פתרון – אלעזר פיין

1.

O(1)	2/n, 34
O(lgn)	Logn, ln(n), nlogn
O(√n)	√n
O(n)	N, n/2, 34n
O(nlgn)	Nloglogn, nlog(n^2), 2nlogn
O(n^c)	$n^{1.5}$ $n^2 - 3n$ $n^2 logn\ (from \lim f\ /n^3)$ $\mathrm{n}lg^2n\ (from \lim f\ /n^2)$ $(n+1)^2$ $n^{1+\varepsilon}\ (0 < \varepsilon < 1)$
O(c^n)	2^n $2^{n/2}$ e^n

2. א. f(n)=O(g(n))
   ב. g(n)=O(f(n))
   ג. g(n)=O(f(n))
   ד. שניהם.

3. נמיר הכל לדקות, נקבל:
   אלגוריתם א': $60n^2$
   אלגוריתם ב': n^3
   אלגוריתם ג': $\frac{2^n}{60}$

נחשב:
$60n^2 > n^3 \rightarrow 60 > n$ כלומר אלגוריתם ב' ירוץ יותר מהר מאלגוריתם א' עבור **n<60**.
$60n^2 > \frac{2^n}{60} \rightarrow 2 > \frac{n}{\lg(60n)} \rightarrow 20 \geq n$ כלומר אלגוריתם ג' ירוץ מהר יותר מא' עבור $n \leq 20$**.**

פתרון – אלעזר פיין

4. פתרון בKotlin לבעיה הבודדת למצוא ***MAX***:

```
fun findMax(arr: Array<Int>, size: Int = arr.size, max: Int = Int.MIN_VALUE): Int
{
    if (size == 0) return max
    return findMax(arr, size - 1, maxOf(arr[size - 1], max))
}
```

מציאת ***MIN*** באופן דומה.
סה"כ n השוואות או 2n השוואות אם מחשיבים את ההשוואה ***size == 0*** .
בהנחה שהמערך ממוין אפשר למצוא בפעולה אחת – לקחת ראשון\אחרון במערך.

פתרון למציאת שניהם בו זמנית:

```
fun findMinMax(arr: Array<Int>, size: Int = arr.size,
               minMax: Pair<Int /*min*/, Int /*max*/> = Pair(Int.MAX_VALUE,
                                                             Int.MIN_VALUE)
): Pair<Int, Int> {
    if (size == 0) return minMax
    return findMinMax(arr, size - 1, Pair(minOf(arr[size - 1], minMax.first),
                                          maxOf(arr[size - 1], minMax.second)))
}
```

סה"כ 2n השוואות או 3n השוואות אם מחשיבים את ההשוואה ***size == 0*** .
בהנחה שהמערך ממוין אפשר למצוא ב2 פעולות - לקחת את הראשון והאחרון במערך.
לכן כדי לבצע בפחות מ***1.5n*** פעולות נוכל לבחור לפי גודל המערך אלגוריתם מיון המתאים לגודל שימיין את המערך ב$\boldsymbol{O(\leq n)}$ פעולות ואז עוד 2 פעולות לקחת ***Min-Max***.

5. א. מהאיטרציה נקבל: $T(n) = 2^i\big(T(n-i)\big) + 2^i - 1$
ומתנאי העצירה: $n - i = 1 \rightarrow \boldsymbol{i = n - 1}$
לכן: $T(n) = 2^{n-1}(2) + 2^{n-1} - 1 = 4 * 2^{n-1} - 1 = 2^{n+1} - 1 = \boldsymbol{O(2^n)}$

ב. מהאיטרציה נקבל: $T(n) = 2^i * T\left(\frac{n}{2^i}\right) + \sum_{k=1}^{i} 2^k$
ומתנאי העצירה: $\frac{n}{2^i} = 1 \rightarrow \boldsymbol{i = log_2 n}$
לכן: $T(n) = 2^{log_2 n} * 1 + 2n - 2 = 3n - 2 = \boldsymbol{O(n)}$

פתרון – אלעזר פיין

6. לפי נוסחת האב הכללית:

$T(n) = 4T(n/3) + n$	$\Theta(n^{log_3 4})$
$T(n) = 4T(n/3) + n^2$	$\Theta(n^2)$
$T(n) = 9T(n/2) + n^2$	$\Theta(n^{log_2 9})$
$T(n) = 8T(n/2) + n^3 \log n$	$\Theta(n^3 logn)$
$T(n) = 2T(n/4) + 2000000$	$\Theta(\sqrt{n})$
$T(n) = 9T(n/3) + n^2 \lg n$	N/A
$T(n) = 3T(n/9) + n^2 \lg n$	$\Theta(n^2 lgn)$
$T(n) = 4T(n/4) + n \lg n$	N/A
$T(n) = 2T(n/4) + \sqrt{n}$	$\Theta(\sqrt{n} * logn)$

7. מכיוון שיש 8 טורים ו8 שורות בכל שורה ובכל טור יכולה להיות רק מלכה אחת, לכן לכל מלכה נקבע שורה משלה ובשורה נשים אותה בטור פנוי שלא נמצא על אלכסון תפוס. (או לחליפין נקבע טור ואז נשים בשורה פנויה) כלומר הבעיה הרקורסיבית הכי קטנה היא לשים מלכה אחת בשורה מסוימת \ טור מסוים. תנאי העצירה הוא יציאה מהלוח, כלומר ניסיון לשים מלכה בטור או שורה שלא נמצאים על הלוח נגיע למצב זה לאחר שנציב את כל המלכות על הלוח.
נגדיר מטריצה 8x8 ונשים את המלכה הראשונה בטור כלשהו בשורה הראשונה, לאחר מכן נעבור לשורה הבאה (המלכה הבאה, צעד רקורסיה) ונעבור על הטורים (לולאה) ונבחר טור ואלכסון פנויים, ונעבור למלכה הבאה עד שנגיע אחרי השורה האחרונה (המלכה האחרונה). כשמגיעים לנקודה זו מדפיסים את הלוח, ואז חוזרים אחורה במחסנית הרקורסיה, ובודקים את השאר ההסתעפויות בעץ.

פתרון – אלעזר פיין

פתרון בKotlin:

הערה: המימוש מגדיר את המטריצה "ישירות" רק בהדפסה, עובד עם רשימת קואורדינטות במקום.

```
fun main() = nQueens()

fun nQueens(n: Int = 8,
            row: Int = 0,
            coordinates: MutableList<Pair<Int, Int>> = ArrayList())
{
    if (row == n) printBoard(n, coordinates) //placed all, now print the board
    else {
        for (column in 0 until n) {
            coordinates.add(Pair(row, column))    // add path in tree
            if (isValid(coordinates.map { it.second })) //check updated columnIndexes
                nQueens(n, row + 1, coordinates)   // explore path with recursion
            coordinates.removeLast()      // in order to explore a different path
        }
    }
}

fun isValid(columnIndexes: List<Int>): Boolean {
    /*
     *  Can't be in the same row, inherit in the way we advance in the recursion,
     *  so we only check for same col or same diagonal by evaluating dx,dy
     *  if in same column distance dx is ofc 0.
     *  if in same diagonal then dx = dy:
     *      for example: (1,4) <-> (3,6) <-> (2,7)
     *          6-4 = 2 = 3-1 -> [(1,4),(3,6)] on same diagonal
     *          7-6 = 1 = 3-2 -> [(3,6),(2,7)] on same diagonal
     *      (basically if both coordinates are on the diagonal then it is the
     *           hypotenuse of a right-angle triangle)
     */
    val currentRowIndex = columnIndexes.size - 1
    for (i in 0 until currentRowIndex) {
        val dx = abs(columnIndexes[i] - columnIndexes[currentRowIndex])
        val dy = currentRowIndex - i
        if (dx == 0 /*same col*/ || dx == dy /*same diag*/) return false
    }
    return true
}

fun printBoard(n: Int, coordinates: List<Pair<Int, Int>>) {
    println(coordinates)
    val matrix = Array(n) { BooleanArray(n) }
    coordinates.forEach { matrix[it.first][it.second] = true }
    matrix.indices.forEach { row ->
        matrix[row].indices.forEach { col ->
            print(if (matrix[row][col]) "  ■" else "  □")
        }
        println()
    }
    println("  ~~~~~~~~~~~~~~~~~~~~~~")
}
```

ניתן להריץ את הקוד אונליין ב: https://pl.kotl.in/k7FCExKLs