Divide and Conquer for solving Insert-Query Problems Offline

আজকে একটা Divide and Conquer Trick নিয়ে কথা বলব। কোন Data Structure এ Insert-Query ধরনের প্রবলেম এর জন্য এই Trick কাজে লাগবে। এই Trick ব্যবহার করা যাবে যখন আমরা Data Structure টা সহজেই Statically Build করতে পারি (সব Insert সব Query এর আগে থাকে), কিন্তু Alternatively Insert আর Query করতে পারি না।

শুরুতে একটা খুবই সাধারণ প্রবলেম দিয়ে শুরু করা যাক –

You have an empty set. You need to perform following operations –

1. *Insert a given number X in the set.*
2. *Count how many numbers in the set are less than or equal to a given number X.*

যেমন এইরকম হতে পারে –

- $Insert(4) \Rightarrow [4]$
- $Insert(2) \Rightarrow [4,2]$
- $Query(3) \Rightarrow 1$
- $Query(4) \Rightarrow 2$
- $Insert(3) \Rightarrow [4,2,3]$
- $Insert(1) \Rightarrow [4,2,3,1]$
- $Query(4) \Rightarrow 4$

এইটা অনেক উপায়েই Solve করা যায়, যেমন: Ordered-Set অথবা Compress + Binary Indexed Tree ইত্যাদি। কিন্তু এইখানে আমরা Divide and Conquer Approach দেখব।

আগেই বলেছি যে, সব Insert যদি সব Query এর আগে দেওয়া থাকে, তাহলে আমাদের Solve করতে পারতে হবে। এখন সেটা কিভাবে করা যায়?

খুব সোজা! আমরা সব Insert করা Number গুলা Sort করে ফেলব, এর পরে একটা Query solve করার জন্য একটা Binary search করাই যথেষ্ট!

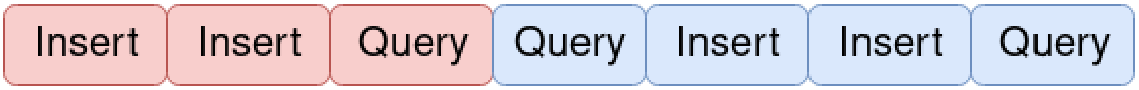
এখন আমাদের প্রবলেম এ তো আর সব Insert গুলা Query এর আগে থাকবে না। তাহলে কি করব? আমরা Divide and Conquer use করে সব Insert কে Query এর আগে নিয়ে যাব।

লক্ষ্য করি, একটা Query তার আগের সব Insert গুলার উপরে নির্ভর করে। কিন্তু তার মানে কিন্তু এইটা না যে আমাদের সব Insert করা জিনিস গুলা একসাথে থাকা লাগবে। আমরা চাইলে আলাদা আলাদা ভাবে একেকটা Insert কে Consider করতে পারি।

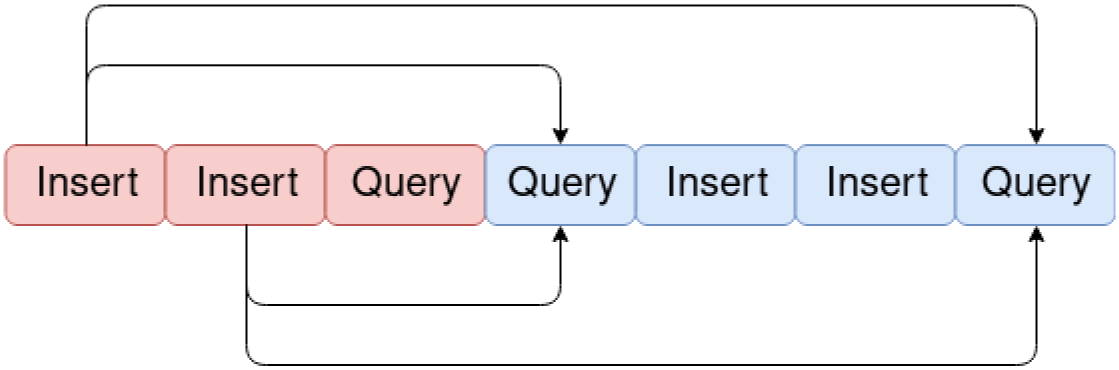
যেমন, উপরের উদাহরণে এ Last Query এর জন্য আমাদের সমস্ত Insert করা Number গুলা Consider করতে হবে, মানে [4,2,3,1]। কিন্তু আমরা [4,2] আর[3,1] এর জন্য Answer merge করলেও একই Result পেতে পারি!

সোজা কথা, আমাদের একটা Query এর জন্য তার আগে সব Insert কে Consider করতে হবে, সেটা একসাথে হোক, অথবা অনেক ভাগে ভাগ করে সব গুলার Answer merge করে হোক। এই কাজটাই আমরা Divide and Conquer দিয়ে করব।

সেটা করার জন্য আমরা সব Operation গুলোকে মোটামুটি সমান ২ ভাগে ভাগ করব –



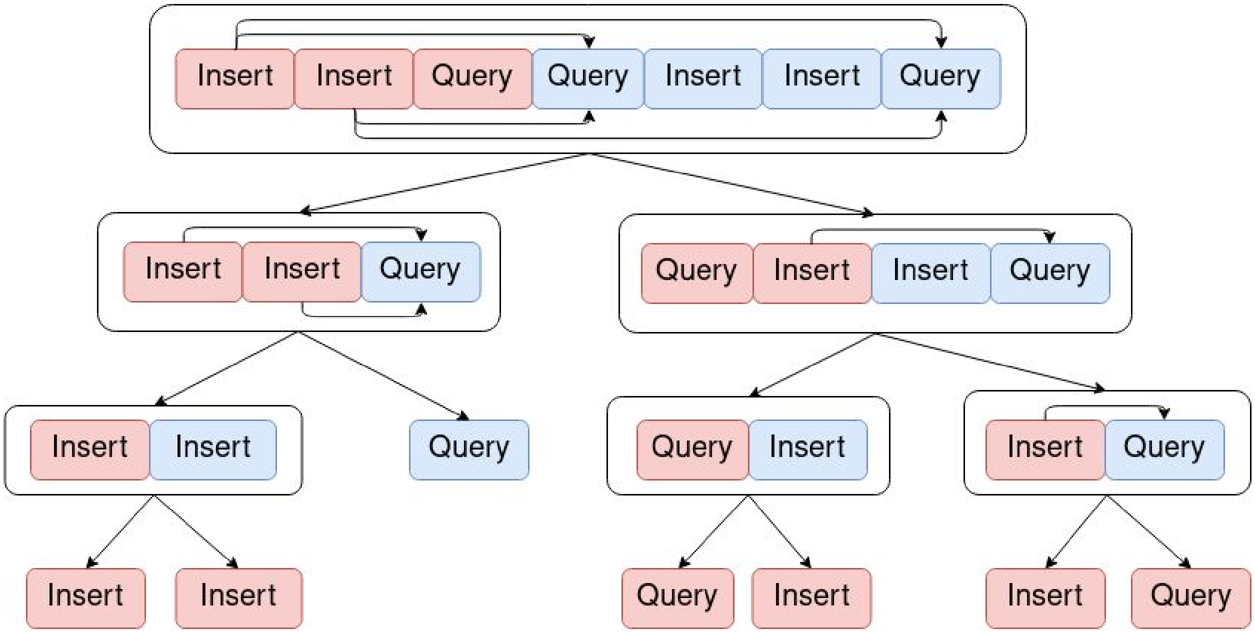
এখন লক্ষ্য করি, Left side এর সমস্ত Insert, Right side এর সমস্ত Query কে Affect করে। তাহলে আমরা সব Left side এর Insert গুলা দিয়ে আগের মত করে একটা Data Structure বানিয়ে ফেলতে পারি। আর সেইটা ব্যবহার করে Right side এর সব Query গুলার Answer update করে ফেলতে পারি –



কিন্তু, আমাদের Right side এর Query গুলা তো আর শুধু মাত্র Left side এর Insert গুলার উপরে নির্ভর করে না। বরং আর কিছু Insert তো Right side এর মধ্যেও আছে। সেইগুলার কি হবে? আবার Left side এর কোন Query-ই তো Solve করা হল না! সেইগুলারই বা কি হবে?

এইখানেই আসল মজা 😉। আমরা এখন Left আর Right side এর জন্য আবার একই কাজ করব 😉। কারণ এখন আমাদের Right side এর Query গুলার জন্য আর Left side এর Insert গুলা Consider করতে হবে না। আমাদের যা Consider করা বাকি আছে সেইগুলা হল Left side এর Query এর জন্য Left Side এর Insert, আর Right side এর Query এর জন্য Right side এর Insert। তাই আমরা একই কাজ এই ২ ভাগের জন্য Recursively করতে পারি। এভাবে করলে সবশেষে যেকোনো Query এর জন্য তার আগের সব Insert কেই Consider করা হয়ে যাবে।

যেমন উপরের কাজ টা Continue করলে এমন হবে –



এখানে সব Query এর জন্যই তার আগের সব Insert গুলা Consider করা হয়েছে।

এখন এইটার Time Complexity Analysis করা যাক। আমাদের এই Solve Process টা সর্বচ্চ O(logn) Depth পর্যন্ত যাবে। Complexity হবে সব Depth এর Total complexity যোগ করে। এইক্ষেত্রে আমাদের একটা Depth এ অনেক গুলা Sort হচ্ছে। কিন্তু সব গুলা Sort যোগ করলে O(nlogn) হবে [merge sort এর মতো করে sort করতে হবে]। এইরকম logn টা Level রয়েছে। তাই মোট Complexity $O(nlog^2n)$ ।

Source: https://rezwanarefin01.github.io/