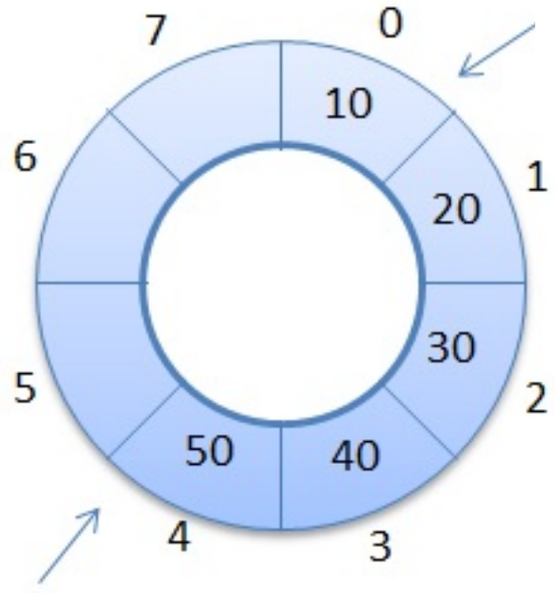
vetor **circular**

front = 0



back = 4

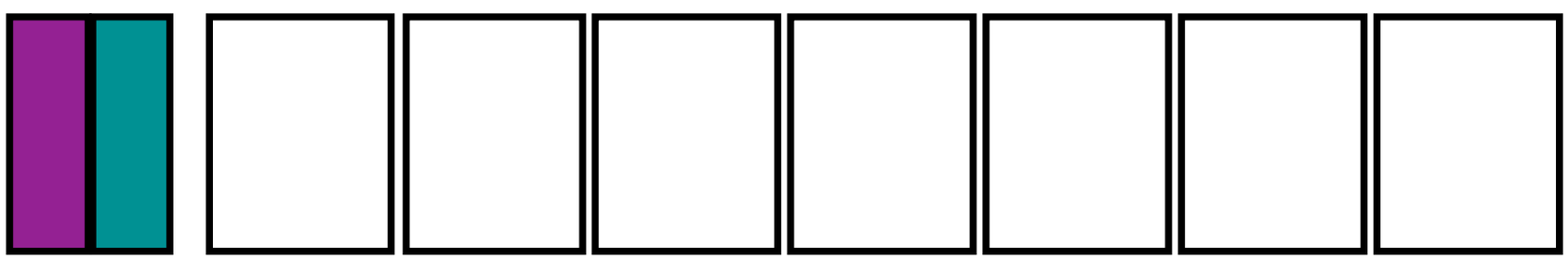
circularArray.h

```
template <typename T>
class CircularArray {
    private:
        vector<T> data;
        int _front, _back, _size;
    public:
        CircularArray() : data(5), _front(-1),
                          _back(-1), _size(0) {}

        unsigned size() { return _size; }
        bool empty () { return !size(); }
        unsigned capacity() {
            return data.size();
        }
        bool full() {
            return size() == capacity();
        }

    public:
        T front (void) { return data.at(_front); }
        T back (void) { return data.at(_back); }

        void print () {
            if (empty()) return;
            unsigned i = (_front - 1) % capacity();
            do {
                i = (i+1) % capacity();
                cout << data.at(i) << " ";
            }
            while (i != _back);
            cout << endl;
        }
};
```

push_back

circularArray.h

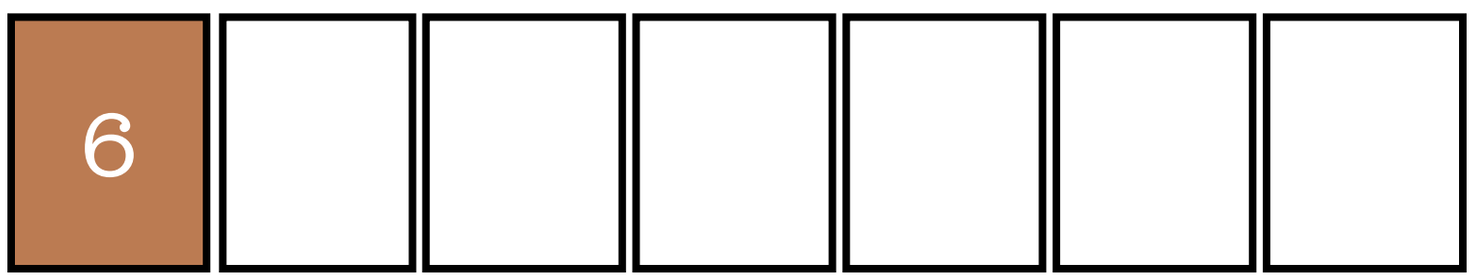
```
template <typename T>
void CircularArray<T>::push_back(T chave) {
    if (full()) resize(capacity() * 2);
    if (empty()) _front = _back = 0;
    else _back = (_back + 1) % capacity();
    data.at(_back) = value;
   _size++;
}
```



push_back

circularArray.h

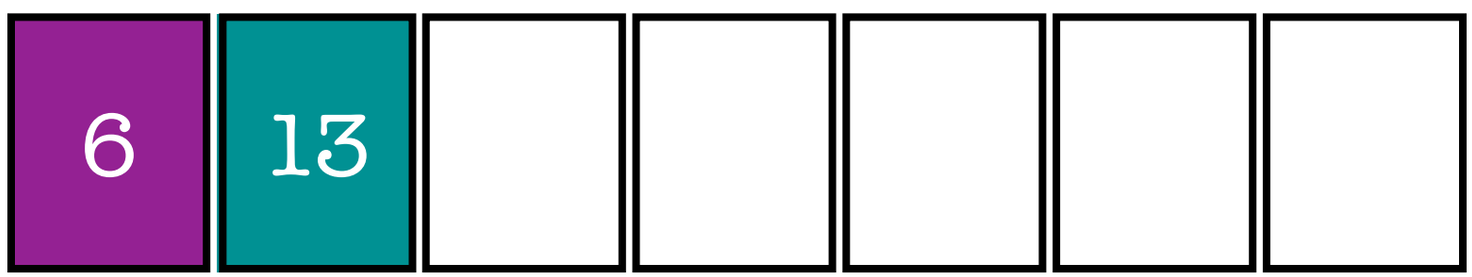
```
template <typename T>
void CircularArray<T>::push_back(T chave) {
    if (full()) resize(capacity() * 2);
    if (empty()) _front = _back = 0;
    else _back = (_back + 1) % capacity();
    data.at(_back) = value;
   _size++;
}
```



push_back

circularArray.h

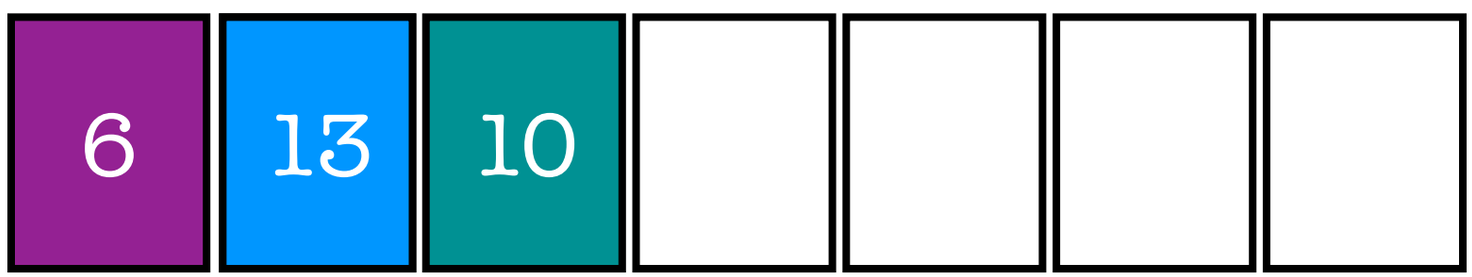
```
template <typename T>
void CircularArray<T>::push_back(T chave) {
    if (full()) resize(capacity() * 2);
    if (empty()) _front = _back = 0;
    else _back = (_back + 1) % capacity();
    data.at(_back) = value;
   _size++;
}
```



push_back

circularArray.h

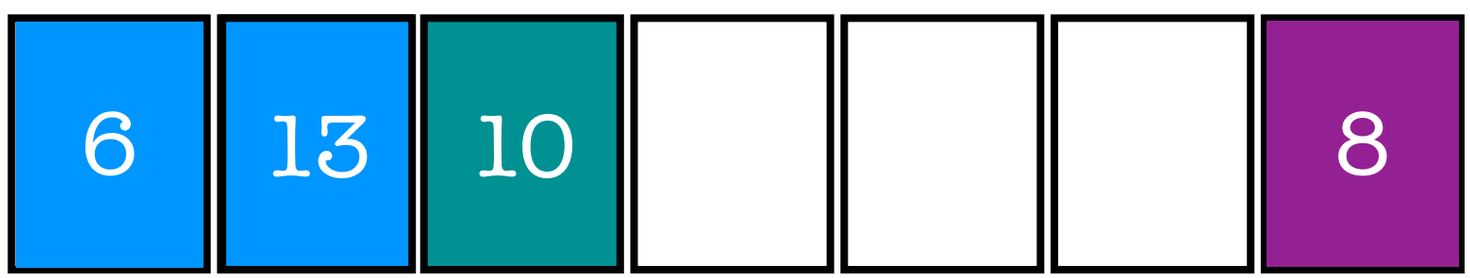
```
template <typename T>
void CircularArray<T>::push_back(T chave) {
    if (full()) resize(capacity() * 2);
    if (empty()) _front = _back = 0;
    else _back = (_back + 1) % capacity();
    data.at(_back) = value;
   _size++;
}
```



push_front

circularArray.h

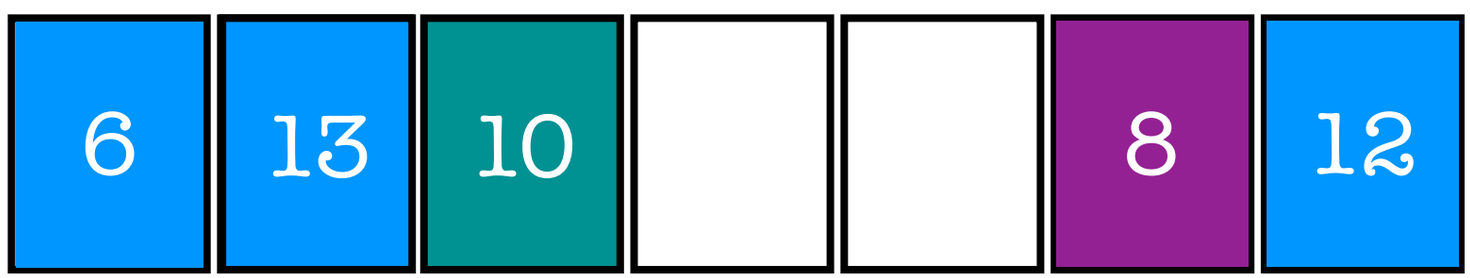
```
template <typename T>
void CircularArray<T>::push_front(T chave) {
    if (full()) resize(capacity() * 2);
    if (empty()) _front = _back = 0;
    else _front = (_front - 1) % capacity();
    data.at(_front) = value;
   _size++;
}
```



push_front

circularArray.h

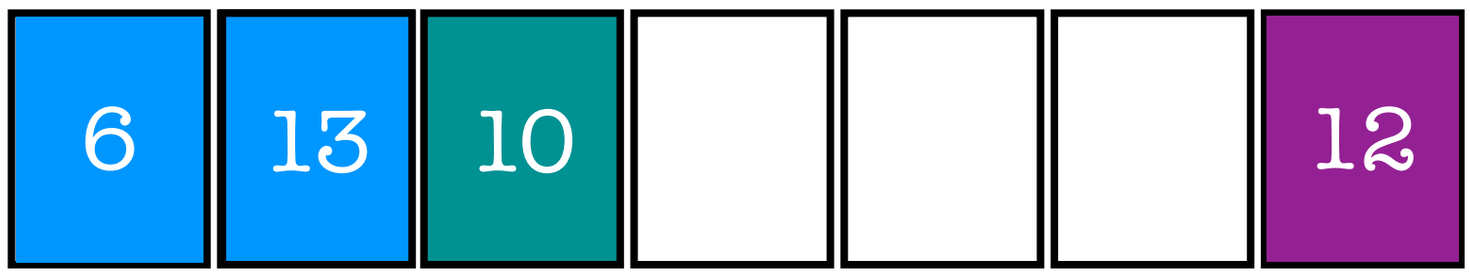
```
template <typename T>
void CircularArray<T>::push_front(T chave) {
    if (full()) resize(capacity() * 2);
    if (empty()) _front = _back = 0;
    else _front = (_front - 1) % capacity();
    data.at(_front) = value;
   _size++;
}
```



pop_front

circularArray.h

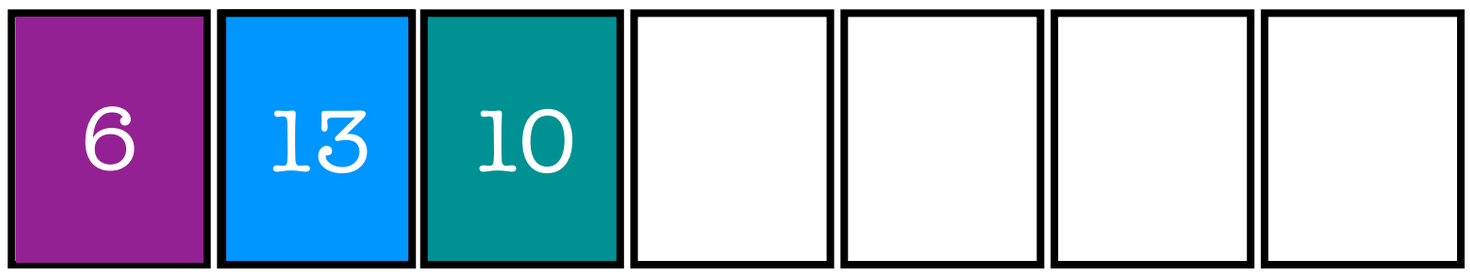
```
template <typename T>
void CircularArray<T>::pop_front(T chave) {
    if (_front == _back) _front = _back = -1;
    else {
        _front = (_front +1 ) % capacity();
        _size--;
        if (size() <= capacity() / 8) resize(capacity() / 2);
    }
}
```



pop_front

circularArray.h

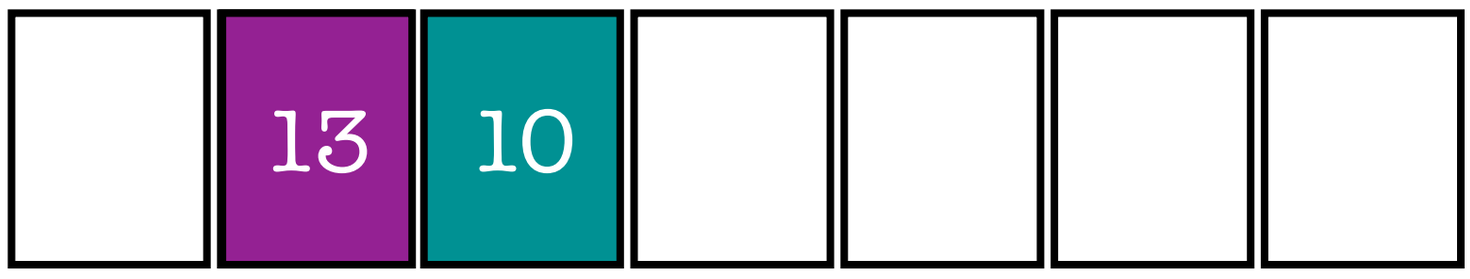
```
template <typename T>
void CircularArray<T>::pop_front(T chave) {
    if (_front == _back) _front = _back = -1;
    else {
        _front = (_front +1 ) % capacity();
        _size--;
        if (size() <= capacity() / 8) resize(capacity() / 2);
    }
}
```



pop_front

circularArray.h

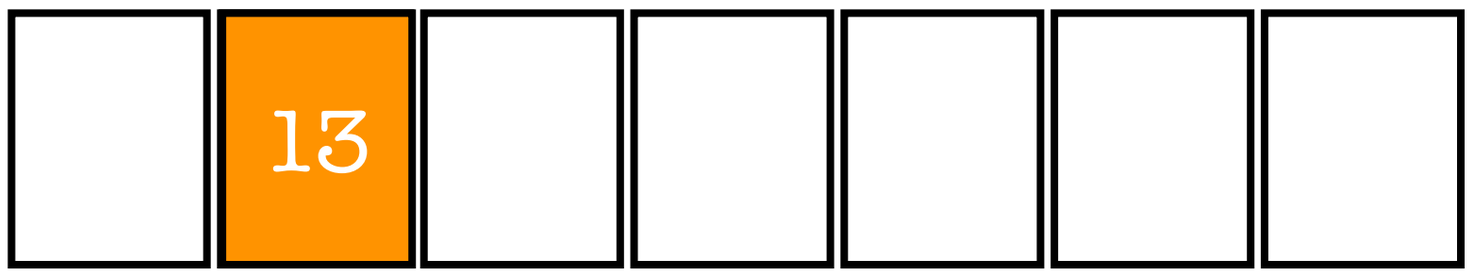
```
template <typename T>
void CircularArray<T>::pop_front(T chave) {
    if (_front == _back) _front = _back = -1;
    else {
        _front = (_front + 1) % capacity();
        _size--;
        if (size() <= capacity() / 8) resize(capacity() / 2);
    }
}
```



pop_back

circularArray.h

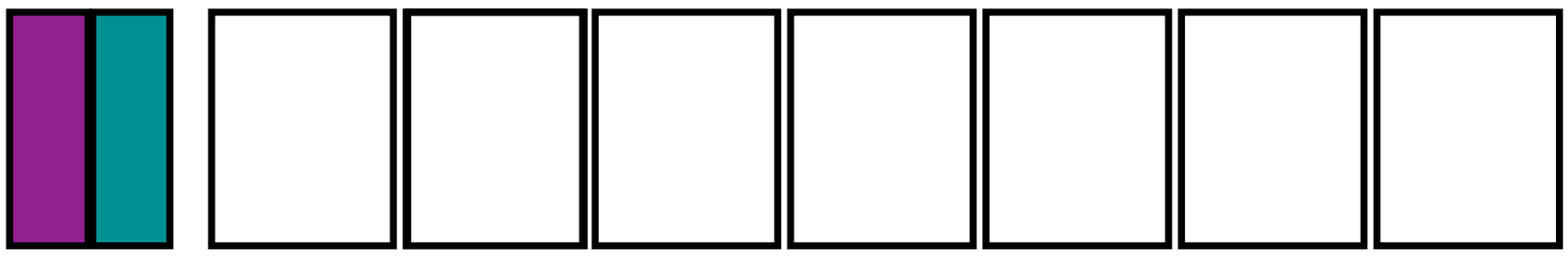
```
template <typename T>
void CircularArray<T>::pop_back(T chave) {
    if (_front == _back) _front = _back = -1;
    else {
        _back = (_back - 1) % capacity();
        _size--;
        if (size() <= capacity() / 8) resize(capacity() / 2);
    }
}
```



pop_back

circularArray.h

```
template <typename T>
void CircularArray<T>::pop_back(T chave) {
    if (_front == _back) _front = _back = -1;
    else {
        _back = (_back - 1) % capacity();
        _size--;
        if (size() <= capacity() / 8) resize(capacity() / 2);
    }
}
```

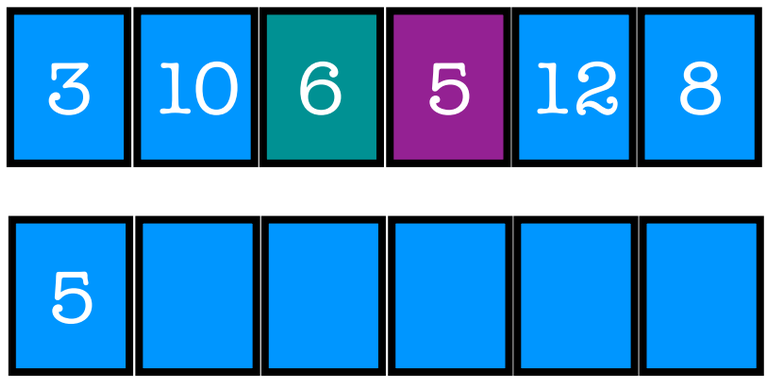


resize

circularArray.h

```
template <typename T>
void CircularArray<T>::resize(unsigned _capacity) {
    vector<T> aux(_capacity);
    unsigned k = 0;
    unsigned i = (_front - 1) % capacity();
    do {
        i = (i + 1) % capacity();
        aux.at(k++) = data.at(i);
    }
    data = vector<T>(aux);
    _front = 0;
    _back = _size() - 1;
}
```

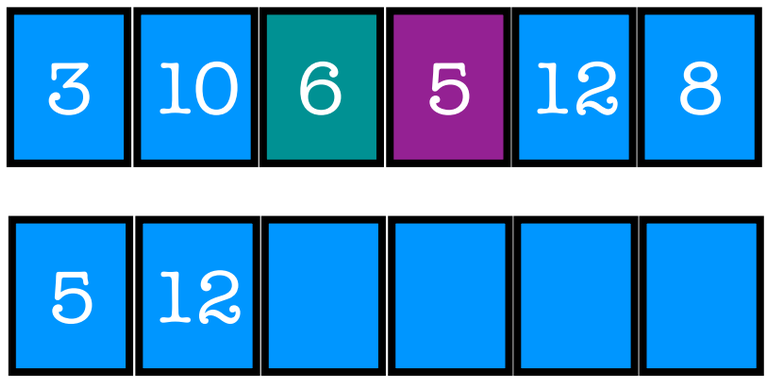
resize



circularArray.h

```
template <typename T>
void CircularArray<T>::resize(unsigned _capacity) {
    vector<T> aux(_capacity);
    unsigned k = 0;
    unsigned i = (_front - 1) % capacity();
    do {
        i = (i + 1) % capacity();
        aux.at(k++) = data.at(i);
    }
    data = vector<T>(aux);
    _front = 0;
    _back = _size() - 1;
}
```

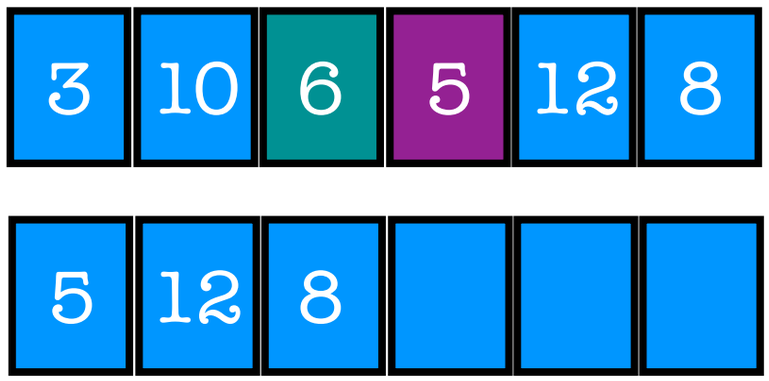
resize



circularArray.h

```
template <typename T>
void CircularArray<T>::resize(unsigned _capacity) {
    vector<T> aux(_capacity);
    unsigned k = 0;
    unsigned i = (_front - 1) % capacity();
    do {
        i = (i + 1) % capacity();
        aux.at(k++) = data.at(i);
    }
    data = vector<T>(aux);
    _front = 0;
    _back = _size() - 1;
}
```

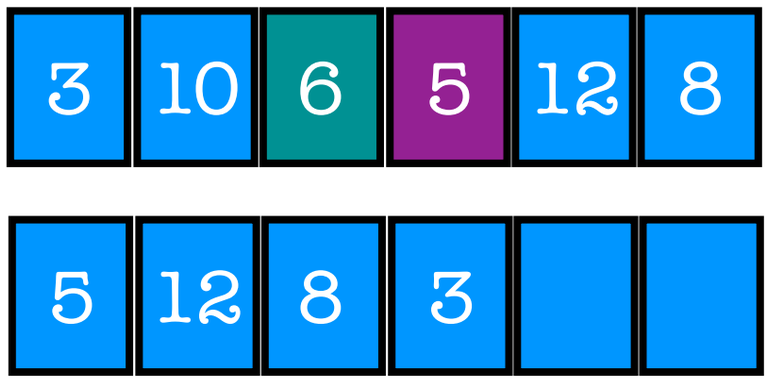
resize



circularArray.h

```
template <typename T>
void CircularArray<T>::resize(unsigned _capacity) {
    vector<T> aux(_capacity);
    unsigned k = 0;
    unsigned i = (_front - 1) % capacity();
    do {
        i = (i + 1) % capacity();
        aux.at(k++) = data.at(i);
    }
    data = vector<T>(aux);
    _front = 0;
    _back = _size() - 1;
}
```

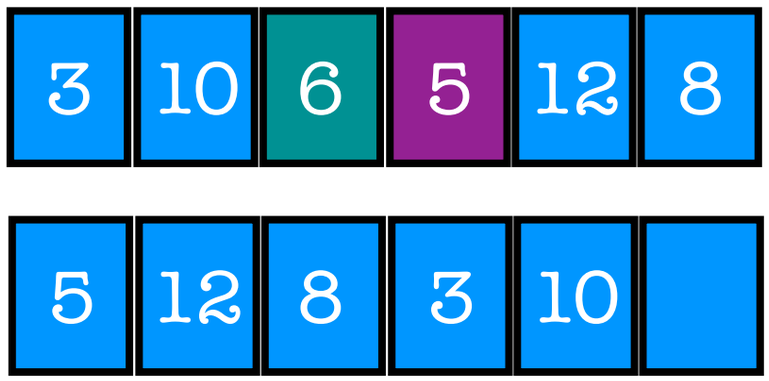
resize



circularArray.h

```
template <typename T>
void CircularArray<T>::resize(unsigned _capacity) {
    vector<T> aux(_capacity);
    unsigned k = 0;
    unsigned i = (_front - 1) % capacity();
    do {
        i = (i + 1) % capacity();
        aux.at(k++) = data.at(i);
    }
    data = vector<T>(aux);
    _front = 0;
    _back = _size() - 1;
}
```

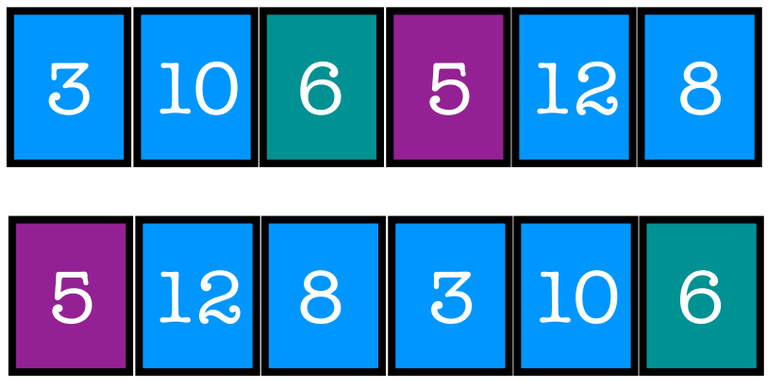
resize



circularArray.h

```
template <typename T>
void CircularArray<T>::resize(unsigned _capacity) {
    vector<T> aux(_capacity);
    unsigned k = 0;
    unsigned i = (_front - 1) % capacity();
    do {
        i = (i + 1) % capacity();
        aux.at(k++) = data.at(i);
    }
    data = vector<T>(aux);
    _front = 0;
    _back = _size() - 1;
}
```

resize



circularArray.h

```
template <typename T>
void CircularArray<T>::resize(unsigned _capacity) {
    vector<T> aux(_capacity);
    unsigned k = 0;
    unsigned i = (_front - 1) % capacity();
    do {
        i = (i + 1) % capacity();
        aux.at(k++) = data.at(i);
    }
    data = vector<T>(aux);
    _front = 0;
    _back = _size() - 1;
}
```