QUICK SORT

- Si seleziona un elemento dell'array = PERNO attorno al quale si riorganizzano gli altri elementi in modo che i più piccoli lo precedano e i più grandi lo succedano.
- Si ripete a SX e a DX (due chiamate ricorsive)

```
QUICK-SORT(A[1..n])
if n > 1 then
    p ← PARTITION(A[1..n])        ▷ partizionamento porta il perno nella posizione p
    if p > 2 then                 ▷ se prima del perno ci sono almeno 2 elementi
        QUICK-SORT(A[1..p − 1])
    end if
    if p < n − 1 then             ▷ se dopo il perno ci sono almeno 2 elementi
        QUICK-SORT(A[p + 1..n])
    end if
end if
```

$T(n) \in \Theta(n^2)$

```
PARTITION(A[1..n])
i ← 2, j ← n
while i ≤ j do
    if A[i] ≤ A[1] then
        i ← i + 1
    else
        if A[j] > A[1] then
            j ← j − 1
        else
            scambia A[i] con A[j]
            i ← i + 1, j ← j − 1
        end if
    end if
end while
scambia A[1] con A[j]
return j
```

→ Questo non va bene per il progetto !

$T(n) = an$

QUICK SORT **QUADRATICO NEL CASO PEGGIORE**

$\Theta(n \log n)$ NEL CASO MEDIO E MIGLIORE

DIVIDE ET IMPERA

- Dividere ricorsivamente il problema in due o piu' sottoproblemi, sino a che questi non diventano di risoluzione banale
- Si combinano le soluzioni per ottenere la soluzione al problema di partenza.

```
DEI(P, n)        ▷ n è la dimensione del problema P
if n ≤ k then
    r ← soluzione diretta del problema
    return r
else
    dividi il problema in sotto-problemi P_i, ..., P_h di dimensione n_1, ..., n_h
    for i ← 1 to h do
        r_i ← DEI(P_i, n_i)
    end for
    return combinazione di r_1, ..., r_h
end if
```

Esempio: **RICERCA DICOTOMICA**

$T(n) \in \Theta(\log n)$

```
BINSEARCH-RIC(x, A, i, j)
    ▷ Pre: A[i..j] ordinato
    ▷ Post: true se x ∈ A[i..j]
if i > j then
    return false
else
    m ← ⌊(i + j)/2⌋
    if x = A[m] then
        return true
    else
        if x < A[m] then
            return BINSEARCH-RIC(x, A, i, m − 1)
        else      ▷ A[m] < x
            return BINSEARCH-RIC(x, A, m + 1, j)
        end if
    end if
end if
```

MERGE SORT

Ordinamento per fusione:

- vettore unitario = ordinato
- se $|A|>1$ => diviso in due parti bilanciate

↳ ogni parte viene ordinata applicando ricorsivamente l'algoritmo

↳ poi vengono fuse

```
MERGE-SORT(A[1..n])
if n > 1 then
    m ← ⌊n/2⌋
    MERGE-SORT(A[1..m])
    MERGE-SORT(A[m+1..n])
    MERGE(A, 1, m, n)        FUSIONE ⟶ Θ(n)
end if
```

VERSIONE ITERATIVA

```
MERGE(A, f, m, l)
i, j, k ← f, m+1, 1
while i ≤ m ∧ j ≤ l do
    if A[i] ≤ A[j] then
        B[k] ← A[i]
        i ← i+1
    else
        B[k] ← A[j]
        j ← j+1
    end if
    k ← k+1
end while
if i ≤ m then
    B[k..k+m−i] ← A[i..m]
else
    B[k..k+l−j] ← A[j..l]
end if
A[f, l] ← B[1..l−f+1]
```

VERSIONE RICORSIVA

```
MERGE(B[1..n_B], C[1..n_C])
if n_B = 0 then
    return C
else
    if n_C = 0 then
        return B
    else
        if B[1] ≤ C[1] then
            return [B[1], MERGE(B[2..n_B], C)]
        else
            return [C[1], MERGE(B, C[2..n_C])]
        end if
    end if
end if
```

MERGE SORT ha $T(n) \in \Theta(n \log n)$

MERGE preferibile a QUICK su array molto grandi