Програмування-1

Лекція 11

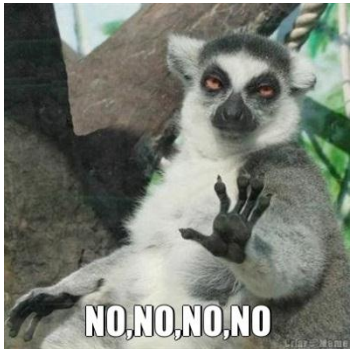
Колекції – Set, TreeSet

Інтерфейс **Set**

- Не допускає додавання дублікатів
- Слід використовувати виключно для immutable об’єктів!
- Є нащадком Collection
  - усі методи такі ж самі як у Collection
    - додатково є контракт на унікальність елементів
- Порядок елементів при перегляді в загальному випадку не гарантується
    - **SortedSet** (інтерфейс) – елементи відсортовані
    - **LinkedHashSet** (класс) – елементи у тому порядку в якому додавались
- Реализації: **TreeSet**, **HashSet**, **LinkedHashSet**, …

Як гарантувати відсутність дублікатів у колекції?

- Відповідь:
  - перед додаванням провести пошук
  - не додавати, якщо такий елемент вже є



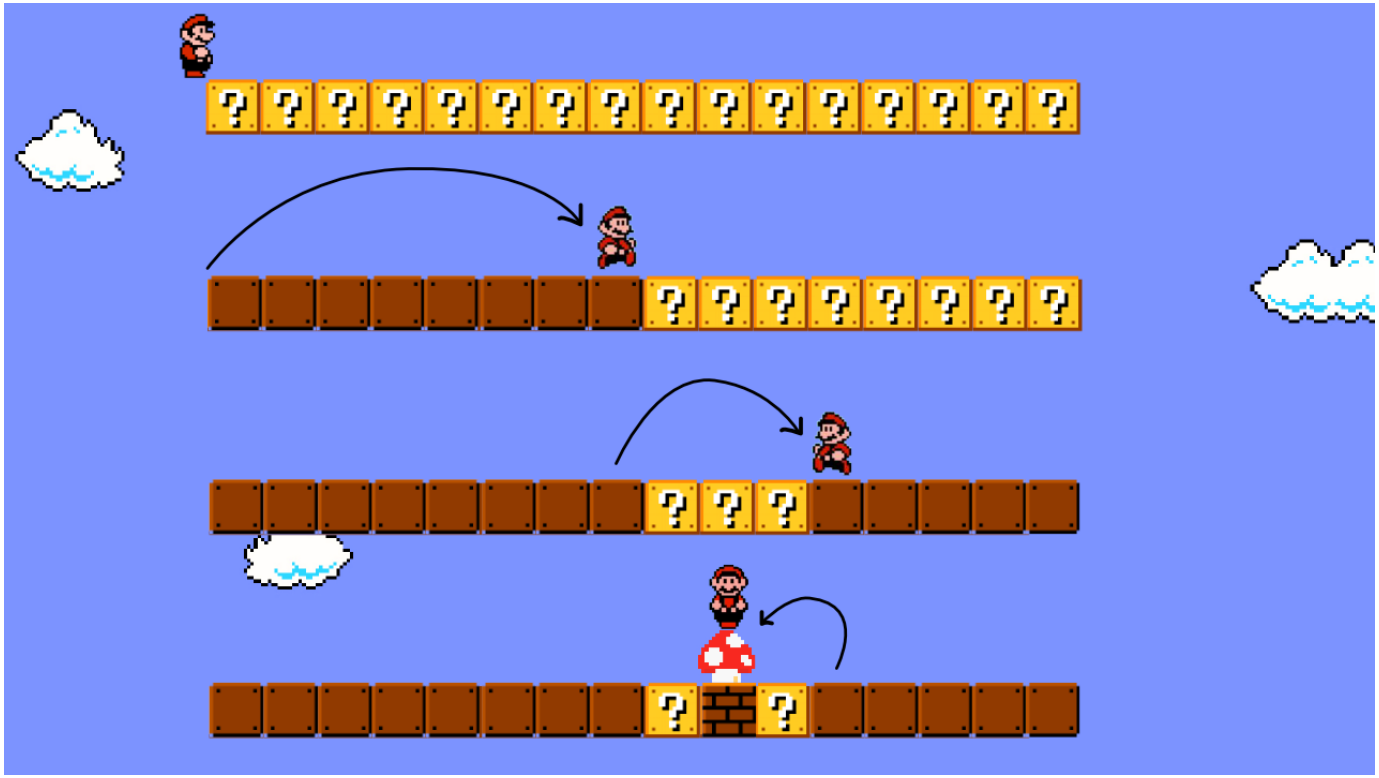
- Пошук має бути швидким!!!

Алгоритми пошуку

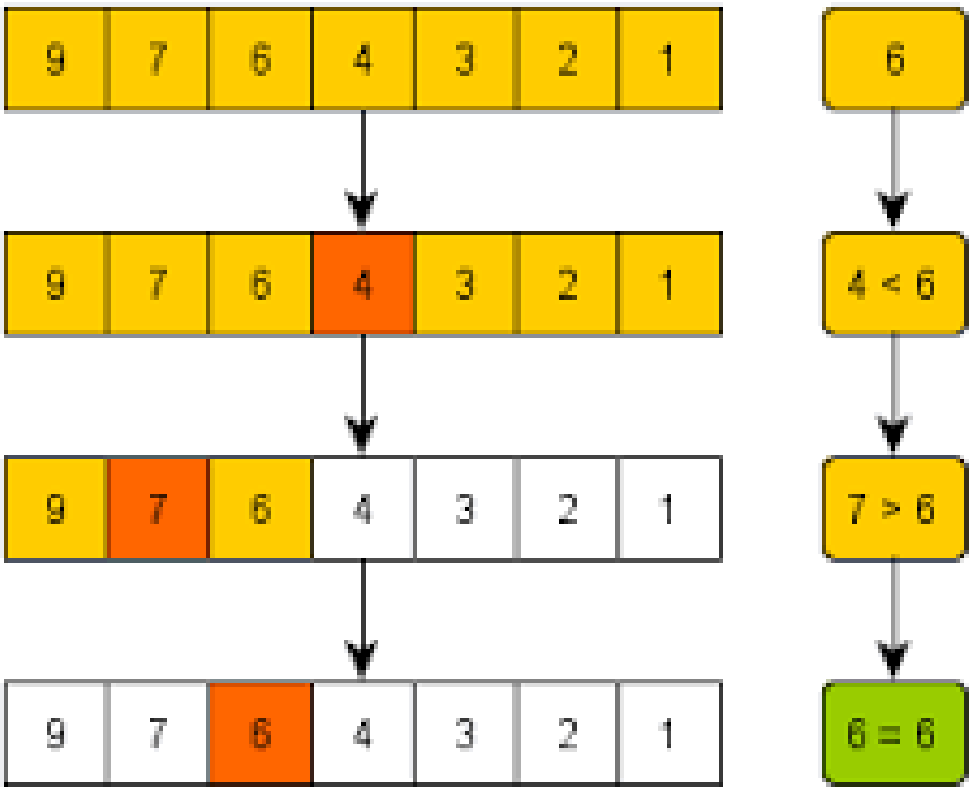
	Лінійний пошук	Двійковий пошук	Хешування
Швидкість (найгірший випадок)	O(n)	O($\log_2 n$)	Залежіть від налаштувань Як правило швидший за двійковий
Вимоги до незмінності об’єктів	немає	immutable	immutable
Вимоги до функціональності	equals()	Comparable or Comparator	equals() + hashCode()

Двійковий пошук

- Працює лише на відсортованих даних

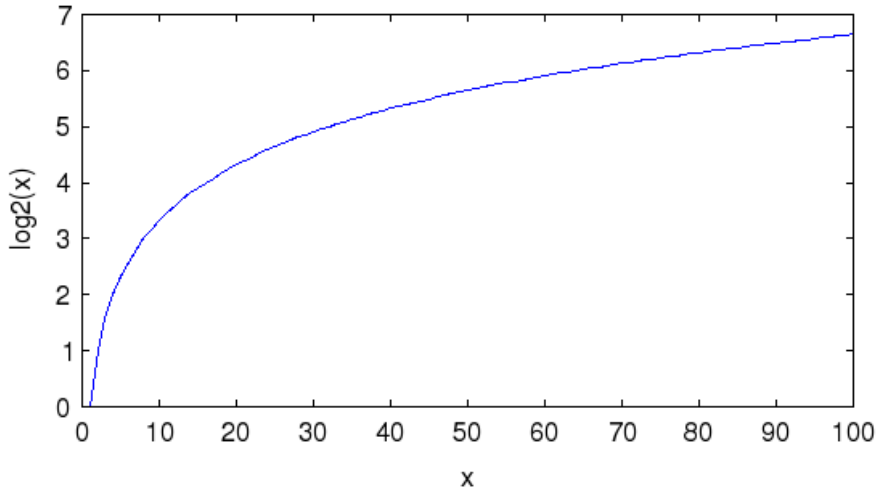


Двійковий пошук

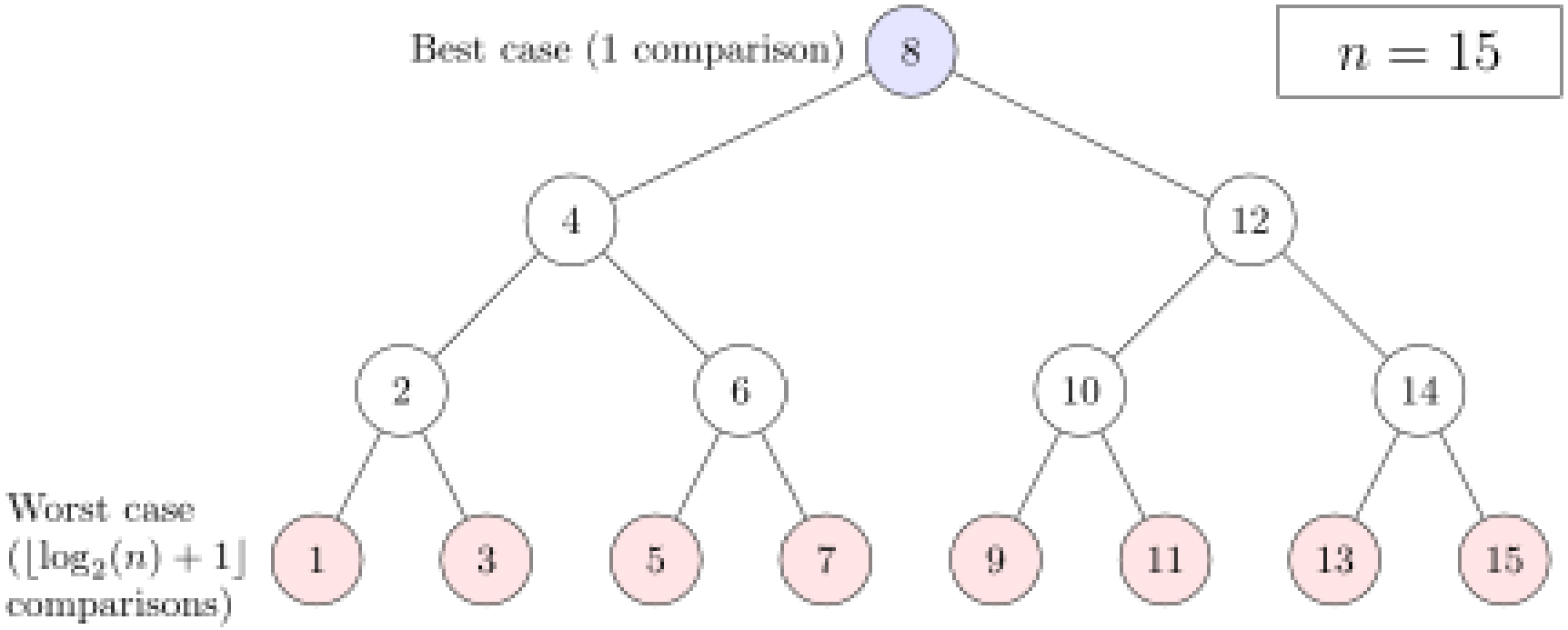


Ефективність двійкового пошуку

n	$\log_2 n$
10	~ 4
100	~ 7
1 000	~ 10
1 000 000	~ 20
1 000 000 000	~ 30



Двійкове дерево



Класс TreeSet

- Дублікати не додаються (implements Set)
- Використовує двійковий пошук по дереву - $O(\log_2 N)$
- Ітератор переглядає елементи у відсортованому порядку (implements SortedSet)
- Додавати null заборонено
- Об’єкти мають бути доступними для порівняння
  - Comparable, Comparator (далі буде)

Клас TreeSet - приклад

```
import java.util.*;

public class TreeSetDemo{

  public static void main(String[] args) {

     Set set = new TreeSet();
     set.add("one");
     set.add("two");
     set.add("three");
     set.add("four");
     set.add("one");
     set.add("two");

    // try this:
    //set.add(new Integer(1)); // Exception. String cannot be compared to Integer
    //set.add(null); // NullPointerException

    // check order (alphabetical for String) and duplicates (none)
    System.out.println(set);
  }
}
```

Порівняння об’єктів у TreeSet

- Один з двох способів:
  - **natural ordering** (implements Comparable)
    - Реалізується в класі, об’єкти якого будуть зберігатися у TreeSet
    - Тому може бути лише один
    - **null** не допускается
  - **Comparator** (implements Comparator)
    - Реалізується в окремому класі
    - Тому може буди будь-яка кількість компараторів
    - **null** допускается, якщо передбачено компаратором
    - деякі методи **TreeSet**, **NavigableSet** повертають **null** як спеціальне значення

- Спосіб порівняння задається конструктором:
  - TreeSet() // natural ordering by Comparable
  - TreeSet(Comparator comparator) // Comparator

Interface Comparable

```
public interface Comparable<T> {
        public int compareTo(T other);
}
```

- compareTo
  - порівнює себе «**this**» з іншим об’єктом «**other**»
  - повертає:
    - < 0 , якщо **this** < **other**
    - 0 , якщо **this** = **other**
    - > 0 , якщо **this** > **other**

Interface Comparator

```
public interface Comparator<T> {
        int compare(T o1, T o2);
}
```

- compareTo
  - порівнює об’єкт «o1» з іншим об’єктом «o2»
  - повертає:
    - < 0 , якщо **o1** < **o2**
    - 0 , якщо **o1** = **o2**
    - > 0 , якщо **o1** > **o2**

Приклад

Є клас:

```
class Person {
        String name;
        String surname;
}
```

Зробити так, щоб об’єкти цього класу можна було зберігати у TreeSet