Esercizio. Si consideri un array $A[1..n]$ composto da $n \geq 1$ interi (negativi oppure positivi) distinti ordinati in senso crescente ($A[1] < A[2] < \cdots < A[n]$). Scrivere un algoritmo efficiente che, dato in input l'array A, determina un indice i, se esiste, tale che $A[i] = i$. Nel caso esistano più indici che soddisfano la relazione precedente, è sufficiente restituirne uno qualsiasi. Determinare il costo computazionale dell'algoritmo.

Soluzione. È possibile utilizzare il seguente algoritmo ricorsivo, molto simile a quello della ricerca binaria (i e j rappresentano rispettivamente gli indici estremi del sottovettore $A[i..j]$ in cui effettuare la ricerca, e all'inizio la funzione va invocata con $i = 1$, $j = n$):

Algorithm 1: PUNTOFISSO(*Intero* $A[1..n]$, *Intero* i, *Intero* j) $\rightarrow$ *Intero*

```
if i > j then
    return −1
else
    Intero m = Floor((i + j)/2)
    if A[m] = m then
        return m
    else if A[m] > m then
        return PuntoFisso(A, i, m-1)
    else
        return PuntoFisso(A, m+1, j)
```

Questo algoritmo si basa sull'osservazione che se nella posizione media m si trova un valore $A[m] > m$, tutte le posizioni successive $k > m$ saranno tali che $A[k] > k$. Questo vale in quanto ogni valore dell'array risulta essere più grande del precedente di almeno una unità, quindi i valori dell'array crescono almeno quanto i corrispettivi indici nell'array. Simmetricamente, se $A[m] < m$, tutte le posizioni precedenti $k < m$ saranno tali che $A[k] < k$. L'algoritmo proposto è una semplice variante dell'algoritmo di ricerca binaria e ha lo stesso costo computazionale $T(n) = O(\log n)$.

Esercizio. Si consideri un array $A[1..n]$ contenente valori reali ordinati in senso non decrescente; l'array può contenere valori duplicati. Scrivere un algoritmo ricorsivo di tipo divide-et-impera che, dato A e due valori reali $low < up$, calcola quanti valori di A appartengono all'intervallo $[low, up]$. Determinare il costo computazionale dell'algoritmo proposto.

Soluzione. Si può procedere con il seguente algoritmo ricorsivo di tipo divide-et-impera che evita di fare il passo ricorsivo se è possibile verificare immediatamente che tutti gli elementi del sottovettore $A[i..j]$ ricadono ($A[i] \geq low$ *and* $A[j] \leq up$) oppure non ricadono ($A[i] > up$ *or* $A[j] < low$) nell'intervallo $[low, up]$:

Algorithm 2: CONTAINTERVALLO(*Reale* $A[1..n]$, *Reale* low, *Reale* up, *Intero* i, *Intero* j) $\rightarrow$ *Intero*

```
if i > j then
    return 0
else if A[i] ≥ low and A[j] ≤ up then
    return j − i + 1
else if A[i] > up or A[j] < low then
    return 0
else
    Intero m = Floor((i + j)/2)
    return ContaIntervallo(A, low, up, i, m)+ ContaIntervallo(A, low, up, m+1, j)
```

Tale algoritmo viene inizialmente invocato con $i = 1$ e $j = n$. Si consideri ora un certo livello di annidamento delle chiamate ricorsive: solo due istanze della funzione a tale livello (quelle relative a sottovettori $A[i..j]$ che contengono sia valori in $[low, up]$ che fuori) effettueranno le 2 chiamate ricorsive. Quindi ad ogni livello al più 4 istanze vengono eseguite. Il numero massimo di livelli di annidamenti è logaritmico in quanto la lunghezza dei sottovettori si dimezza ad ogni passaggio di livello. Il numero totale di istanze eseguite è quindi logaritmico. Visto che ogni istanza esegue un numero costante di operazioni, si ottiene il costo $T(n) = O(\log n)$.