1. Example 2.1 of the lectures considered the average complexity of sequential search assuming that the keys are found in the list. Let’s generalize: Assume that the searches are evenly distributed over items of a universe \( \{a_1, \ldots, a_N\} \), out of which the items \( \{a_1, \ldots, a_n\} \) \( (n < N) \) are stored in the list. How many keys of the list does the search examine on the average if (a) \( n = N/2 \) or (b) \( n = \sqrt{N} \)?

2. Solve \( T(n) \), when \( T(0) = 0, T(1) = 1 \), and for \( n \geq 2 \)

(a) \( T(n) = 2T(n-1) - T(n-2) \);
(b) \( T(n) = 3T(n-1) - 2T(n-2) \).

Hint: (i) Examine small values \( T(0), T(1), \ldots, T(5) \); (ii) Observe, how \( T(n) \) differs from \( T(n-1) \), expand \( T(n) \) based on this recurrence, and simplify the resulting sum. (This is not necessary in case (a), which is quite simple.) (iii) Verify the result using induction.

3. Figure out recurrences for describing the time complexities of the below functions. Use the recurrences to analyze the time complexity of each function with respect to the number of items in array \( A[i..j] \). (\( i \leq j \). An asymptotic \( O() \) or \( \Theta() \) estimate is sufficient.)

```plaintext
function funA(A: array of int; i, j: int) returns int:
    m := 0;
    for k := i to j do m := m + A[k];
    if i = j then return m;
    else // i < j
        return m + funA(A, i + 1, j);

function funB(A: array of int; i, j: int) returns int:
    if i = j then return A[i];
    else // i < j
        return A[i] + funB(A, i, j - 1) + funB(A, i + 1, j);
```
function funC(A: array of int; i, j: int) returns int:
    if i = j then return A[i];
    else // i < j
        m := ⌊(i + j)/2⌋;
        for k := 0 to m - i do B[k] := A[j - k];
        return A[i] + funC(A, i, m) + funC(A, m + 1, j) + funC(B, 0, m - i);

function funD(A: array of int; i, j: int) returns int:
    if i = j then return A[i];
    else
        for k := i to j do
            m := ⌊(i + j)/2⌋;
            a := funD(A, i, m); b := funD(A, m + 1, j);
            c := funD(A, i, m); d := funD(A, m + 1, j);
        return a + b + c + d;

4. How many character comparisons are performed by naive string pattern matching to locate occurrences of the pattern (a) ”00001”, (b) ”10000” or (c) ”01010” inside a text that consists of thousand '0' characters?

5. Simulate how (a) merge sort and (b) quicksort would order the letters of the word ”AUTORATA” in increasing alphabetic order. (In quicksort, use the first item as the pivot, and partition the keys in sublists according to their order in the original list.)

6. A pair (A[i], A[j]) of items is an inversion, if the items appear in the array incorrectly ordered concerning sorting, that is, if i < j and A[i] > A[j]. We want to compute the number of inversions in array A[1..n]. Present
   (a) a ”brute force” solution whose time complexity is O(n^2);
   (b) a ”divide-and-conquer” solution whose time complexity is O(n log n).
   (Hint: modify merge sort.)