BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ	Ad Soyad	
BIL203 Veri Yapıları	Numara	
Final sınav – Ocak 2011	İmza	

1	2	3	4	5	6	7	8	9	Toplam

Açıklamalar:

1. Toplam soru adedi **5** dir.

2. Cevaplama süresi **100** dakikadır.

3. Gerçeklemelerinizi verilen boşluklara yazınız.

1. (20 p) özyineleme, şifreleme, takip

RSA temelli şifreleme yapılmak istenmektedir. `A` kişisinin anahtar çiftleri: `A.özel, A.genel`; `B` kişisinin anahtar çiftleri: `B.özel, B.genel` olmak üzere. Mesajı şifrelemek için **`şifrele(m, anahtar)`**, çözmek içinse **`çöz(c, anahtar)`** hazır işlevleri sağlandığına göre,

a) Sacece **`A`** kişisinin alabileceği mesaj nasıl şifrelenmelidir? Bu mesaj nasıl çözülür?

Şifreleme:	**c** =
Çözme:	**mc** =

b) **`B`** kişisinin **`m`**-mesajını aşağıdaki gibi şifrelemesi durumunda şifreli mesajı kimler ve nasıl çözebilir?

Şifreleme:	**c = şifrele(m, B.özel)**
Kim (**A - B – Hiçbiri**):	
Çözme:	**mc** =

c) **`m`**-mesajını gönderenin **`B`** kişisi, alanın ise **`A`** kişisinin olmasını sağlamak için şifreleme ve çözme nasıl olmalıdır?

Şifreleme:	**c** =
Çözme:	**mc** =

d) **`ext_gcd(31, 11)`** durumu için özyineli çağrı ardışıllığını (ağacını) **`ext_gcd(a,b)`** formunda ve dönüş değerlerini de **`(d,a,b)`** formunda yazınız.

```
def ext_gcd(x,y):
    if y == 0:
        return(x,1,0)
    else:
        (d,a,b) = ext_gcd(y, x%y)
        return(d,b,a-(x/y)*b)
```

(..., ..., ...) = ext_gcd(31, 11)
(..., ..., ...) = ext_gcd(..., ...)
(..., ..., ...) = ext_gcd(..., ...)
(..., ..., ...) = ext_gcd(..., ...)
(..., ..., ...) = ext_gcd(**1, 0**)

2. (20 p) Algoritma, iyileştirme, takip

QUICK-FIND algoritmasının `BIRLESTIR` işlevi aşağıda verildiği gibidir.

```
# BIRLESTIR: "id[p]" li tüm girdileri "id[q]" ile değiştir.
def BIRLESTIR(p, q):
        pid = id[p]
        for i in range(id):
            if id[i] == pid:
                id[i] = id[q]
```

a) Bu algoritmayı aşağıdaki başlangıç durumu için takip ederek, tabloyu doldurun.

Birleştir	**i**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
(p, q)	**id[i]**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
"3 - 4"	**id[i]**										
"4 - 9"	**id[i]**										
"8 - 0"	**id[i]**										
"2 - 3"	**id[i]**										
"3 - 9"	**id[i]**										

b) **"4–9"** aşamasında **"9-4"** olacak olsa **2** değer yerine **1** değer değişecekti ve daha verimli çalışacaktı; benzer şekilde **"3-9"** aşamasında gereksiz bul-değiştir yapılmaktadır. Bu durumu işleyerek daha verimli çalışacak biçimde **BIRLESTIR** kod parçasını iyileştirin.

```
def BIRLESTIR(p, q):
```

3. (15 p) SVT, miras

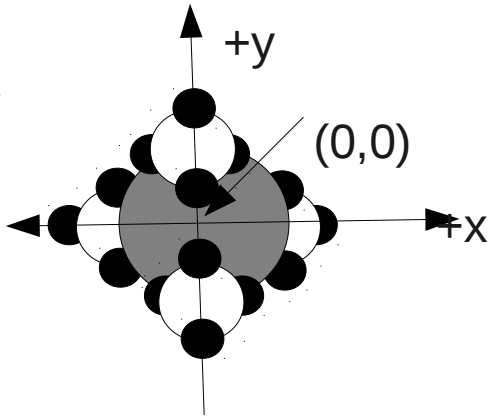
Ekte verilen Deqeue Soyut Veri Türünden (SVT) miraslanan yığıt SVT'nü gerçekleyin.

```
class Stack(Deque):
      def __init__(self):

      def isEmpty(self):

      def push(self,item):

      def pop(self):

      def peek(self):

      def size(self):
```

4. (20 p) fraktal, özyineleme

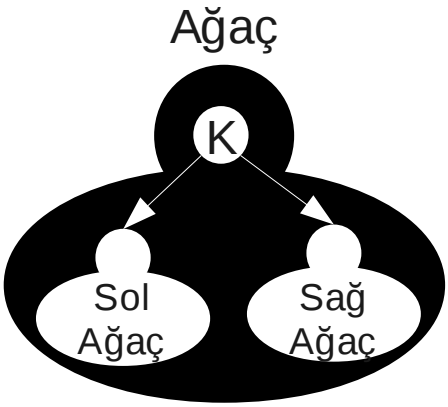
Yanda verildiği biçimde olan fraktal çizen özyineli **çiçek**(x,y,r,h) işlevi tasarlanmak istenmektedir. Bu amaçla çember(x,y,r) yardımcı işlevini kullanabilirsiniz. Burada (x,y) merkez, r: yarıçap ve h: derinliktir.

Yandaki gösterimde h=3, (x,y) = (0,0) seçilmiştir. En büyük yarıçaplı (beyaz) çember "r" yarı çapındayken, orta boy olan "r/2" ve en küçük olan da "r/4" yarı çapındadır.




```
def çiçek(x, y, r, h):
```

5. (25 p) özyineleme, ağaçta dolaşım



"Ağaç, kök ve/veya sol ve/veya sağ ağaçtan oluşur" tanımından hareketle, bu ağaçta özyineli ziyaret için sunulan üç farklı yaklaşım aşağıdaki tabloda verilmiştir.

Post-order	**In-order**	**Pre-order**
Sol ağacı ziyaret et **Sağ** ağacı ziyaret et **Kökü** ziyaret et	**Sol** ağacı ziyaret et **Kökü** ziyaret et **Sağ** ağacı ziyaret et	**Kökü** ziyaret et **Sol** ağacı ziyaret et **Sağ** ağacı ziyaret et

a) Aşağıda verilen ağaç durumları için her bir ziyaretle üretilecek diziyi yazınız.

post-			
in-			
pre-			

b) Aşağıdaki PostScript kodundaki işleçler köke gelecek şekilde post-order ile ziyaret edildiği bilinmektedir. Bu diziyi üretecek ağacı oluşturunuz.

```
6 2 – 4 3 + * 18 – 10 / 10 –
```

Yardımcı işlevler

```
class Stack():
      def __init__(self):
            self.items=[]

      def isEmpty(self):
            return self.items==[]

      def push(self,item):
            self.items.append(item)

      def pop(self):
            return self.items.pop()

      def peek(self):
            return self.items[len(self.items)-1]

      def size(self):
            return len(self.items)
```

```
class Queue:
    def __init__(self):
             self.items = []

    def isEmpty(self):
             return self.items == []

    def enqueue(self, item):
             self.items.insert(0,item)

    def dequeue(self):
             return self.items.pop()

    def size(self):
             return len(self.items)
```

```
class Deque:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def addFront(self, item):
        self.items.append(item)

    def addRear(self, item):
        self.items.insert(0,item)

    def removeFront(self):
        return self.items.pop()

    def removeRear(self):
        return self.items.pop(0)

    def size(self):
        return len(self.items)
```