Assignment 2 Psuedocode

February 9, 2020 1:19 PM

Q1.1) Brute Force for Inversions

Algorithm BurteInversions(A[0....n-1])

// Counts number of inversions
// Input: Array with numbers A[0...n-1]

//Output: Number of inversions

numInversions ← 0

for i ←0 to n-1 do
  for j←i+1 to n-1 do

    if A[i] > A[j]
      numInversions++

The basic operation is comparision.

$$C(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 \in O(n^2)$$

This algorithm has an efficiency class of $O(n^2)$.

Q1.2) Divide and Conquer version of inversions

Algorithm InversionMerge (A[0...n-1])

// counts number of inversions
// Input: Array of size n.

//Output: Number of Inversions
  num ← 0
if n > 1

  copy A[0....(n-1/2)] to B[0...(n-1/2)]

  copy A[(n-1/2)....n-1] to C[0....(n-1/2)]

num ← InversionMerge(B[0...(n-1/2)]) + num

num ← InversionMerge(C[0...(n-1/2)]) + num

num ← Merge(B, C, A) + num

  return num

Algorithm Merge(B[0...p-1], C[0...q-1], A[0....p+q-1])

// Merge two sorted arrays
// Input: 2 sorted arrays B and C
//Output: Sorted array A

Analysis using masters theorem,

Best case

```
// Input: 2 sorted arrays B and C
// Output: Sorted array A

i ← 0 ; j ← 0 ; k ← 0 ; num ← 0

while i < p and j < q do
  if B[i] ≤ C[j]
      A[k] ← B[i];
      i++
  else
      A[k] ← C[j];
      num ← num + (p - i)
      j++
  k++

if i = p
    copy C[j ... q-1] to A[k ... p+q-1]
else
    copy B[i ... p-1] to A[k .. p+q-1]

return num
```

Q2.1) Brute Force Convex Hull

```
Algorithm BruteConvex(A[0....n-1])

// Get points in convex hull
// Input: Array with points (x,y) with size n

// Output: ConvexHull Points

  convexHullPoints ← [];

  for i ← 0 to n-1 do
     for j ← i+1 to n-1 do
         sign ← 0; validPoint ← True
         for k ← 0 to k ← n-1 do
             if (A[i] = A[k] || A[j] = A[k])
               continue
             currentSign ← getSign(A[i], A[j], A[k])

             if (! sameSign(sign, currentSign)
                validPoint ← False

         if (validPoint)
             addToConvexHull(convexHullPoints, A[i], A[j])

  return convexHullPoints
```

Here basic operation is comparision from checking if two numbers have the same sign.

$$c(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \sum_{k=0}^{n-2} 1 \in O(n^3)$$

Best case

$T(n) = 2T(n/2) + n/2$ for $n>1$ $n=1, T(1)=0$

From master's theorem if $a = b^d$, then $T(n) \in O(n^d \log(n))$

$\Rightarrow a=2 \quad b=2 \quad d=1 \quad \Rightarrow 2 = 2^1$

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

Worst Case:

$T(n) = 2T(n/2) + n$ for $n>1$, $T(1)=0$

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

Q 2.2) QuickHull

```
Algorithm ConvexHull ( A[0....n-1], C[])
// Get points in convex hull
// Input: Array with points (x,y) with size n
//Output: ConvexHull Points

    p1 ← getMinPoint (A[0....n-1])
    p2 ← getMaxPoint (A[0....n-1])
    addToConvexHull (p1, p2)
    B[0....n-1]; C[0....n-1];
    p ←0 ; k ← 0
    for i ←0 to n-1 do
        if ( getSign( p1, p2 , A[i]) ≥0)
            B[p] = A[i]
            p++
        else
            C[k] = A[i]
              k++
    QuickHull ( B[0....p-1], p1, p2)
    QuickHull ( C[0...k-1] , p1, p2)

Algorithm QuickHull ( B[0....p-1], p1, p2)
    p3; distance ← 0;
    for i ←0 to p-1 do
        currentDistance ← distance(p1,p2, B[i])
        if currentDistance > distance
            distance ← currentDistance
            p3 ← B[i]
    addToConvexHull (p3)
    L[0...p-1] ; M[0...p-1];
      k←0; q←0;
    s1 ← getSign ( p1, p3, p2)
    for i ←0 to p-1 do
        s2 ← getSign ( p1, p3, B[i])
        if (!sameSign(s1, s2))
            L[k] ← B[i]
              k++
    s1 ← getSign ( p2, p3, p1)
    for i ←0 to n-1 do
```

```
s1 ← getSign(p2, p3, p1)
for i ← 0 to p-1 do
    s2 ← getSign(p2, p3, B[i])
    if (!sameSign(s1, s2))
        M[q] ← B[i]
        q++
QuickHull(L[0...k-1], p1, p3)
QuickHull(M[0...q-1], p2, p3)
```

Analysis Using Master Theorem,

We Consider only quickHull function.

Comparision is the basic operation.
We get the following recurrence equation, for best case:

$T(n) = 2T(n/2) + n$

$a = 2 \quad b = 2, d = 1$, we get

$2 = 2^1$

From master theorem, we know that

If $a = b^d$, then $T(n) \in O(n^d \log n)$

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