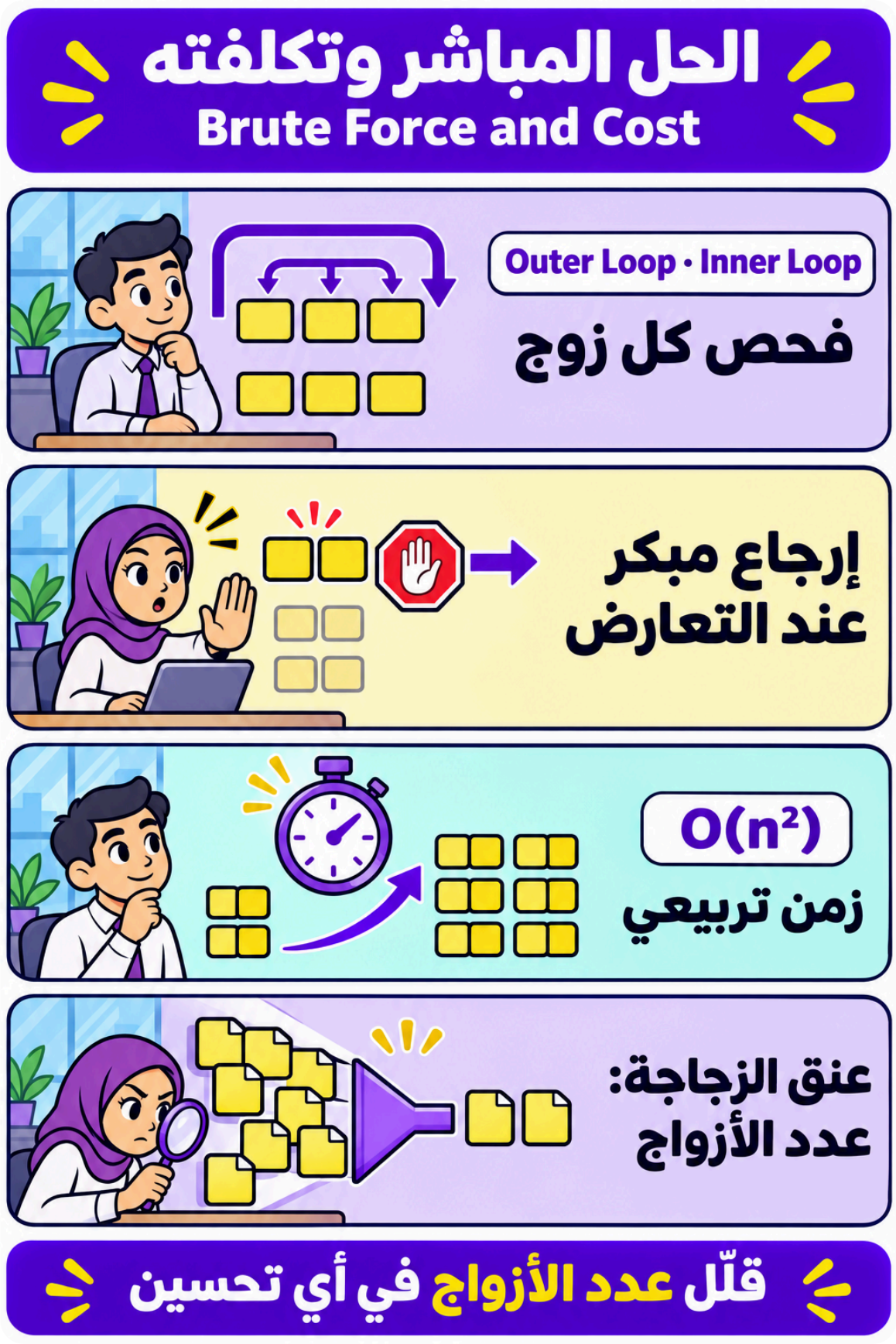
مختصر كتابي

Brute Force and Bottleneck Analysis



كيف تكتب الحل المباشر لمسألة تعارض الحجوزات بلغة Java، وما الذي يجعله بطيئًا مع البيانات الكبيرة؟

الأسلوب الذي نبدأ به هو **الحل المباشر** (Brute Force): أن نفحص الاحتمالات المطلوبة صراحةً حتى نصل إلى الإجابة. نكتب Method اسمها `hasConflict` ترجع Boolean، وتستقبل الحجوزات في **Array ثنائية الأبعاد** (Two-Dimensional Array)؛ كل صف فيها حجز، العنصر رقم 0 وقت البداية، والعنصر رقم 1 وقت النهاية. ونفترض أن المدخلات سليمة كما اتفقنا.

الـLoop الخارجية (Outer Loop) تختار بالـVariable `i` الحجز الأول في المقارنة، والـLoop الداخلية (Inner Loop) تبدأ من `j = i + 1`، حتى لا نقارن الحجز بنفسه ولا نكرر الزوج نفسه معكوسًا. فمع 3 حجوزات نقارن 0 مع 1، و0 مع 2، ثم 1 مع 2، وعندما يكون `i` يساوي 2 لا تُنفَّذ الـLoop الداخلية.

```
static boolean hasConflict(int[][] bookings) {
    for (int i = 0; i < bookings.length; i++) {
        for (int j = i + 1; j < bookings.length; j++) {
            if (bookings[i][0] < bookings[j][1]
                    && bookings[j][0] < bookings[i][1]) {
                return true;
            }
        }
    }
    return false;
}
```

إذا تحقق **شرط التعارض** (Overlap Condition) فإن `return true` تنهي الـMethod كلها فورًا، وهذا هو **الإرجاع المبكر** (Early Return). أما `return false` فمكانها بعد الـLoops الاثنتين؛ ولو وضعناها داخل الـLoop بعد أول مقارنة فاشلة لحكمنا على القائمة كلها من أول زوج، وفاتنا التعارض الذي بعده.

اختبرنا الـMethod على قائمة فيها تعارض بين الحجز الأول والثالث، وقائمة مواعيدها متلامسة بلا تداخل، وحجز داخل حجز، وقائمة فارغة، وحجز واحد؛ والنتائج المتوقعة على الترتيب: `true` و`false` و`true` و`false` و`false`.

لنحلّل التكلفة. إذا رمزنا لعدد الحجوزات بـ `n`، فالحجز الأول يُقارن بـ `n - 1` حجزًا، والثاني بـ `n - 2`، وهكذا حتى آخر زوج. مع 10 حجوزات يكون لدينا 45 زوجًا، ومع 100 حجز 4950 زوجًا، ومع 1000 حجز 499500 زوج، وكل اختبار فيه عدد ثابت من العمليات. هذا **نمو تربيعي** (Quadratic Growth)، والحد الأعلى لـ**التعقيد الزمني** (Time Complexity) هو `O(n^2)`.

- إذا كان التعارض في أول زوج، تنتهي الـMethod في وقت ثابت `O(1)`.
- إذا لم يوجد تعارض، يجب فحص كل الأزواج قبل إرجاع `false`.

- **المساحة الإضافية** (Auxiliary Space Complexity) هي `O(1)`؛ فنحن نستخدم اثنين من الـVariables للفهرسة فقط دون نسخ الحجوزات أو إنشاء Array إضافية.
- **عنق الزجاجة** (Bottleneck) هو عدد الأزواج؛ فاختبار الزوج الواحد بسيط، لكن عدد الأزواج يكبر بسرعة.
- هل يمكن أن نستفيد من ترتيب أو معلومة إضافية لتقليل المقارنات؟ سنعود إلى هذا السؤال مع Algorithms الترتيب (Sorting Algorithms).

جرّب أن تضيف سطر طباعة داخل الـLoop الداخلية قبل شرط التعارض، وشغّل حالة فيها تعارض وأخرى بلا تعارض، وراقب الأزواج التي فُحصت واللحظة التي توقف عندها التنفيذ، ثم احذف سطر الطباعة.

الخلاصة: الحل المباشر يفحص كل زوج مختلف مرة واحدة بتكلفة زمنية `O(n^2)` ومساحة إضافية `O(1)`، وعدد الأزواج هو ما يجب تقليله في أي تحسين لاحق.