Algoritmi e Strutture Dati - Prova d'esame
11/01/12

Esercizio 1

La funzione di ricorrenza per MergeSortK è la seguente:

$$T(n) = \begin{cases} k(T(n/4)) + O(n) & n > 1 \\ 1 & n = 1 \end{cases}$$

Infatti, l'operazione MergeK costa $O(kn)$, che per valori costanti di k resta comunque $O(n)$. Quindi il costo totale di MergeSortK è comunque $O(n \log n)$.

Esercizio 2

Parte (i) Utilizziamo la tecnica di backtrack.

```
stampaCombinazioni(char[] V, integer i, integer n, integer m)
    if n = 0 and m = 0 then
        print V
    if n > 0 then
        V[i] ← "R"
        stampaCombinazioni(V, i + 1, n − 1, m)
    if m > 0 then
        V[i] ← "G"
        stampaCombinazioni(V, i + 1, n, m − 1)
```

Parte (ii) Per calcolare il numero di combinazioni, è possibile usare un algoritmo ricorsivo, ma il suo costo computazionale sarebbe esponenziale, in quanti molti sottoproblemi sarebbero risolti più volte.

```
calcolaCombinazioniRic(integer n, integer m)
    if n = 0 or m = 0 then
        return 1
    else
        return calcolaCombinazioniRic(n − 1, m) + calcolaCombinazioniRic(n, m − 1)
```

È meglio quindi usare la programmazione dinamica, utilizzando una tabella $n \times m$ con costo $O(n, m)$. Si noti inoltre che sarebbe possibile sfruttare la simmetria per cui il numero di combinazioni (n, m) è uguale al numero di combinazioni (m, n).

```
calcolaCombinazioni(integer n, integer m)
    integer[][] M ← newinteger[1 . . . n][1 . . . m]
    for i ← 1 to n do  M[i, 0] ← 1
    for j ← 1 to m do  M[0, j] ← 1
    for i ← 1 to n do
        for j ← 1 to n do
            M[i, j] = M[i − 1, j] + M[i, j − 1]
    return M[n, m]
```

Alternativamente, si potrebbe utilizzare memoization.

Esercizio 3

Si ordini il vettore e si considerino le somme degli elementi i e $n - i + 1$, con $1 \leq i \leq n/2$. Se sono tutti uguali, si ritorna **true**, altrimenti si ritorna **false**. L'approccio seguito è greedy. Il costo dell'algoritmo è dominato dall'ordinamento, ed è quindi $O(n \log n)$.

Dimostrazione: supponiamo per assurdo che esista un insieme di coppie che rispetti le condizioni per restituire **true**, in cui l'elemento maggiore M sia associato ad un elemento M' diverso dal minore m ($m < M'$). Quindi il minore m è associato ad un elemento m' diverso dal massimo M ($m' < M$). Allora $m + m' < M + M'$, il che contraddice l'ipotesi che tale insieme di coppie rispetti le condizioni per restituire **true**.
La scelta greedy consiste quindi nel scegliere il minore e il maggiore, e confrontarli con il secondo minore e maggiore, il terzo minore e maggiore, e così via.

```
checkPairs(integer[] A, integer n)
    sort(A, n)
    integer s ← A[1] + A[n]
    for i ← 2 to n/2 − 1 do
        if A[i] + A[n − i + 1] ≠ s then
            return false
    return true
```

Esercizio 4

Questo è simile al problema dello zaino; notate però che le lunghezze possono essere selezionate più volte. Il guadagno massimo per un bastone di lunghezza L è espresso dalla seguente formulazione ricorsiva:

$$\mathit{max}(L) = \begin{cases} \max_{1 \leq t \leq L} G(t) + \mathit{max}(L - t) & L > 0 \\ 0 & L \leq 0 \end{cases}$$

In parole, bisogna guardare cosa succede vendendo un bastone di lunghezza t e poi tagliando un bastone di lunghezza $L - t$, per tutti i possibili t.
La funzione seguente utilizza memoization per calcolare il guadagno massimo. La chiamata iniziale è bestCut$(G, L, \mathit{max}, \mathit{cut})$, dove *max* è la tabella dei guadagni massimi per bastoni di lunghezza L e *cut* è il punto in cui fare il primo taglio per un bastone di lunghezza L.

```
bestCut(integer[] G, integer ℓ, integer[] max, integer[] cut)
    if ℓ ≤ 0 then
        return 0
    if max[ℓ] == nil then
        max[ℓ] ← 0
        for t ← 1 to ℓ do
            integer gain ← G[t] + bestCut(G, t, max, C)
            if gain > max[ℓ] then
                max[ℓ] ← gain
                cut[ℓ] ← t
    return max[ℓ]
```

Il costo della procedura è $O(L^2)$.
Per stampare la lunghezza dei tagli, è sufficiente utilizzare la seguente funzione ricorsiva.

```
printCut(integer[] cut, integer ℓ)
    if ℓ > 0 then
        print C[ℓ]
        printCut(cut, L − C[ℓ])
```