inversion Count Divide Conquer(A)

// A[0..n-1] array of distinct numbers size n

// inversion means A[i] > A[j] for i < j

count ← 0

count Inversion(0, n-1, count)

count Inversion (s, e, count)

If s == e-1 return

m = ⌊(s+e)/2⌋

count Inversion(s, m, count)

Count Inversion(m, e, count)

merge (s, m, e, count)

merge (s, m, e, count) // m..e is right sub array // l..m is left sub array

l ← s, r ← m, k ← 0

sorted Subarray [0... e-s]

while (l < m and r < e) do

if A[l] > A[r]

sorted Subarray [k] = A[l]

r ← r+1

count ← count + m-l

all items to the right of current item A[l] is greater than A[r]

else

sorted Subarray [k] = A[r]

l ← l+1

k ← k+1

if l < m do

copy remaining m-l items from left sub array

else

copy remaining e-r item from right sub array

copy all items from sorted Subarray to A[s..e]

⊛ The algorithm is very similar to merge sort with additional logic added to count inversions

Best case analysis

Basic operation: comparison

Best case is when largest item of one subarray is smaller than smallest item of other subarray → $\frac{n}{2}$ comparisons will be needed

$C(n) = 2C(\frac{n}{2}) + \frac{n}{2}, \; C(1) = 0$

$n = 2^k$

$C(n) = 2[2C(n/4) + n/4] + n/2$
$= 2^2 C(n/4) + n/2 + n/2$
$= 2^2 C(n/4) + 2 \cdot n/2$

$C(n) = 2^2[2C(n/8) + n/8] + 2 \cdot n/2$
$= 2^3 C(n/8) + n/2 + 2 \cdot n/2$
$= 2^3 C(n/8) + 3 \, n/2$

$C(n) = 2^i \, C(n/2^i) + i \cdot n/2$

$i = k$

$C(n) = 2^k \overset{0}{C(1)} + k \cdot n/2$
$= \frac{n \cdot \log_2 n}{2} \quad [\because n = 2^k]$
$\in \Theta(n \log n)$

Using master Theorem

$C(n) = 2C(n/2) + \frac{n^{1 \to d}}{2}$

a, b

$a = 2, \; b^d = 2^1 = 2$

$\therefore a = b^d$. so $\Theta \in (n^d \log n)$
$= \Theta(n \log n)$

master thm says:

$a < b^d, \; T(n) \in \Theta(n^d)$

$a = b^d, \; T(n) \in \Theta(n^d \log n)$

$a > b^d, \; T(n) \in \Theta(n^{\log_b a})$

The master theorem verifies analysis