CSE 321 HOMEWORK 03

Seyfullah Bezcikli
121044028

1) a) Euclid' algoritması $gcd(m,n) = gcd(n, m\%n)$ formülünü kullanır. Yani yeni oluşan ikilinin size'ı $m\%n$ kadar olur. Bundandır ki size 0 ile $n-1$ arasında herhangi bir sayı olabilir. Böylece size 1 ve n arasında herhangi bir sayı ile azalabilir.

b) iki ardışık iterasyon şu formüle göre hesaplanır;

$$gcd(m,n) = gcd(n,r) = gcd(r, n\%r) \;;\; r = m\%n$$

Burada $n\%r$ nin $n/2$ den küçük eşit olduğunu göstermemiz gerekmektedir.

Eğer $r \leq n/2$ ise $n\%r < r \leq n/2$

Eğer $n/2 < r < n$ ise $n\%r = n - r \leq n/2$

2) - öncelikle en küçük problemi çözmek gerek.
- Sonrasında problemin tamamı çözülür.
  - ↳ n i koyabileceğimiz bütün ihtimaller arasına yerleştiririz ve $n-1$ elemanın permütasyonu oluşur.
  - ↳ n yi sağdan sola ve soldan sağa olacak şekilde elimizde olan bir listeye ekleyebiliriz.

örneğin:

A $\quad n=1,\ n! = 1$

AB BA (sağdan sola) $\quad n=2,\ n! = 2$

ABC ACB CAB (sağdan sola) CBA BCA BAC (soldan sağa) $\quad n=3,\ n! = 6$

3) a) Birinci durum için eğer bir key çocuğu olmayan bir node dan silinecek ise key in parent ında key'in olduğu node'u null gösterecek bir pointer oluştur. Eğer key bir parent'a sahip değilse bu key'in root olduğunu gösterir.
İkinci durum için, eğer key bir çocuklu ise ve silinecek ise keyin parent'ında keyin child'ını gösteren bir pointer yapılır. Eğer silinecek elemanın parent'ı yoksa ve tek çocuğu varsa tek çocuğu yeni root yapılır. Üçüncü durum için silinecek key iki çocuğa sahip ise silme işlemi 3 adımda yapılır.

=)

Önce keyin sağ alt ağacındaki en küçük eleman bulunur. Sonra key ile bulunan en küçük eleman yer değiştirir. Son olarak key birinci ve ikinci duruma göre düşünülerek silme işlemi gerçekleştirilir.

Bu durumlara göre iterilen algoritma variable-size-decrease algoritması olamaz. Çünkü problemi daha küçük bir binary tree'ye küçülterek silme işlemi için algoritmayı çalıştırıyoruz.

b) En kötü durum için bir input düşündüğümüzde binary tree den kök silme olayı, $-2, 1, n, n-1, \dots, 3$ keylerinin ardışık olarak eklenmesi ile elde edilir. Sağ sub tree deki en küçük keyi bulmak için $n-2$ pointer ın takip ettiği bir zincire ihtiyaç vardır, Silme algoritmasının en kötü durumda verimi $\Theta(n)$ dir. Bir binary tree nin yüksekliği n random key den oluşturulduğunda logaritmik bir fonksiyon olur ve ortalama durumda silme algoritmasının verimının logaritmik olmasını bekleriz.

4) Burada counting sort kullanabiliriz. İlk olarak input arrayindeki bütün elemanlara counting sort algoritmasının gereği olarak 1 ekleriz. Counting sort çalıştırıldıktan sonra, output arrayinde olan elemanlardan 1 çıkarırız.

$A[1, \dots, n]$ input array imiz olsun

$A[1 \dots i]$ sadece -1 içerir.

$A[i+1 \dots j]$ " 0 "

$A[h \dots n]$ " $+1$ "

Başlangıçta $i=0$ $j=0$ ve $h=n+1$ olsun. Eğer $h=j+1$ ise işlem tamamdır yani array sıralıdır. Döngüde incelediğimizde $A[j+1]$, Eğer $A[j+1]=-1$ ise $A[j+1]$ ve $A[i+1]$ yer değiştirir ve i yi ve j yi arttırırız. Eğer $A[j+1]=0$ ise j yi arttırırız. Son olarak eğer $A[j+1]=1$ ise $A[j+1]$ ve $A[h-1]$ yer değiştirir ve h 1 azaltılır.

5) FindIndexEqElement(A[1....n], offset)
If A[n/2] equals offset + (n/2) return True
If |A| < 1 return false
If A[n/2] < offset + (n/2) return FindIndexEqElement(A[n/2,... n], offset + n/2)

else return findIndexEqElement (A[1... (n/2 - 1), offset)