1. Soru

Problemin Tanımı:

Elimizde müzede çalışan bir grup çalışan vardır. Biz de bir grup işçi tarafından çalındığını bilinmektedir. Bildiğimiz başka bir şey de çalışan grubun yarısından fazlası her zaman doğruyu söylüyor ve mantıken doğruyu söyleyenler de hırsız olamıyor. Bir tek doğru söyleyen kişi bulursak kimler hırsız bulabiliriz. Bize bir kişi bile yetiyor.

Problemin Çözümü:

Elimde çalışan grubunda yarısından fazla kesin doğru söyleyen olması bu bizim için bir avantaj. 'Merge Sort' yaklaşımı ile ele alıcaz. Öncelikle çalışan kişileri tek bir kişi haline gelene kadar parçalıyoruz. Sonra birleştirirken birbirine doğru diyen adamları yeni bir dizide birleştiriyoruz. Çünkü ya ikisi hırsızdır ya da kesin doğru söyleyen. Birleştirirken en küçük bir iddiada bile farklı düşünen varsa o çiftleri dahil etmiyoruz. Çünkü kesin doğru söyleyen adamlar daha fazla olduğu için arada sırada yalan söyleyenlerin bir önemi kalmayacak. Birbirine doğru diyen iki adam ya kesin doğru söyler ya da ikisi de hırsız. Ondan dolayı başka insanlarla da sorgu yapacağı zaman onların durumu ortaya çıkacak. En sonda kalanlar kesin doğruyu söyleyenler olacaktır.

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

Şekil-1

Şekil-1'deki rekürans bağıntısını ele alınırsa 'T(n/2)' ifadesinin içerisinde ikiye bölmemizin sebebi her defasında iki parça diziye ayırdığımız için bölündü. 'T(n/2)' ifadesinin kat sayısı iki olmasının sebebi bölünen iki dizi de kullanıldığı başına yazıldı. Sabit sayının 'n' olmasının sebebi her adımda n kere karşılaştırma yapıldığından dolayı ekleme yapılması gerekiyor.
Master teoremini kullanarak da karmaşıklığını ele alalım:
a = 2, b = 2, d = 1
a = b^d şartı sağlandığı için karmaşıklık O(n^d logn) olur. Sonuç olarak karmaşıklık O(nlogn) olur.

100 kişiyi 1 kişiye inecek kadar ayırmak. Ondan sonrada birbiri hakkında doğru söyleyen kişileri birleştirmek.
İlk başta birbirlerinin iddialarına doğru diyen ikili gruplar olacak. Ondan sonra da 2 tane ikili grubu alarak birbirleri Hakkında soru sormasını sağlıyoruz. En baştan başlayarak soru sordurmaya başlıyoruz. İlk iki kişi birbiri hakkında doğru söylüyorsa hemen yanındaki arkadaşlarına sorguya çekiyoruz farklı görüşler olana kadar bu devam ediyor ve eğer farklı görüş varsa bu iki grubu eliyoruz. Farklı gruplara geçiyoruz. Herkes aynı birbiri için doğru derse elimiz-De yeni birbirleri hakkında doğru söyleyen çalışanlar olacak. Bu böyle sonuna kadar gidecek.

2. Soru

Problemin Tanımı:

Elimizde kilit ve anahtarlar vardır. Her kilit sadece bir kilide özgüdür. Her kilidin ve anahtarın boyutları farklıdır. (N*logN) zaman karmaşıklığında çözülmesi isteniyor. Anahtarlar ve kilitler kendi aralarında kıyaslama yapamıyoruz .Ondan dolayı her seferinde rastgele anahtar alınarak kilitlere takılıp deneniyor.

Problemin Çözümü:

Öncelikle aklımıza ‘Quick Sort’ sıralama algoritması geliyor. Çünkü ‘Merge sort’ yapamayız anahtarları ve kilitleri kendi aralarında kıyas yapamıyoruz. O zaman rastgele aldığımız anahtarın kilitlere denenmesi gerek ve denerken de aynı zamanda eğer anahtar kilide küçük geldiyse anahtarın sağ tarafına koymamız gerek. Eğer anahtar kilide sığmıyorsa anahtarın sol tarafına koymalıyız. Kontrol ederken aynı zamanda diziyi de düzenlemiş oluyoruz. Eğer anahtar kilide uyduysa ve benim elimde hala düzenlenmemiş kilitler varsa onları da düzenlemeye devam etmek gerek.
Tüm işlemlerim tek bir anahtar-kilit uyumuyla bittikten sonra elimde anahtarın sağ tarafında kendisinden büyük sol tarafında kendisinden küçük kilitler olacak. Elimde iki parça var ve onlara da aynı bu şekilde adımlar uygulanmaktadır.
En sonunda da elimizde sıralı anahtar-kilit uyumu olan ikililer var

Karşılaşılan Sorunlar:

En başta ‘Merge Sort’ ile yapılmayacağını anlamak çok kolaydı. ‘Quick Sort’ aklıma geldi. Derste işlediğimiz Hoare’s yaklaşımı ile çözmeye çalıştım. Burada dizinin sağından(i) ve solundan(j) geliyoruz kısacası böyle. En çok kafamı karıştıran eğer anahtarım doğru kilitle erken zamanda karşılaşırsa ne yapmam gerektiğini tam anlayamadım. Ben de direkt pivot değerini görünce hiçbir şey yapmayayım dedim. Pivot görünce yani i ya da j değerleri değişmiyor. O zaman da kodun verimliliği düşüyordu. Ben de eğer erkenden pivota erişirsem pivot değerini dizinin en başına alayım. İşlemler tamamlandıktan sonra en baştaki pivotumu doğru yere yerleştiririm. Düşünmek için çok kafa patlattım.

Problemin Karmaşıklığı:

```
keytoLock:
Input:
key, array that contains the values of keys
lock, array that contains the values of locks
left, first index of array
right, last index of array

if(left >= right) return
randomIndex <-- left + ( rand() % (right-left) ) + 1
pivotIndex <-- partition(lock, left, right,  key[randomIndex])
partition(key, left, right, lock[pivotIndex]);
keyToLock(key, lock, left, pivotIndex-1);
keyToLock(key, lock, pivotIndex+1, right);
```

```
partition:
Input:
arr, array that contains the values
left, first index of array
right, last index of array
pivotValue, value of key
return  j, index of the pivot in the array

i <-- left
j <-- right
while (i<j)
        while(arr[i] <= pivotValue)
                if(arr[i] == pivotValue)
                        swap(&arr[i], &arr[left]);
                ENDIF
                i <-- i + 1
          ENDWHILE
          while(arr[j] > pivotValue)
                   j <-- j-1
           ENDWHILE
           swap(&arr[j], &arr[i])
ENDWHILE
swap(&arr[j], &arr[i])
swap(&arr[j], &arr[left])
return j
```

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

Şekil-2

Şekil-2'de verilen denklem rekürans bağıntısıdır. Bu denklemin ortaya çıkmasında 'keyToLock' ve 'partition' fonksiyonları kullanılarak elde edilmiştir. 'T(n/2)' ifadesinin içinde ikiye bölmemizin sebebi 'keyToLock' fonksiyonun içinde küçük ifadelerin sağda büyüklerin solda sıralandıktan sonra yani ikiye ayrılmasına sağladığı için bölündü. 'T(n/2)' ifadesinin kat sayısının 2 olmasının sebebi ise ikiye ayrılan diziyi ayrı ayrı aynı anda işleme tabi tutulduğu için 2 oldu. Denklemdeki sabit sayımız ise 'partition' fonksiyonunda geliyor. Her adımda n tane karşılaştırma yapıldığı için sabit olarak 'n' yazıyoruz.
Master teoremini kullanarak da karmaşıklığını ele alalım:
a = 2, b = 2, d = 1
a = b^d şartı sağlandığı için karmaşıklık O(n^d logn) olur. Sonuç olarak karmaşıklık O(nlogn) olur.

Senaryo 1

```
Keys:
A B E L D C F
Locks:
B L D F E C A
Matched keys and locks are :
Keys:
A B C D E F L
Locks:
A B C D E F L

--------------------------------
Process exited after 0.04027 seconds with return value 0
Press any key to continue . . .
```

Senaryo 2

```
Keys:
A B C D E F
Locks:
A B C D E F
Matched keys and locks are :
Keys:
A B C D E F
Locks:
A B C D E F

--------------------------------
Process exited after 0.03544 seconds with return value 0
Press any key to continue . . .
```

Senaryo 3

```
Keys:
A B C D E F
Locks:
F E D C B A
Matched keys and locks are :
Keys:
A B C D E F
Locks:
A B C D E F

--------------------------------
Process exited after 0.03381 seconds with return value 0
Press any key to continue . . .
```