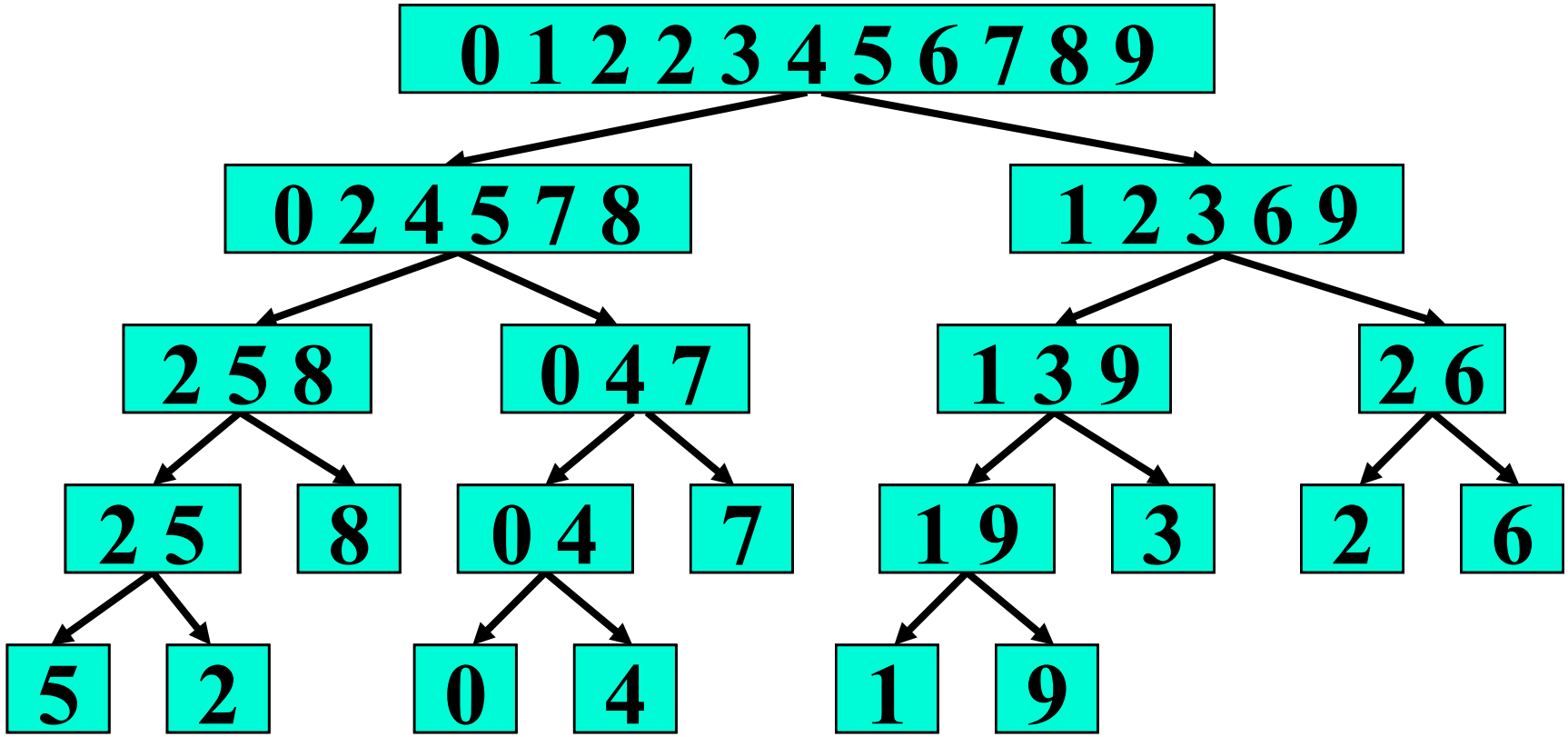
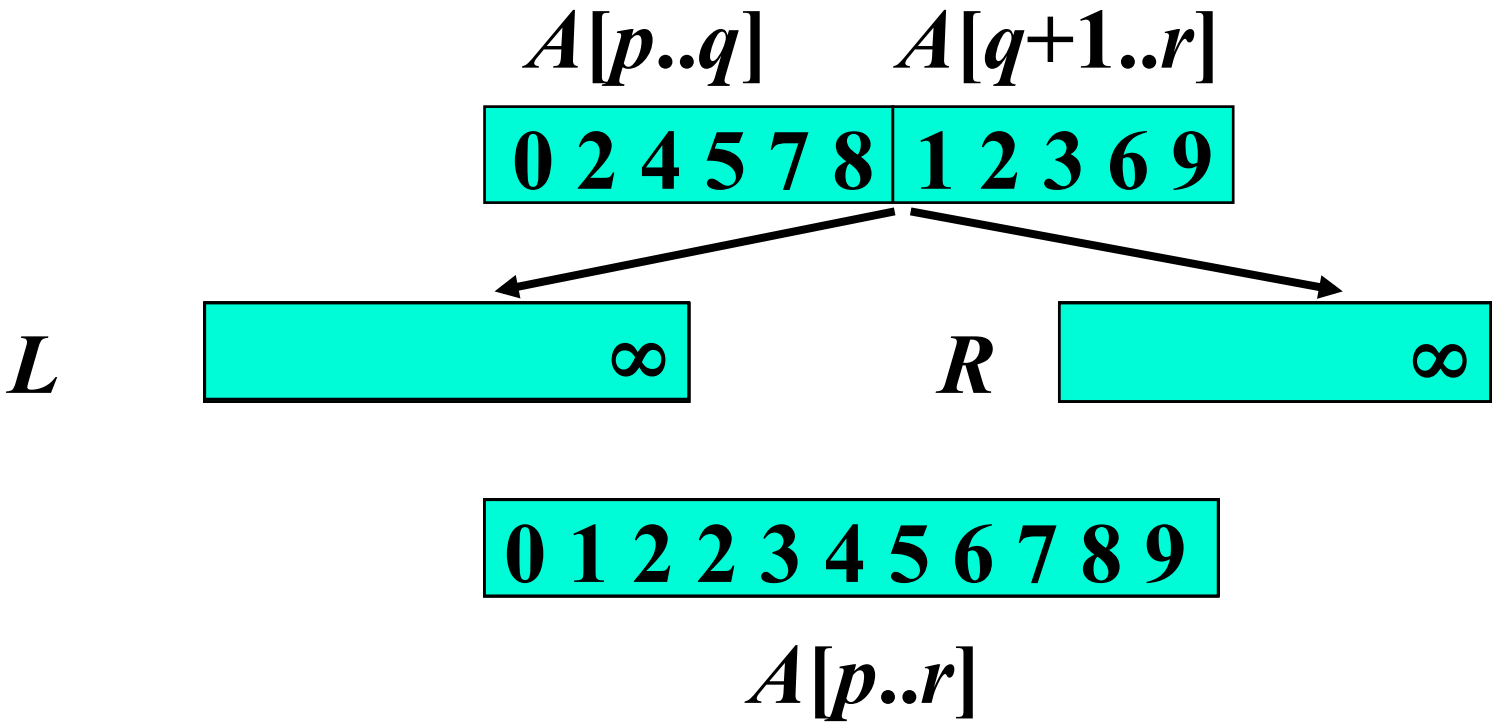
# TECNICA
# DIVIDE ET IMPERA

# Soluzione2: Algoritmo Merge-Sort



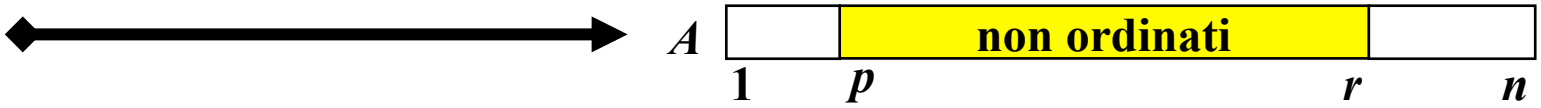
```
Merge-Sort(A,p,r)
    if p < r
        q = ⌊(p+r)/2⌋
        Merge-Sort(A,p,q)
        Merge-Sort(A,q+1,r)
        Merge(A,p,q,r)
```


A[p..q]
A[q+1..r]
0 2 4 5 7 8
1 2 3 6 9
L
∞
R
∞
0 1 2 2 3 4 5 6 7 8 9
A[p..r]

```
Merge(A,p,q,r)
    n1 = q – p + 1
    n2 = r – q
    for i = 1 to n1
        L[i] = A[p + i – 1]
    for j = 1 to n2
        R[j] = A[q + j]
    L[n1 + 1] = R[n2 + 1] = ∞
    i = j = 1
    for k = p to r
        if L[i] ≤ R[j]
            A[k] = L[i]
            i = i + 1
        else
            A[k] = R[j]
            j = j + 1
```

# Analisi di Merge-Sort: correttezza

**Merge-Sort**($A$,$p$,$r$)

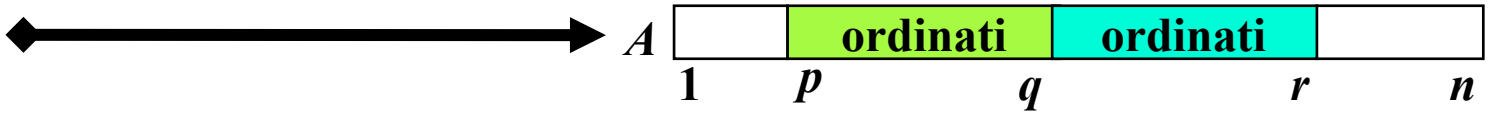



**if** $p < r$ // **altrimenti** $A[p..r]$ **è ordinato**

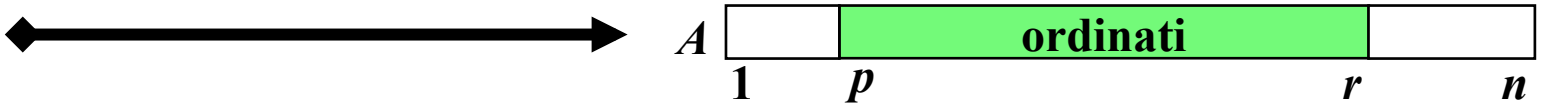
$q = \lfloor (p+r)/2 \rfloor$

**Merge-Sort**($A$,$p$,$q$)

**Merge-Sort**($A$,$q$+1,$r$)



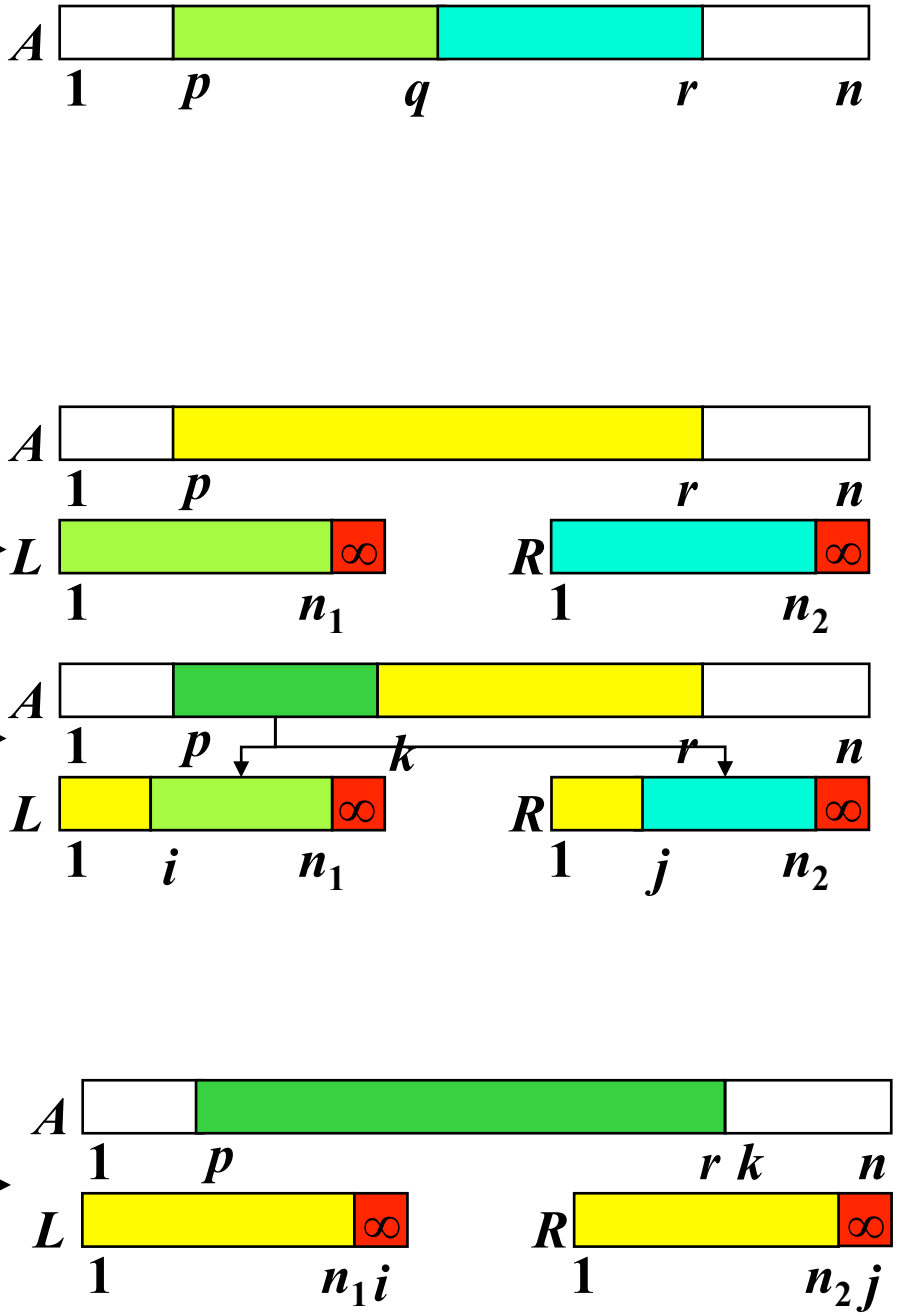

**Merge**($A$,$p$,$q$,$r$)



**Merge**$(A,p,q,r)$

$n_1 = q - p + 1$
$n_2 = r - q$
**for** $i = 1$ **to** $n_1$
  $L[i] = A[p + i - 1]$
**for** $j = 1$ **to** $n_2$
  $R[j] = A[q + j]$
$L[n_1 + 1] = R[n_2 + 1] = \infty$

$i = j = 1$
**for** $k = p$ **to** $r$
  **if** $L[i] \le R[j]$
    $A[k] = L[i]$
    $i = i + 1$
  **else**
    $A[k] = R[j]$
    $j = j + 1$



```
Merge(A,p,q,r)  // complessità
  n1 = q – p + 1
  n2 = r – q
  for i = 1 to n1
    L[i] = A[p + i – 1]
  for j = 1 to n2
    R[j] = A[q + j]
  L[n1 + 1] = R[n2 + 1] = ∞
  i = j = 1
  for k = p to r
    if L[i] ≤ R[j]
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```

| | |
|---|---|
| // | $c_1$ |
| // | $c_2$ |
| // | $c_3$ |
| // | $c_4(n_1+1)$ |
| // | $c_5 n_1$ |
| // | $c_4(n_2+1)$ |
| // | $c_5 n_2$ |
| // | $c_6$ |
| // | $c_7$ |
| // | $c_8(n+1)$ |
| // | $c_9 n$ |
| // | $c_{10} n_1$ |
| // | $c_{11} n_1$ |
| | |
| // | $c_{10} n_2$ |
| // | $c_{11} n_2$ |

$$\boxed{n = r - p + 1}$$

$$\boxed{q = \lfloor (p+r)/2 \rfloor}$$

$$\boxed{n_1 = \lceil n/2 \rceil}$$

$$\boxed{n_2 = \lfloor n/2 \rfloor}$$

$$\boxed{n = n_1 + n_2}$$

$$T^M(n) = (c_4 + c_5 + c_8 + c_9 + c_{10} + c_{11})n$$
$$+ c_1 + c_2 + c_3 + 2c_4 + c_6 + c_7 + c_8$$
$$= a'n + b'$$

```
Merge-Sort(A,p,r)  //complessità
    if p < r                                  // c1
        q = ⌊(p+r)/2⌋                         // c2
        Merge-Sort(A,p,q)                     // c3
        Merge-Sort(A,q+1,r)                   // T^MS(n1)
        Merge(A,p,q,r)                        // T^MS(n2)
                                              // T^M(n) = a'n + b'
```

$$\boxed{n = r - p + 1}$$

$$\boxed{n_1 = q - p + 1}$$

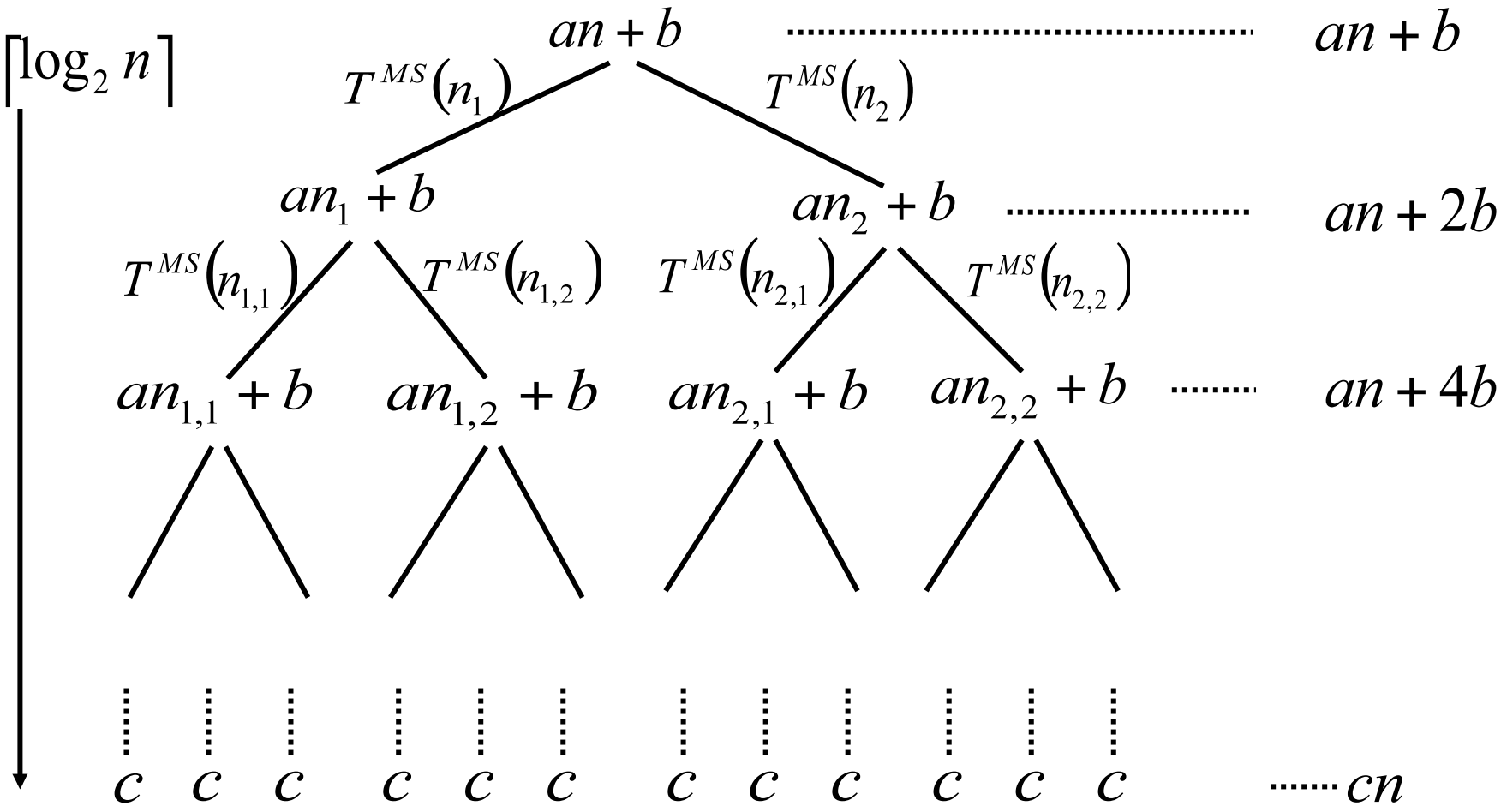
$$\boxed{n_2 = r - q}$$

$$\boxed{n = n_1 + n_2}$$

$$T^{MS}(n) = \begin{cases} c_1 + c_2 & \text{se } n \le 1 \\ c_1 + c_2 + c_3 + T^{MS}(n_1) + T^{MS}(n_2) + T^{M}(n) & \text{se } n > 1 \end{cases}$$

$$\boxed{T^{MS}(n) = \begin{cases} c & \text{se } n \le 1 \\ an + b + T^{MS}(n_1) + T^{MS}(n_2) & \text{se } n > 1 \end{cases}}$$

$$T^{MS}(n) = \begin{cases} c & \text{se } n \le 1 \\ an + b + T^{MS}(n_1) + T^{MS}(n_2) & \text{se } n > 1 \end{cases}$$




$$T^{MS}(n) \cong a' n \lceil \log_2 n \rceil + b' n + c'(n-1) + cn$$

$$T^{MS}(n) \cong a' n \log_2 n + b' n + c'$$

$$T^{IS}_{med}(n) = a'' n^2 + b'' n + c''$$

$$\boxed{\lim_{n\to\infty} \frac{T^{MS}(n)}{T^{IS}_{med}(n)} = \lim_{n\to\infty} \frac{a' n \log_2 n + b' n + c'}{a'' n^2 + b'' n + c''} = 0}$$

Dunque esiste ***N*** tale che $T^{MS}(n) < T^{IS}_{med}(n)$ per ogni ***n* > *N*.**

Qualunque siano i valori delle costanti ***a'***, ***b'***, ***c'***, ***a''***, ***b''*** e ***c''*** l'algoritmo Merge-Sort è superiore a Insertion-Sort per array di dimensione sufficientemente grande.

Possiamo dire che $T_{med}^{IS}(n)$ "*cresce come*" $n^2$ mentre $T^{MS}(n)$ "*cresce come*" $n \log_2 n$.

| $n$ | $n^2$ | $n \log_2 n$ | ***IS*** $n^2$ ns | ***MS*** $n \log_2 n$ ns |
|---|---|---|---|---|
| **10** | **100** | **33** | **0.1μs** | **0.033μs** |
| **100** | **10000** | **664** | **10μs** | **0.664μs** |
| **1000** | $10^6$ | **9965** | **1ms** | **10μs** |
| **10000** | $10^8$ | **132877** | **0.1s** | **133μs** |
| $10^6$ | $10^{12}$ | $2 \cdot 10^7$ | **17m** | **20ms** |
| $10^9$ | $10^{18}$ | $3 \cdot 10^{10}$ | **70anni** | **30s** |

$$\boxed{\lim_{n\to\infty} \frac{T^{MS}(n)}{T^{IS}_{min}(n)} = \lim_{n\to\infty} \frac{a' n \log_2 n + b' n + c'}{a'' n + b''} = \infty}$$

dunque esiste $N$ tale che $T^{MS}(n) > T^{IS}_{min}(n)$ per ogni $n > N$.

Qualunque siano i valori delle costanti $a'$, $b'$, $c'$, $a''$, $b''$ l'algoritmo Insertion-Sort è superiore a Merge-Sort per array (quasi) ordinati e sufficientemente grandi.

Insertion-Sort è anche superiore a Merge-Sort per array piccoli in quanto le *costanti* $a'$, $b'$, $c'$ in $T^{MS}(n)$ sono generalmente molto maggiori delle costanti $a''$, $b''$ e $c''$ in $T_{max}^{IS}(n)$.

Questo suggerisce una modifica di Merge-Sort in cui le porzioni di array di dimensione minore di una certa costante $k$ si ordinano con Insertion-Sort invece di usare ricorsivamente Merge-Sort.

# Soluzione3: Algoritmo Merge-Ins-Sort

```
Merge-Ins-Sort(A,p,r)
  if p < r
    if r-p+1 < 32
      InsertSort(A,p,r)
    else
      q = ⌊(p+r)/2⌋
      Merge-Ins-Sort(A,p,q)
      Merge-Ins-Sort(A,q+1,r)
      Merge(A,p,q,r)
```