# 5.13. Контейнер deque: внутреннее устройство с алгоритмической точки зрения, сходства и различия с vector. Что хранится внутри deque, как происходит его расширение? Объяснение асимптотики работы методов push_back/push_front и [ ], за счет чего она достигается. Как устроены итераторы в deque, что хранят итераторы? Разница между инвалидацией итераторов и инвалидацией ссылок, правила инвалидации итераторов и ссылок в deque с объяснением, почему они такие.

Посмотрим на контейнеры:

Первые три контейнера - последовательные (sequence containers), вторые - associative containers.

| Container | indexating[] | push_front | push_back | insert(it) | erase(it) | find | iter |
|---|---|---|---|---|---|---|---|
| vector | $O(1)$ | - | $O(1)$ amort | $O(n)$ | $O(n)$ | - | RA |
| deque | $O(1)$ | $O(1)$ | $O(1)$ | $O(n)$ | $O(n)$ | - | RA |

**Deque** реализован как массив массивов (T**). В каждой ячейке большого массива есть указатель на элементы другого массива, где уже лежат элементы. Также храним номер ячейки (номер "строки"и номер "столбца") начала + номер ячейки, где конец дека. Всё промежуточное пространство заполнено элементами. [ ] по индексу - вычисляем исходя из позиции начала и размера одного массива, куда достать. Итератор random access - так как он хранит вот эти два индекса (не просто указатель) $\Rightarrow$ за O(1) можно пересчитать в двумерном массиве прибавление int. Аналогично push_front и push_back работают за амортизированное O(1), так как нам изредка надо будет увеличивать размер дека в два раза.

Инвалидация итераторов:

**Iterator invalidation**

This section is incomplete

There are still a few inaccuracies in this section, refer to individual member function pages for more detail

| Operations | Invalidated |
|---|---|
| All read only operations | Never |
| swap, std::swap | The past-the-end iterator may be invalidated (implementation defined) |
| shrink_to_fit, clear, insert, emplace, push_front, push_back, emplace_front, emplace_back | Always |
| erase | If erasing at begin - only erased elements<br>If erasing at end - only erased elements and the past-the-end iterator<br>Otherwise - all iterators are invalidated (including the past-the-end iterator). |
| resize | If the new size is smaller than the old one : only erased elements and the past-the-end iterator<br>If the new size is bigger than the old one : all iterators are invalidated<br>Otherwise - none iterators are invalidated. |
| pop_front | Only to the element erased |
| pop_back | Only to the element erased and the past-the-end iterator |

**Invalidation notes**

- When inserting at either end of the deque, references are not invalidated by insert and emplace.
- push_front, push_back, emplace_front and emplace_back do not invalidate any references to elements of the deque.
- When erasing at either end of the deque, references to non-erased elements are not invalidated by erase, pop_front and pop_back.
- A call to resize with a smaller size does not invalidate any references to non-erased elements.
- A call to resize with a bigger size does not invalidate any references to elements of the deque.