**Bài 4**

**a)Dùng phương pháp vét cạn (bruce force), tức là xem xét tất cả các cặp (i, j) có thể có và kiểm tra xem nó có phải là một nghịch thế. Cho biết độ phức tạp thời gian T(n) nếu thực hiện theo phương pháp này**

BruteForceCount(a)

count = 0

for i từ 0 đến n - 2:

for j từ i + 1 đến n - 1:

if a[i] > a[j]:

count = count + 1

trả về count

**b)Dùng phương pháp chia để trị theo ý tưởng sau và cho biết độ phức tạp thời gian T(n) nếu thực hiện theo phương pháp này.**

MergeAndCount(aL, aR)

count = 0

i = 0, j = 0

while i < độ dài của aL và j < độ dài của aR:

if aL[i] > aR[j]:

count = count + (độ dài của aL - i)

j = j + 1

else:

i = i + 1

trả về count

DivideAndCount(a)

if kích thước của a == 1:

trả về 0 và a

mid = kích thước của a / 2

aL là mảng từ a[0] đến a[mid - 1]

aR là mảng từ a[mid] đến a[kích thước của a - 1]

(nL, sorted\_aL) = DivideAndCount(aL)

(nR, sorted\_aR) = DivideAndCount(aR)

nLR = MergeAndCount(sorted\_aL, sorted\_aR)

merged là mảng kích thước của a

merged = merge(sorted\_aL, sorted\_aR)

trả về nL + nR + nLR và merged

ĐỘ PHỨC TẠP :

Độ phức tạp thời gian của phương pháp chia để trị này phụ thuộc vào bước gộp kết quả. Trong trường hợp tốt nhất, khi mảng đã được sắp xếp hoàn toàn, độ phức tạp sẽ là O(n). Tuy nhiên, trong trường hợp xấu nhất, khi mảng đã được sắp xếp ngược lại, độ phức tạp có thể là O(n^2). Điều này xảy ra khi ta phải kiểm tra mọi cặp giữa các phần tử trong hai nửa mảng, đồng thời số lượng nghịch thế trên biên giữa hai nửa mảng là lớn nhất.