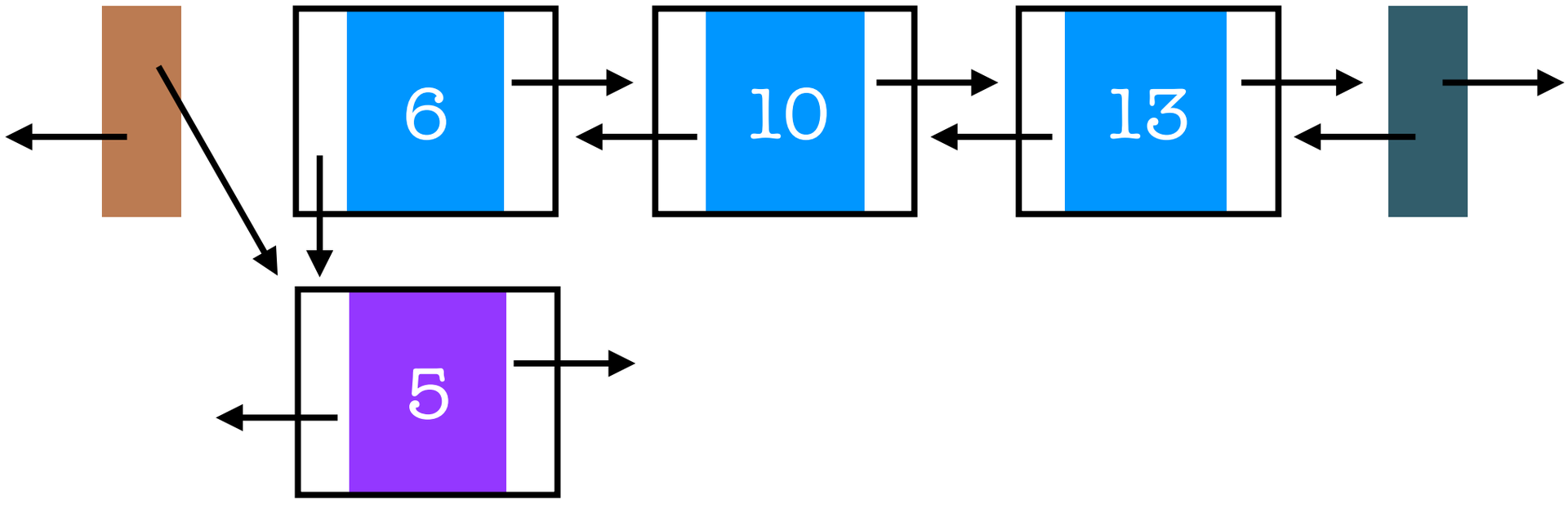
```
template <typename T>
void LinkedList<T>::insert(LinkedList<T>::Node * pos, T chave) {
    Node * novo = new Node (chave);
    Node * anterior = pos->prev;
    anterior->next = novo;
    pos->prev = novo;
    novo->next = pos;
    novo->prev = anterior;
    tamanho++;
}
```



inserção

linkedList.h

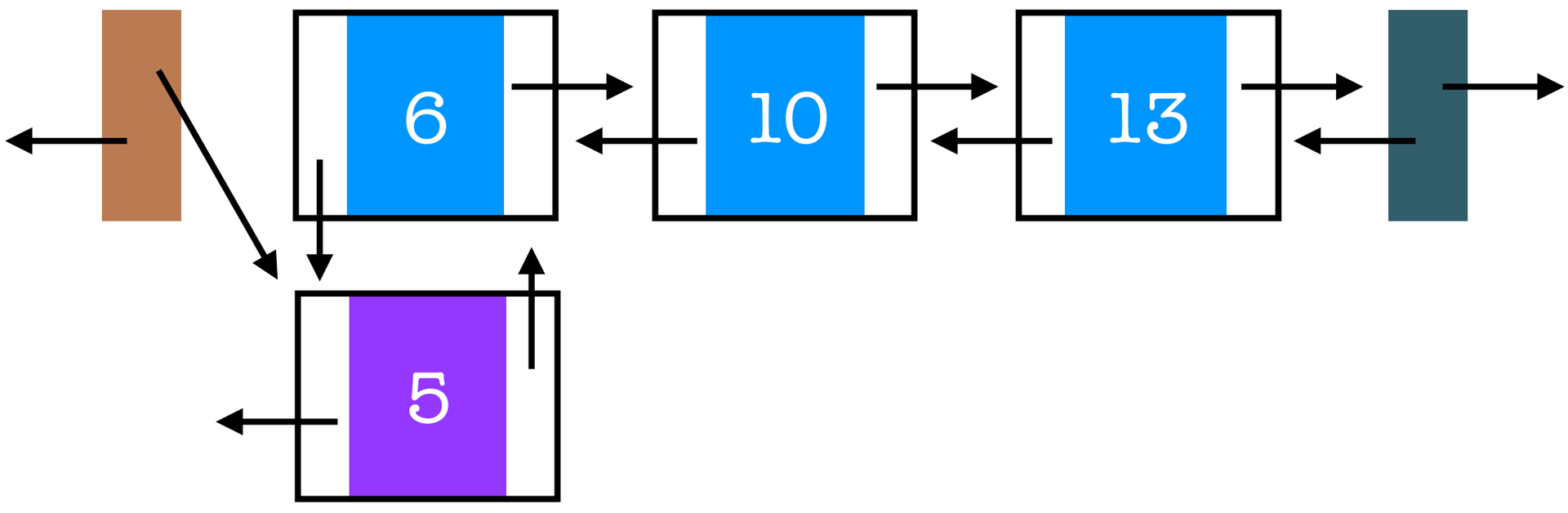
```
template <typename T>
void LinkedList<T>::insert(LinkedList<T>::Node * pos, T chave) {
    Node * novo = new Node (chave);
    Node * anterior = pos->prev;
    anterior->next = novo;
    pos->prev = novo;
    novo->next = pos;
    novo->prev = anterior;
    tamanho++;
}
```



inserção

linkedList.h

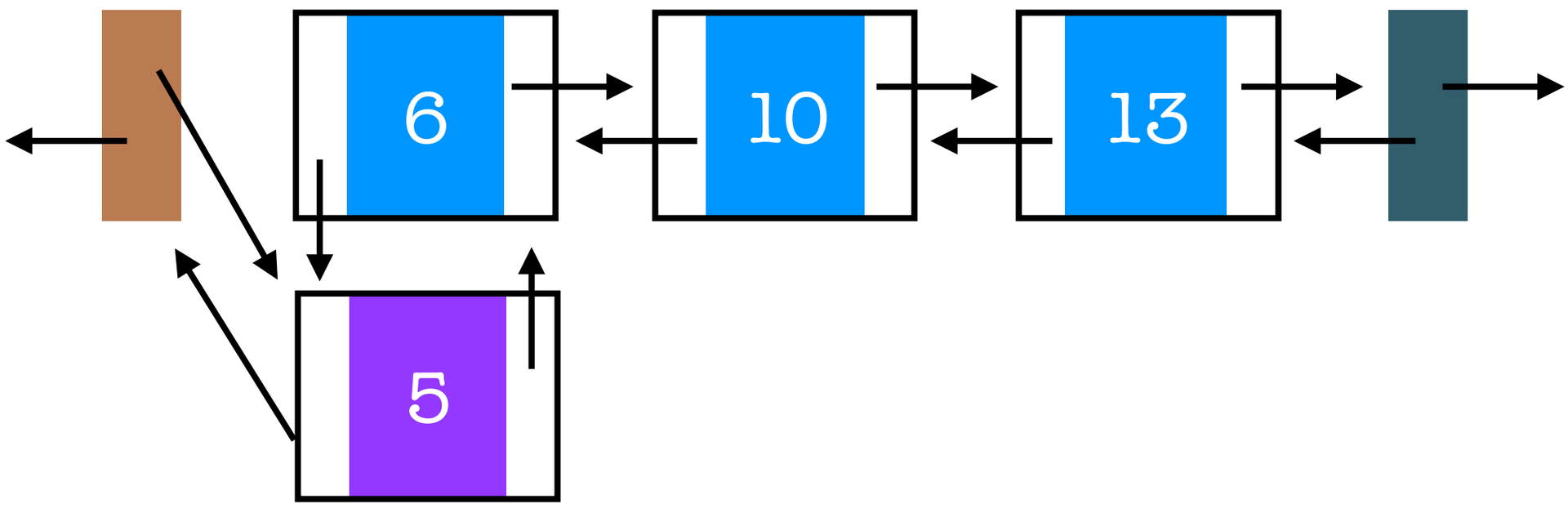
```
template <typename T>
void LinkedList<T>::insert(LinkedList<T>::Node * pos, T chave) {
    Node * novo = new Node (chave);
    Node * anterior = pos->prev;
    anterior->next = novo;
    pos->prev = novo;
    novo->next = pos;
    novo->prev = anterior;
    tamanho++;
}
```



inserção

linkedList.h

```
template <typename T>
void LinkedList<T>::insert(LinkedList<T>::Node * pos, T chave) {
    Node * novo = new Node (chave);
    Node * anterior = pos->prev;
    anterior->next = novo;
    pos->prev = novo;
    novo->next = pos;
    novo->prev = anterior;
    tamanho++;
}
```

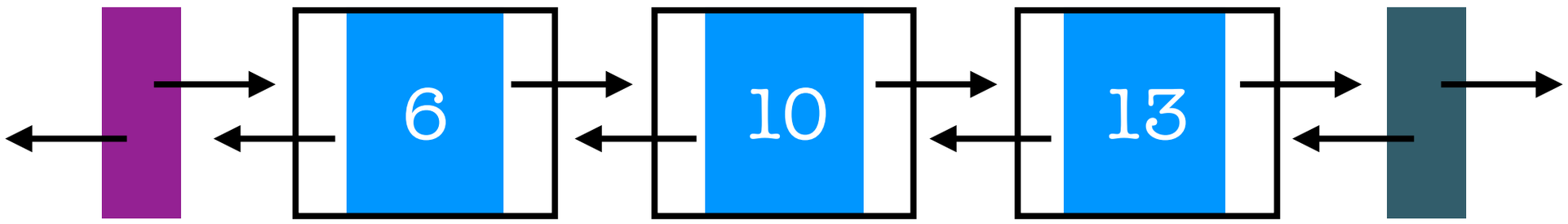


Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	O(n)
tamanho	O(1)	O(1)
inserção	O(n)	O(1)
remoção	O(n)	
particionar	O(n)	
duplicar	O(n)	
ordenar	O(nlogn)	

remoção

linkedList.h

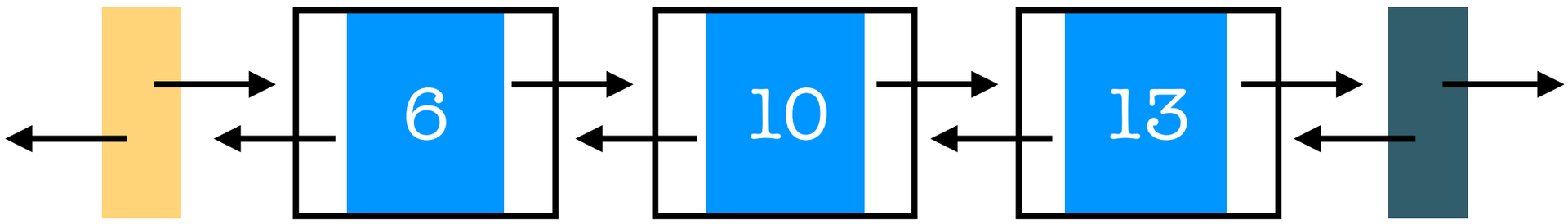
```
template <typename T>
void LinkedList<T>::erase(LinkedList<T>::Node * pos) {
    if (pos != inicio && pos != fim) {
        Node * anterior = pos->prev;
        Node * seguinte = pos->next;
        delete pos;
        anterior->next = seguinte;
        seguinte->prev = anterior;
        tamanho--;
    }
}
```



remoção

linkedList.h

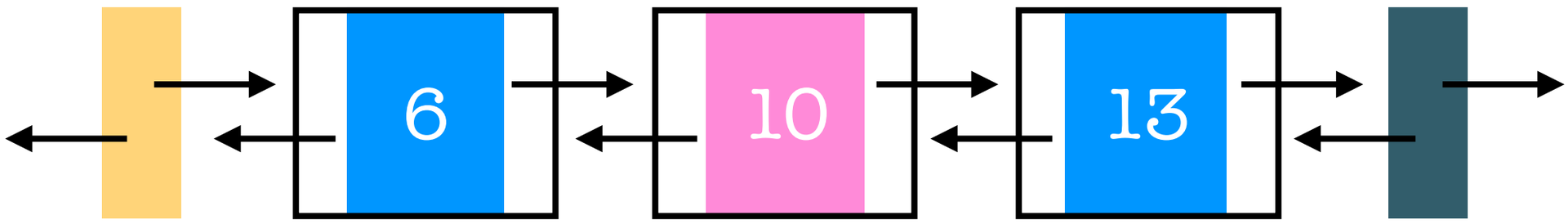
```
template <typename T>
void LinkedList<T>::erase(LinkedList<T>::Node * pos) {
    if (pos != inicio && pos != fim) {
        Node * anterior = pos->prev;
        Node * seguinte = pos->next;
        delete pos;
        anterior->next = seguinte;
        seguinte->prev = anterior;
        tamanho--;
    }
}
```



remoção

linkedList.h

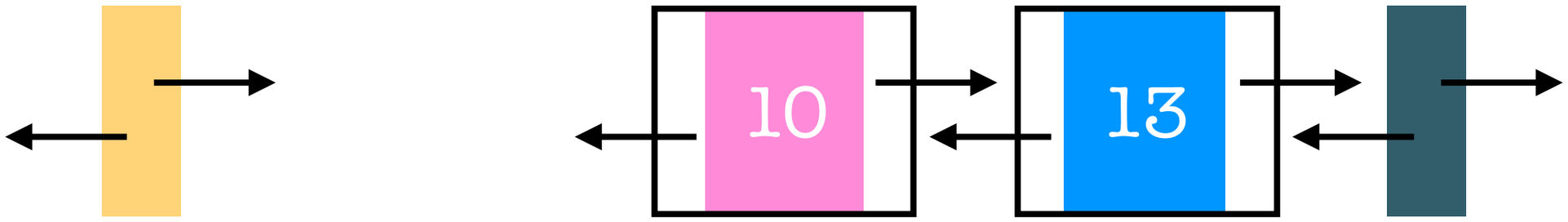
```
template <typename T>
void LinkedList<T>::erase(LinkedList<T>::Node * pos) {
    if (pos != inicio && pos != fim) {
        Node * anterior = pos->prev;
        Node * seguinte = pos->next;
        delete pos;
        anterior->next = seguinte;
        seguinte->prev = anterior;
        tamanho--;
    }
}
```



remoção

linkedList.h

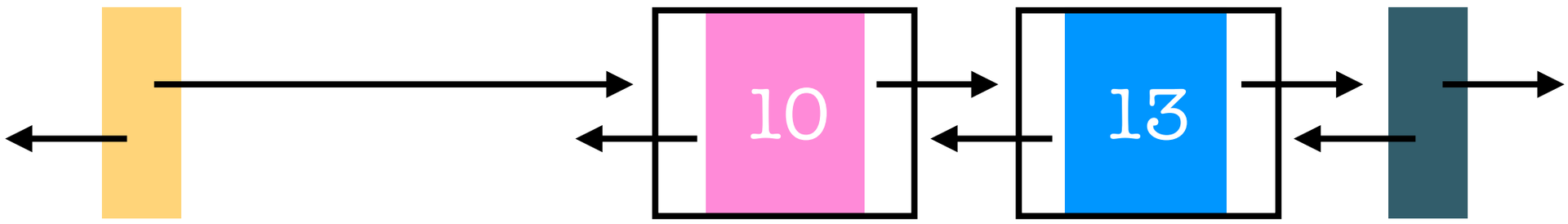
```
template <typename T>
void LinkedList<T>::erase(LinkedList<T>::Node * pos) {
    if (pos != inicio && pos != fim) {
        Node * anterior = pos->prev;
        Node * seguinte = pos->next;
        delete pos;
        anterior->next = seguinte;
        seguinte->prev = anterior;
        tamanho--;
    }
}
```



remoção

linkedList.h

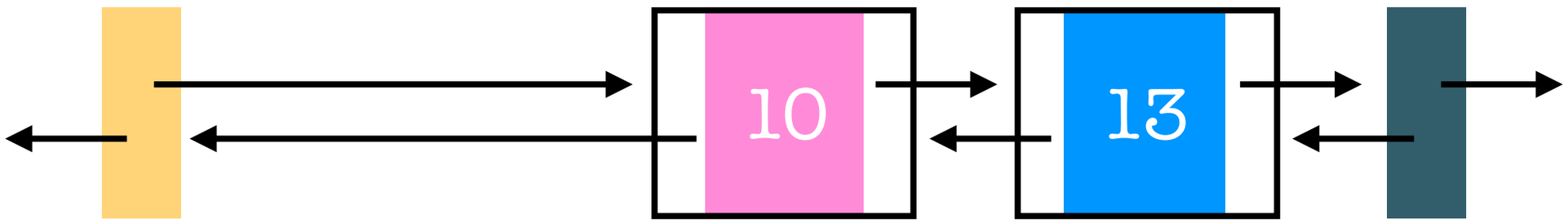
```
template <typename T>
void LinkedList<T>::erase(LinkedList<T>::Node * pos) {
    if (pos != inicio && pos != fim) {
        Node * anterior = pos->prev;
        Node * seguinte = pos->next;
        delete pos;
        anterior->next = seguinte;
        seguinte->prev = anterior;
        tamanho--;
    }
}
```



remoção

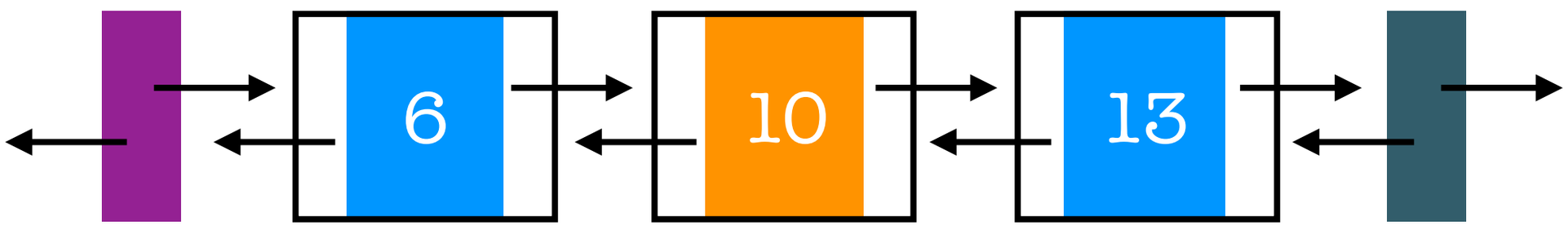
linkedList.h

```
template <typename T>
void LinkedList<T>::erase(LinkedList<T>::Node * pos) {
    if (pos != inicio && pos != fim) {
        Node * anterior = pos->prev;
        Node * seguinte = pos->next;
        delete pos;
        anterior->next = seguinte;
        seguinte->prev = anterior;
        tamanho--;
    }
}
```

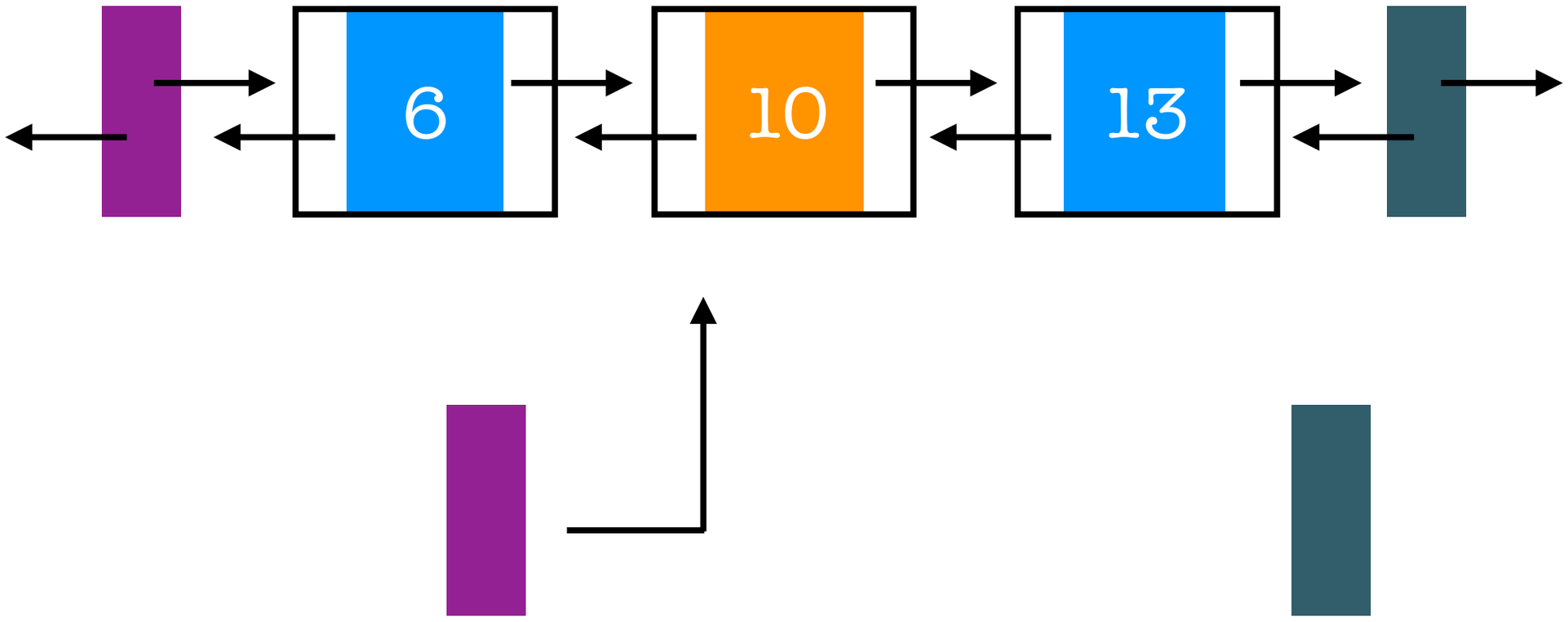


Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	O(n)
tamanho	O(1)	O(1)
inserção	O(n)	O(1)
remoção	O(n)	O(1)
particionar	O(n)	
duplicar	O(n)	
ordenar	O(nlogn)	

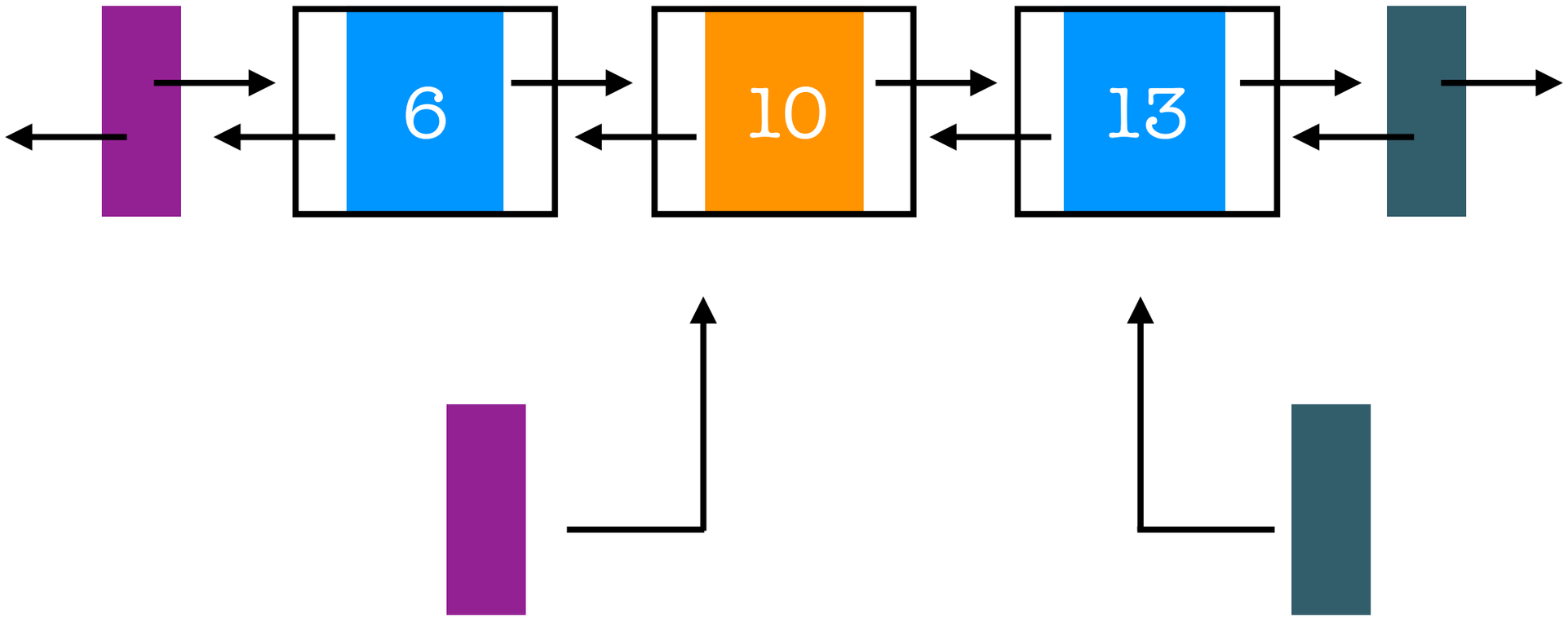
particionar



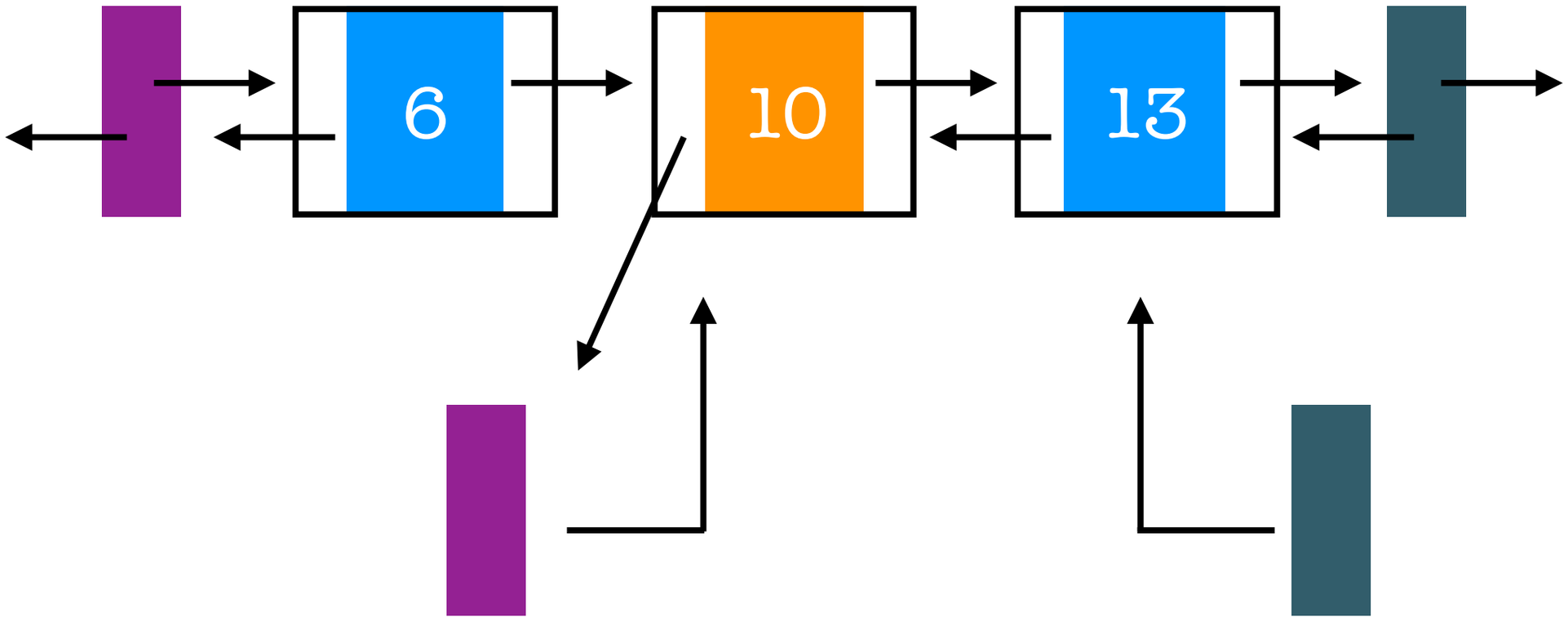
particionar



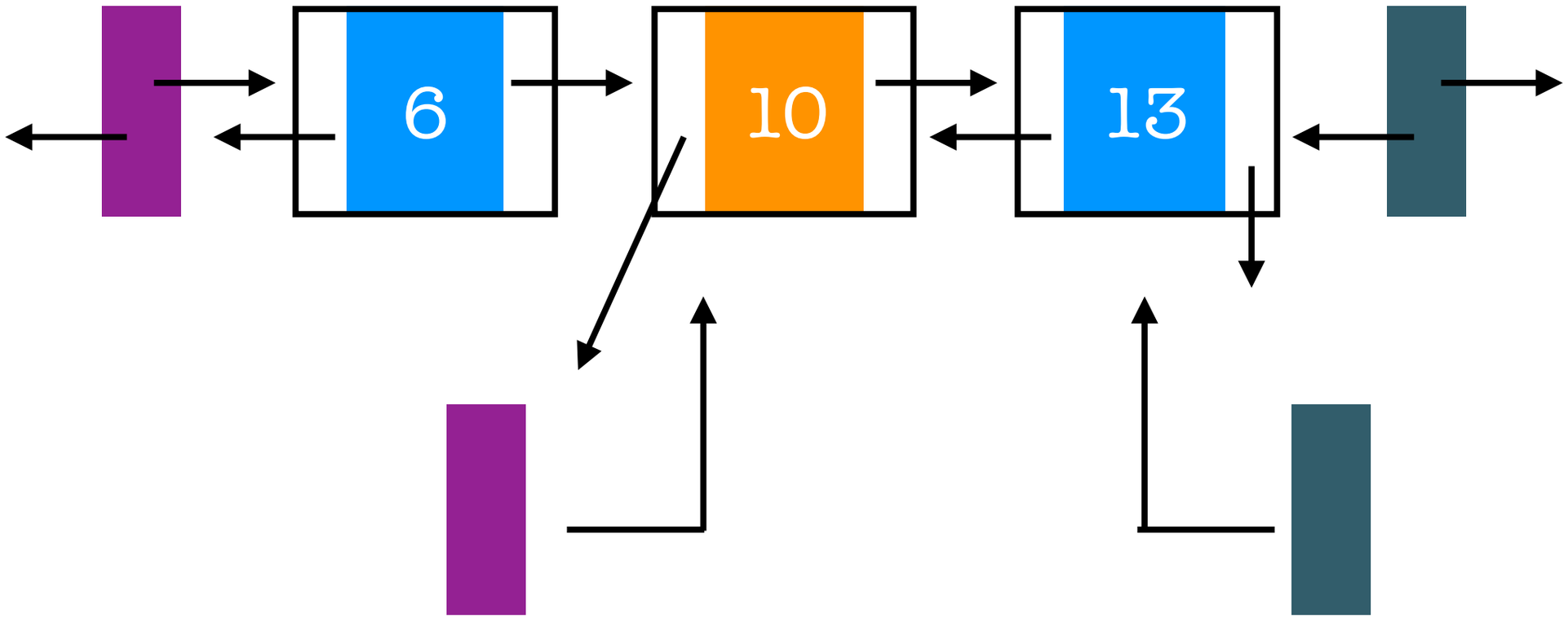
particionar



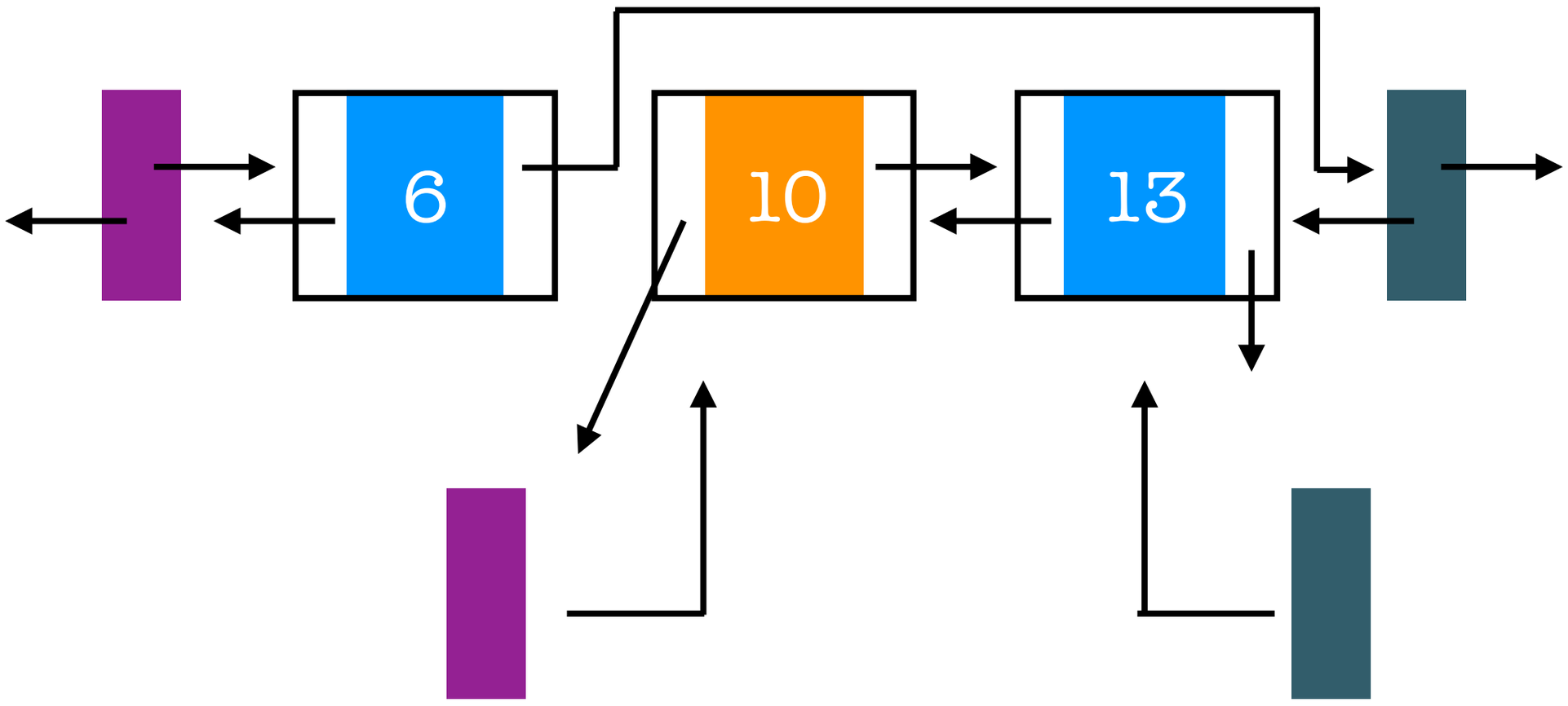
particionar



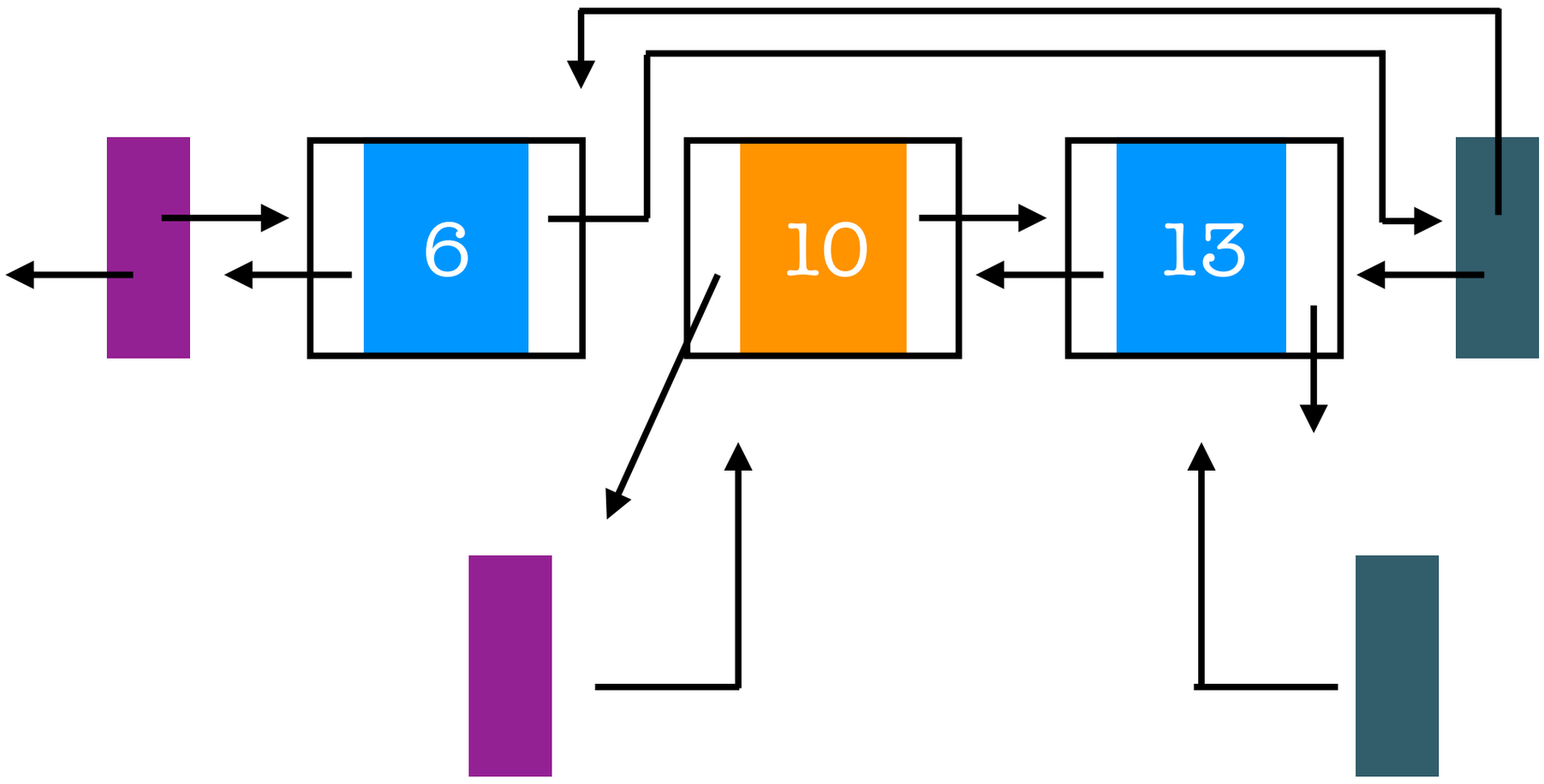
particionar



particionar



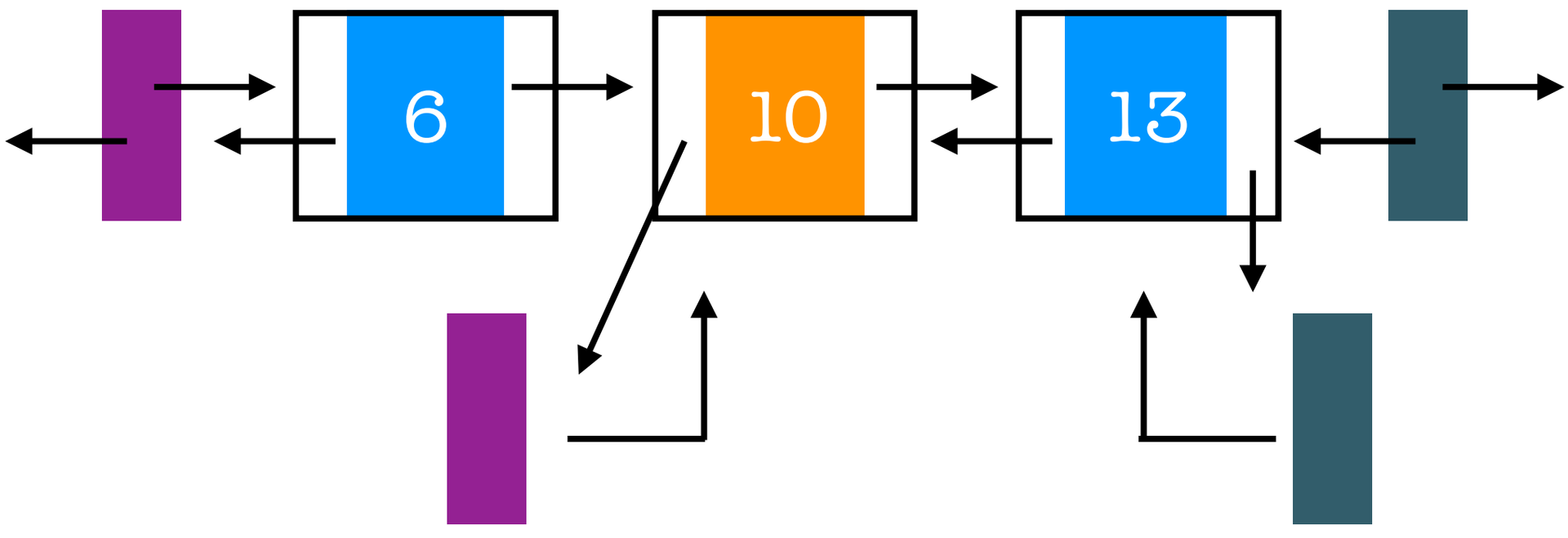
particionar



particionar

linkedList.h

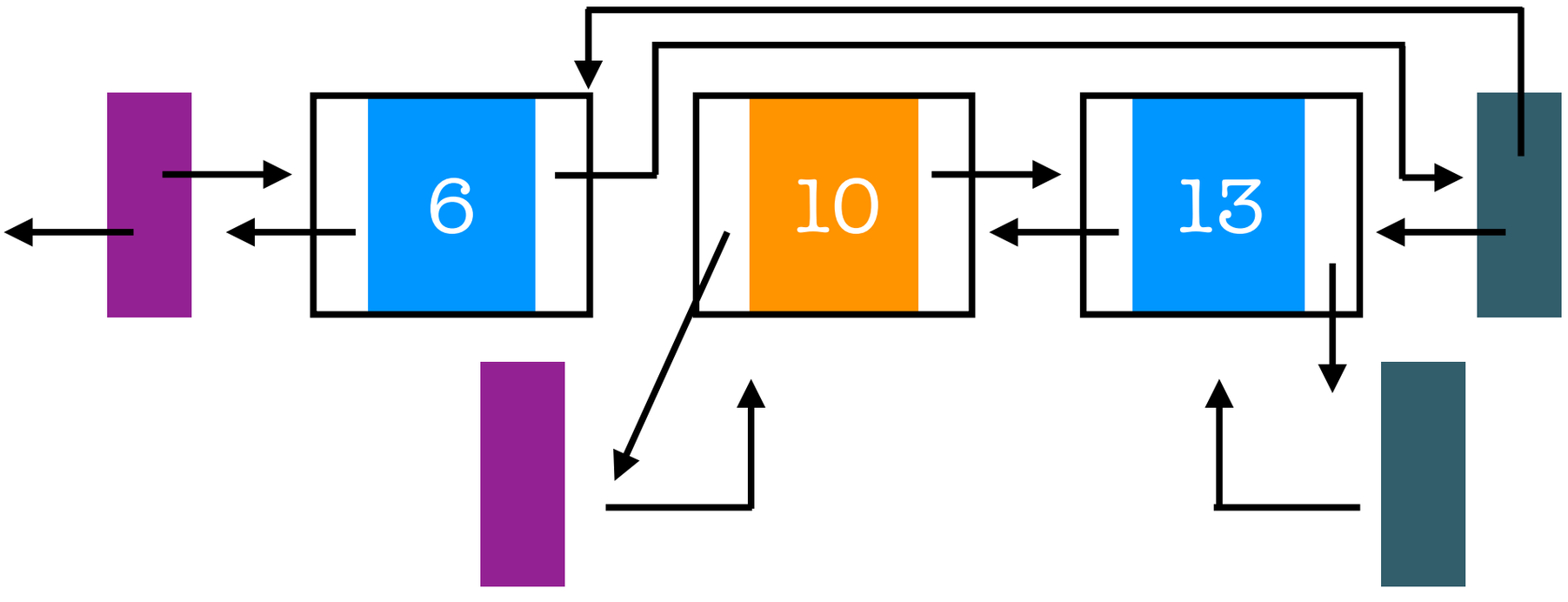
```
template <typename T>
LinkedList<T>::LinkedList(LinkedList<T>::Node * _inicio, LinkedList<T>::Node *_fim) {
    inicio->next = _inicio;
    fim->prev = _fim;
    _inicio->prev = inicio;
    _fim->next = fim;
}
```



particionar

linkedList.h

```
template <typename T>
LinkedList<T> * LinkedList<T>::split(LinkedList<T>::Node * pos) {
    if (pos->prev != inicio && pos->next != fim) {
        Node * anterior = pos->prev;
        LinkedList novaLista = new LinkedList(pos, fim->prev);
        anterior->next = fim;
        fim->prev = anterior;
        return novaLista;
    }
    return nullptr;
}
```



Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	O(n)
tamanho	O(1)	O(1)
inserção	O(n)	O(1)
remoção	O(n)	O(1)
particionar	O(n)	O(n)
duplicar	O(n)	
ordenar	O(nlogn)	

duplicar

linkedList.h

```
template <typename T>
typename LinkedList<T> * LinkedList<T>::clone(void) {
    LinkedList<T> * copia = new LinkedList<T>();
    for (Node * itr = inicio->next; itr != fim; itr++)
        copia->insert(fim, itr->chave);
    return copia;
}
```

Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	O(n)
tamanho	O(1)	O(1)
inserção	O(n)	O(1)
remoção	O(n)	O(1)
particionar	O(n)	O(n)
duplicar	O(n)	O(n)
ordenar	O(nlogn)	

Operação	Vetor	Lista Encadeada
acesso	O(1)	O(n)
busca	O(n)	O(n)
tamanho	O(1)	O(1)
inserção	O(n)	O(1)
remoção	O(n)	O(1)
particionar	O(n)	O(n)
duplicar	O(n)	O(n)
ordenar	O(nlogn)	O(nlogn)