Divide-and-Conquer

May 30, 2019

What is Divide-and-Conque?

- Divide: Splite the problem into several subproblem. (e.g. divide the entire array into half)
- Conquer: Solve the subproblem (recrusively) applying the same algorithm (e.g. compare each element for the subarray)
- Combine: Use subproblem results to derive a final result for the original problem (e.g. merge)

Questions:

Given a permutation $a_1, a_2, \cdots, a_n$ of the number $1, 2, \cdots, n$ find the number of pairs (a_i, a_k) such that $i < k$ and $a_i > a_k$
The method we are using is similar to the merge sort!!
Firstly, we split array into two almost equal pieces.
Then we recrusively solve each subproblem by keep split the subarray into two equally arrays until we hit the base case.
We need to change the merge part a little bit, the algorithm is shown below:

Algorithm 1 Merge and Count

```
1: $i \to 1; j \to 1; r \to 0; C \to \emptyset$
2: while $i \leq m$ and $j \leq n$ do
3:     if $A[i] < B[j]$ then
4:         append $A[i]$ to $C$
5:         i++
6:     else
7:         append $B[j]$ to $C$
8:         j++
9:         r = r + m - i + 1
10:    end if
11: end while
12: return (C ,r)
```

Explanation for line #9:
Since we are merging two array together and want to find out missed match pair. Since we assume (by the recrusively call) two subarrays is sorted and we already know that there is r_l pairs in the left subarray and r_r pairs in the right part.
However, we need to consider those pairs that is across two arrays (i.e. the value of element in left subarray is greater than the value of element in the right subarray.)
Therefore, we do the merge count on those two subarrays.
If the element in $A[i]$(left array) is greater than $B[j]$ that means every element in left array after index i is greater than than $B[j]$ which is the number of mismatched pairs.
Since we have m elements in the left array and the current index is i, then we have $m-i+1$.
And we recrusively call this for each subproblem.

Matrix Multiplication

Consider we have two n by n matrices, X and Y. We want to compute the n by n matrix product $Z = XY$

The first solution:

1. Compute the product of every row in the first matrix and the every column in the second matrix.
2. Every time you have $n \times n$ multiplication and you have n rows to compute.
3. We have $\Theta(n^3)$ complexity.

The second solution:

Consider we are using the divide-and-conque method. We first need to assume that n is power of 2.

Then we can recrusively divide the n by n matrix into four pieces and eventually reach 2 by 2 matrix.

Consider we have

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix} \quad C = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

If A and B are n by n matrices, then $a, \cdots, h, r, s, t, u$ are $\frac{n}{2}$ by $\frac{n}{2}$ matrices, where

$$r = ae + bg \quad s = af + bh$$

$$t = ce + dg \quad u = cf + dh$$

Eventually we requires 8 multiplications of $\frac{n}{2}$ by $\frac{n}{2}$ matrices in order to compute $C = AB$
Therefore we have $T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$
By applying Master Theorem, we know that the complexity of this algorithm is $\Theta(n^3)$.

The improvement of Matrix Multiplication:

Instead of computing eight times multiplications, we can simply reduce it to 7 times.
Define:

$$P_1 = a(f - h) \quad P_2 = (a + b)h$$
$$P_3 = (c + d)e \quad P_4 = d(g - e)$$
$$P_5 = (a + d)(e + h) \quad P_6(b - d)(g - h)$$
$$P_7 = (a - c)(e + f)$$

Then compute:

$$r = P_5 + P_4 - P_2 + P_6 \quad s = P_1 + P_2$$
$$t = P_3 + P_4 \quad u = P_5 + P_1 - P_3 - P_7$$

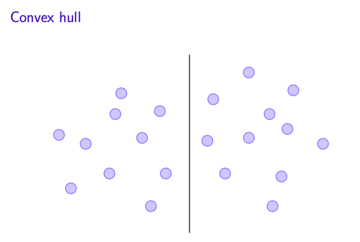
We now require only 7 multiplications of $\frac{n}{2}$ by $\frac{n}{2}$ matrices in order to compute $C = AB$
We have $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$
By using Master Theorem we have $\Theta(n^{log_2^7})$ complexity that is less than $\Theta(n^3)$

Convex hull

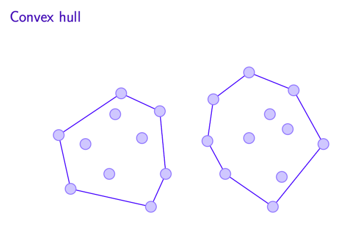
Convex hull problem:
For a given set S of n points, consturct the convex hull of S.
Solution:
Find the points that will serve as the vertices of the polygon in question and list them in some regular order.
Assume that hulls are defined in ccw (counter-clock wise) order.

Algorithm 2 ConvexHullMerge(L,R)

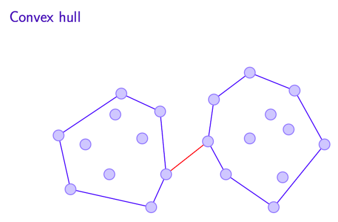
```
1:  $A \leftarrow$ rightmost point of $L$
2:  $B \leftarrow$ leftmost point of $R$
3:  while $T = AB$ is not the lower tangent to both $L$ and $R$ do
4:      while $T$ is not lower tangent to $L$ do
5:          $A \leftarrow A - 1$
6:      end while
7:      while $T$ is not lower tangent to $R$ do
8:          $B \leftarrow B + 1$
9:      end while
10:     while $T$ is not higher tangent to $L$ do
11:         $A \leftarrow A + 1$
12:     end while
13:     while $T$ is not higher tangent to $R$ do
14:         $B \leftarrow B - 1$
15:     end while
16: end while
```



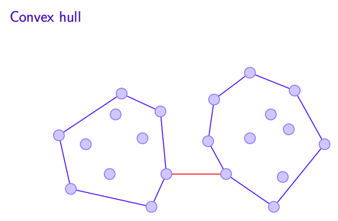
$\rightarrow$



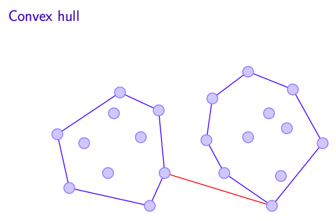
$\rightarrow$



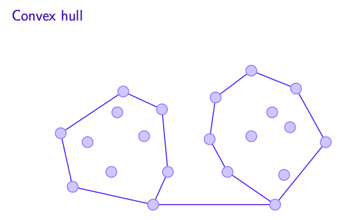
$\rightarrow$

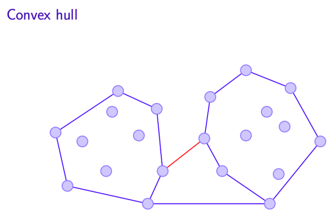


$\rightarrow$

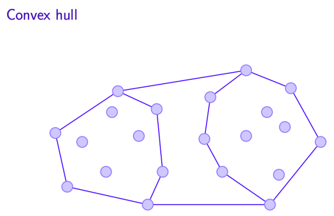


$\rightarrow$

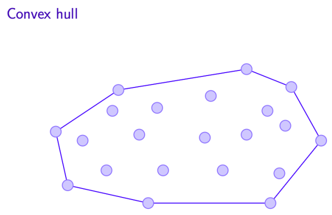




$\rightarrow$



$\rightarrow$



$\rightarrow$

We recursively divide the entire points into half and merge them by using the above alogrithm. Therefore the complexity is $T(n) = 2T(n) + \Theta(n) \Rightarrow \Theta(nlog(n))$.