# Алгоритми за търсене

1 При несортиран масив мин сложност е $O(n)$

• Ако ще търсим многократно в масива тогава сортираме

## Търсене в C++/STL

1 find търси линейно и връща итератор към първото срещане

```
//find(v.begin(), v.end(), 5)        //auto result = -//-
```

2 binary_search(b, e, 5) - връща true/false

3 upper_bound(-//-) - връща итератор към 1вия ел. по-голям от търсения

Пр! `vector<int> v = {1,2,3,3,4,4};`
`auto up1 = upper_bound(v.b, v.e, 5);` // v.end защото го няма
`up1 - v.begin() = 6;` // size на v

4 lower_bound(-//-) - връща итератор към 1вия ел. по-голям или равен на ел. (пак връща v.end ако няма)

### Задача за телефони и сграда

1. 100 етажна сграда и 2 телефона – кога ще се скъсат

Отг1! Jump search с step = $\sqrt{100} = 10$ и после 1 по 1 напоре

Отг2! Jump search с step = от пирамида: $\frac{1}{2} \cdot x \cdot (x-1) = 100$

$\Rightarrow x^2 - x = 200 \Rightarrow x^2 - x - 200 = 0$ $\quad x = 13.65 = 14$

$\Rightarrow$ step = 14 - i, за $i = \overline{0,14}$ → i-тото пускане

дане покрива 105 етажа, целта е да покрием повече етажи от малките

## Complexity

Binary - $O(\log_2 N)$

Jump - $O(\sqrt{n})$ // All with space complexity $O(1)$

Ternary - $O(\log_3 N)$

# Масиви

- Знаем адреса на първия и че са последователно в паметта
- ~~константно~~ O(1) за достъп до елемент и промяна на произволна позиция
- O(n) премахване на елемент / добавяне на ел по средата
- фикс размер при създаване

# Списъци

- Знаем адреса на 1вия node (възел) и всеки node знае адреса на следващия


- O(1) - добавяне на ел ~~нач~~ в края, сливане на масиви
- динамичен размер
- O(n) - достъп до елемент
- допълнително заделена памет за ptr към next

- Основни операции - принтиране, търсене, доб. на елемент в нач / края / определена поз., трие не ел. в нач / края / опр. поз.

- добавяне в нач/край -> O(1), друго са O(n)
- push_back - амортизирана конст сложност за ~~LinkedList~~ Vector
- добавяне на елемент в края на linked list има сложност O(1) или O(n) зависи от имплементацията

# Обобщение

| | **Масив** | **Свързан списък** |
|---|---|---|
| Позиции на елементите в паметта | Последователни | Непоследователни |
| Размер | Фиксиран | Нефиксиран |
| Достъп до елемент на определена позиция | O(1) | O(n) |
| Добавяне/премахване на елемент в началото и в края | O(n) | O(1) |
| Добавяне/премахване на елемент | O(n) | O(n) |
| Възможност за реализация на двоично търсене |  |  |

| Operation / Data structure | Array | Singly linked list without tail | Singly linked list with tail | Doubly linked list without tail | Doubly linked list with tail |
|---|---|---|---|---|---|
| push_front | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |
| pop_front | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |
| get_front | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |
| push_back | $\mathcal{O}(1)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ |
| pop_back | $\mathcal{O}(1)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ |
| get_back | $\mathcal{O}(1)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ |
| get_at | $\mathcal{O}(1)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |
| find_key | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |
| erase_key | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |
| is_empty | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |
| add_before | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |
| add_after | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |