Design and Analysis of Algorithm

Practice Problem Sheet - 2

PPS2

Name: Khushi Agarwal

Regno: 21BRS1529

(2)

Problem statement:

Input: Array of Integers, A.

Output: Maximum Integer in array.

Logic

Divide and Conquer

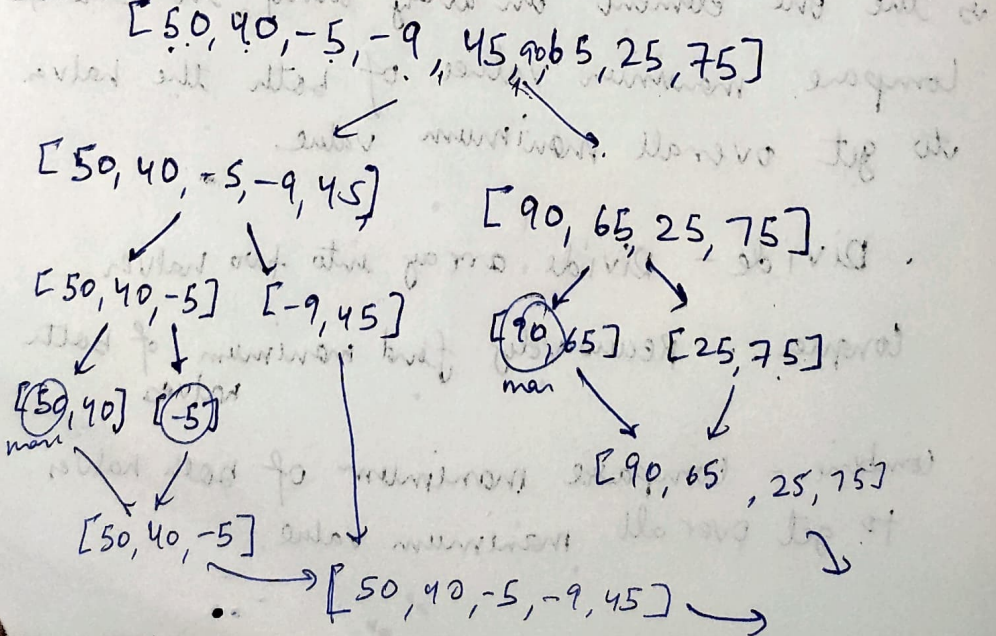
We divide the array into two halves of equal size or almost equal size ~~again~~ till we reach a subarray of length 1 then maximum element is the one element in array itself, ~~in case of~~ Compare maximum values of both the halves to get overall maximum value.

- Divide - Divide array into two halves
- Conquer - Recursively find maximum of both halves
- Combine - Compare maximum of both halves to get overall maximum value.

Pseudocode

```
Algorithm Max(i, j, max)
    if (i=j)
        max = A[i]
    else if i=j-1 then
        if A[i] < A[j] then
            max = A[j]
        else
            max = A[i]
    else
        mid = (i+j)/2
        Max(i, mid, max)
        max(mid+1, j, max)
    if ( max < max_new then
        max = max_new
```

Illustration



[50, 40, -5, -7, 45, 90, 15, 25, 75]

max

Recurrence Relation

$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ 2T\left(\frac{n}{2}\right)+1 & n>2 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right)+1$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right)+1$$

$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right)+1$$

$$T(n) = 2\left[2T\left(\frac{n}{2^2}\right)+1\right]+1$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2+1$$

$$= 2^2\left(2T\left(\frac{n}{2^3}\right)+1\right) + 2+1$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^2 + 2+1$$

$$= 2^K T\left(\frac{n}{2^K}\right) + 2^{K-1} + 2^{K-2} + \cdots 1$$

Assume $\frac{n}{2^K} = 2 \quad n = 2^{K+1}$

$\log_2 n = K+1 \quad K = \log_2 n - 1$

$$T(n) = 2^k + 2^{k-1} + 2^{k-2} + \cdots + 1$$

$$= \frac{1\,(2^{k+1} - 1)}{(2-1)} = 2^{k+1} - 1$$

$$\boxed{T(n) = n - 1}$$

$O(n) \rightarrow$ is the time complexity.

(6) Input: An n-element array of numbers.

Output: Inversion count of array.

Logic

Inversion count for an array indicates how close the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if the array is sorted in reverse order, inversion count is maximum.

Example:

Input: A[] = [3, 2, 1]

Output: 3

Explaination:

The three pair of inversions are —

(3, 2), (3, 1), (2, 1)

Approach 1: ~~Merge Sort~~ Divide and Conquer

Divide the array into two parts. For each left and right half count the inversions

Pseudocode

```
COUNT-INVERSIONS (A, p, r)
1  if p ≥ r
2      return 0
3  q = ⌊(p+r)/2⌋
4  left = Count-INVERSIONS (A, p, q)
   right = Count-Inversions (A, q+1, r)
   Inversions = left + right + Merge (A, p, q, r)
   return Inversions

MERGE (A, p, q, r)
  n1 = q - p + 1
  n2 = r - q
  let L[1...n1] and R[1...n2] be new
    arrays.
  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 = 1
   j = 1
  inversions = 0
  for k = p to r
        if L[i] ≤ R[j]
            A[k] = L[i]
            i = i + 1
        else
```

inversions = inversions + (n1 - i + 1)

A[k] = R[j]

j = j + 1

return inversions

Time complexity: O(N log N)

Space complexity: O(N)

The more the number of inversions in an array, the more times the inner loop will run.

Worst case - Scenario

The numbers will be totally inverted, we would have list as [6, 5, 4, 3, 2, 1]

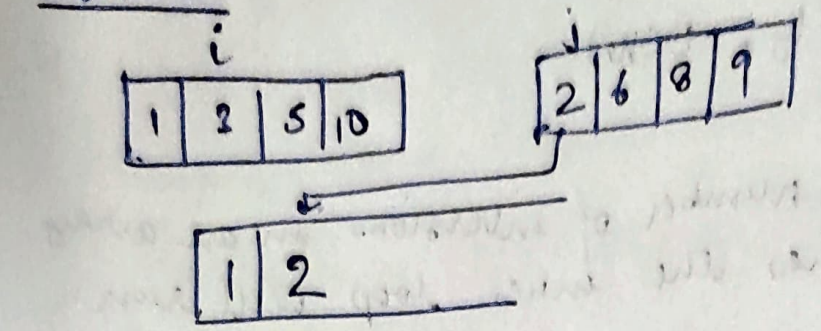
The total number of inversions will be:

$\frac{n(n-1)}{2} = O(n^2)$

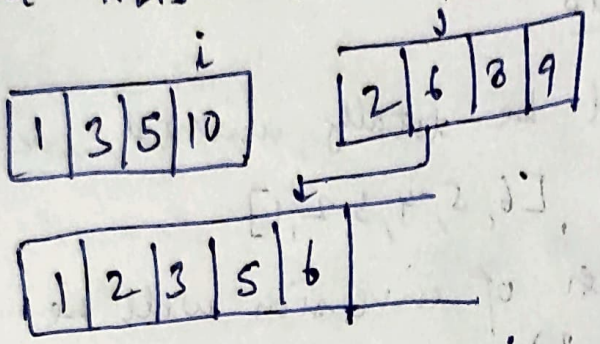
Best case

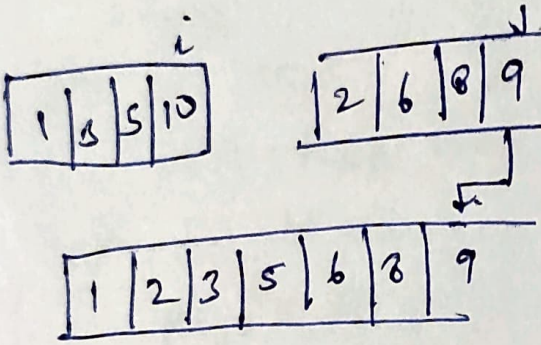
O(nlogn)

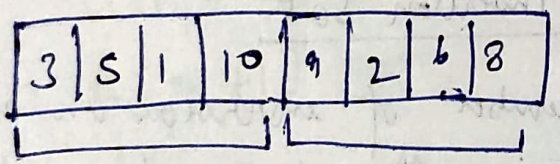
Illustration

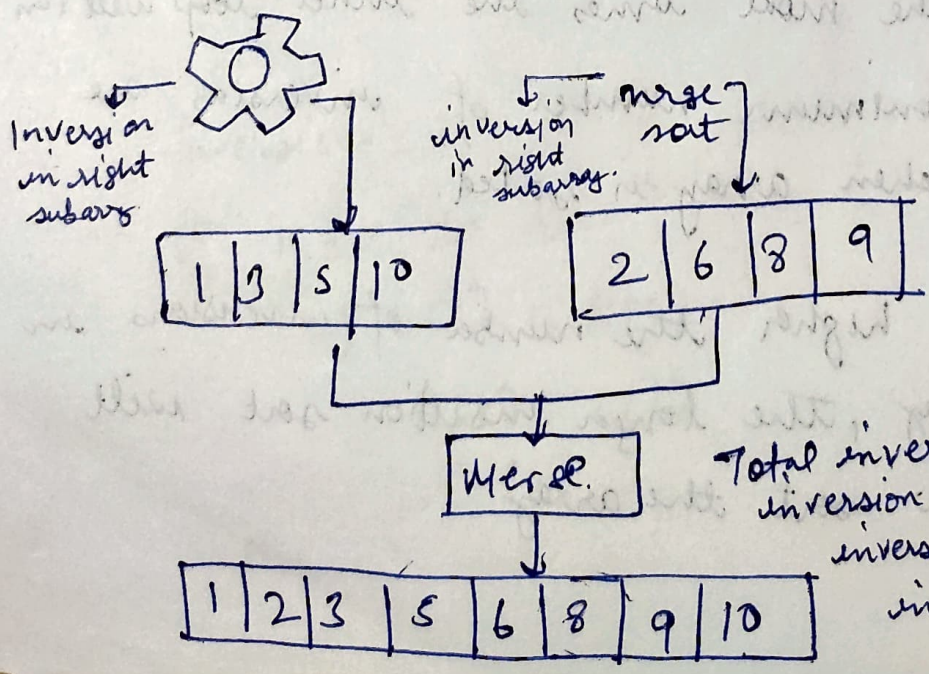


Since A[i] > A[j]









7) Problem Statement (Strassen's Matrix Multiplication)

To compute the product of two matrices, we can use Divide and Conquer (DCC) strategy. The DCC strategy involves dividing the problem into smaller subproblems, solving each subproblem independently, and then combining the solutions of the subproblem to obtain the solution of the original problem.

Input: 2 square matrices of equal size.

Output: Product of these 2 matrices.

Logic

The Divide and Conquer approach for matrix multiplication can be applied recursively as follows:

1. Divide each matrix into four equal sized submatrices.
2. Compute the products of the submatrices recursively using the same algorithm, until the submatrices become small enough to be multiplied using a standard algorithm.
3. Combine the submatrices to obtain the product of the original matrices.

Pseudocode

```
Algorithm MM (A, B, n)
    if (n <= 2) {
```

$C_{11} = (A_{11} \times B_{11}) + (A_{12} * B_{21})$

$C_{12} = (A_{11} \times B_{12}) + (A_{12} * B_{22})$

$C_{21} = (A_{21} \times B_{11}) + (A_{22} * B_{21})$

$C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})$

}

else.

$mid = \frac{n}{2}$.

$MM(A_{11}, B_{11}, \frac{n}{2}) + MM(A_{12}, B_{21}, \frac{n}{2})$

$MM(A_{11}, B_{12}, \frac{n}{2}) + MM(A_{12}, B_{22}, \frac{n}{2})$

$MM(A_{21}, B_{11}, \frac{n}{2}) + MM(A_{22}, B_{21}, \frac{n}{2})$

$MM(A_{21}, B_{12}, \frac{n}{2}) + MM(A_{22}, B_{22}, \frac{n}{2})$

}}.

Recurrence Relation.

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 8T(\frac{n}{2}) + n^2 & n > 2 \end{cases}$$

By Master's Theorem

$a = 8 \quad b = 2 \quad f(n) = n^2$

$\log_b^a = \log_2^8 = 3$

$n^k = n^2$

Case 1 ~~O(n^2)~~ $O(n^3)$

In the above divide DAC, the main compant of high time complexity is 8 recursive calls. The idea of strassen's method is to reduce the number of recursive calls to 7.

$$P1 = a(f-h) \qquad P2 = (a+b)h$$

$$P3 = (c+d)e \qquad P4 = d(g-e)$$

$$P5 = (a+d)(e+h) \qquad P6 = (b-d)(g+h)$$

$$P7 = (a-c)(e+f)$$

The AXB can be calculated using 7 multiplication.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} =$$

A B

$$\begin{bmatrix} P5+P4+P2+P6 & P1+P2 \\ P3+P4 & P1+P5-P3-P7 \end{bmatrix}$$

C

$$T(n) = 7T(n/2) + O(n^2)$$

Time complexity $O(n^{\log 7}) \simeq O(n^{2.8074})$

Illustration

Sums

$$\underset{A}{\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix}} \times \underset{B}{\begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}}$$

$S_1 = B_{12} - B_{22} = 8 - 2 = 6$

$S_2 = A_{11} + A_{12} = 1 + 3 = 4$

$S_3 = A_{21} + A_{22} = 7 + 5 = 12$

$S_4 = B_{21} - B_{11} = 4 - 6 = -2$

$S_5 = A_{11} + A_{22} = 1 + 5 = 6$

$S_6 = B_{11} + B_{22} = 6 + 2 = 8$

$S_7 = A_{12} - A_{22} = 3 - 5 = -2$

$S_8 = B_{21} + B_{22} = 4 + 2 = 6$

$S_9 = A_{11} - A_{21} = 1 - 7 = -6$

$S_{10} = B_{11} + B_{12} = 6 + 8 = 14$

Products

$P_1 = A_{11} \cdot S_1 = 6$

$P_2 = S_2 \cdot B_{22} = 8$

$P_3 = S_3 \cdot B_{11} = 72$

$P_4 = A_{22} \cdot S_4 = -10$

$P_5 = S_5 \cdot S_6 = 48$

$P_6 = S_7 \cdot S_8 = -12$

$P_7 = S_9 \cdot S_{10} = -84$

The result sub matrix sums

$$C_{11} = P_5 + P_4 - P_2 + P_6 = 18.$$

$$C_{12} = P_1 + P_2 = 14$$

$$C_{21} = P_3 + P_4 = 62$$

$$C_{22} = P_5 + P_1 - P_3 - P_7 = 66$$

Ans $\rightarrow$ $\begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}$