Mergesort con allocazione singola

Realizzare una versione del mergesort nella quale l'array di supporto è allocato staticamente (quindi la procedura sarà del tipo `MergeSort(A,B, ...)` dove `A` è l'array di input, mentre `B` è un array della stessa dimensione utilizzato per la memorizzazione temporanea

- *Versione 1*: l'array `B` è utilizzato per memorizzare temporaneamente le parti di array già ordinate delle quali si deve effettuare il merge;
- *Versione 2*: la procedura Merge fa alternativamente il merge dei sottoarray già ordinati da `A` verso `B` e vice versa, di modo da evitare la fase di copia dei sottoarray dei quali fare il merge.

Soluzione:

Versione 1. In questo caso la soluzione è molto semplice. È sufficiente allocare l'array `B` staticamente. Quindi `B` diventa un argomento della procedura `MergeSort` ricorsiva. Viene usato in `Merge` per contenere temporaneamente i due sottoarray da fondere.

```
# chiamata base:

MergeSort (A)
    n = A.length
    allocate B[1..n]            // alloca l'array B di supporto
    MergeSortRic (A, B, 1, n)


# procedura ricorsiva
MergeSortRic (A, B, p, r)
    if p < r
       q = (p+r)/2
       MergeSortRic (A, B, p, q)
       MergeSortRic (A, B, q+1, r)
       Merge (A, B, p, q, r)



# Merge: usa l'array di supporto per memorizzare i due sottoarray da fondere
Merge (A, B, p, q, r)
    for i = p to q
        B[i] = A[i]
    for j = q+1 to r
        B[j] = A[j]

    i=p
    j=q+1
    for k = p to r
        if (i <= q) and ((j > r) or (B[i] <= B[j])):
            A[k] = B[i]
            i ++
        else:
            A[k] = B[j]
            j ++
}
```

Versione 2. Si definisce una procedura di mergesort `MergeSortAlt(A,B,p,r,destA)` con un parametro addizionale, `destA`, un flag booleano, che indica se la destinazione dell'ordinamento è `A` (quando vero) oppure `B` (quando falso). Occorre dunque avere l'accortezza di negare il flag ad ogni passaggio ricorsivo. Inoltre, nel caso base, se la destinazione dell'ordinamento è `A` non c'è niente da fare, se invece è `B`, l'unico elemento di cui si compone il sottoarray va copiato in `B`. Il resto è immutato rispetto a prima.

```
# chiamata base:

MergeSort (A)
    n = A.length
    allocate B[1..n]            // alloca l'array B di supporto
    MergeSortRic (A, B, 1, n, true)


# MergeSortAlt(A, B, p, r, destA)
# se  destA=true -> ordina  A[p,r] con risultato in A[p,r]
# altrimenti     -> ordina  A[p,r] con risultato in B[p,r]

MergeSortAlt (A, B, p, r, destA)
    if p < r
       q = (p+r)/2
       MergeSortAlt (A, B, p, q, not destA)
       MergeSortAlt (A, B, q+1, r, not destA)
       if destA
            Merge (B, A, p, q, r)
        else
            Merge (A, B, p, q, r)
    else
        if not destA
            B[p]=A[p]


# Merge: fa il merge di S(ource) in T(arget)

Merge (S, T, p, q, r):
    i=p
    j=q+1
    for k = p to r
        if (i <= q) and (not (j <= r) or (S[i] <= S[j]))
            T[k] = S[i]
            i ++
        else:
            T[k] = S[j]
            j ++
}
```