R-2.19 Draw the 11-item hash table resulting from hashing the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, using the hash function *h(i)* = (2*i* + 5) mod 11 and assuming collisions are handled by chaining.

	20			16-11	44-88-11	94-39	12-23		13	
0	1	2	3	4	5	6	7	8	9	10

H(12)=(2*12+5)%11 =7
H(44)=(2*44+5)%11=5
H(13)=(2*13+5)%11=9
H(88)=(2*88+5)%11=5
H(23)=(2*23+5)%11=7
H(94)=(2*94+5)%11=6
H(11)=(2*11+5)%11=5
H(39)=(2*39+5)%11=6
H(20)=(2*20+5)%11=1
H(16)=(2*16+5)%11=4
H(5)=(2*5+5)%11=4

R-2.20 What is the result of the previous exercise, assuming collisions are handled by linear probing?

11	39	20	4	16	44	88	12	23	13	94
0	1	2	3	4	5	6	7	8	9	10

H(12)=(2*12+5)%11 =7
H(44)=(2*44+5)%11=5
H(13)=(2*13+5)%11=9
H(88)=(2*88+5)%11=5
H(23)=(2*23+5)%11=7
H(94)=(2*94+5)%11=6
H(11)=(2*11+5)%11=5
H(39)=(2*39+5)%11=6
H(20)=(2*20+5)%11=1

H(16)=(2*16+5)%11=4
H(5)=(2*5+5)%11=4

R-2.21 Show the result of Exercise R-2.19, assuming collisions are handled by quadratic probing, up to the point where the method fails because no empty slot is found.

39	20	16		11	44	88	12	23	13	94
0	1	2	3	4	5	6	7	8	9	10

H(12)=(2*12+5)%11 =7
H(44)=(2*44+5)%11=5
H(13)=(2*13+5)%11=9
H(88)=(2*88+5)%11=5 i=5 J=1 A[(i+j^2)%11]=> A[(5+1)%11]=> A[6] =6
H(23)=(2*23+5)%11=7 i=7 j=1 A[(7+1)%11]=>A[8]=>8
H(94)=(2*94+5)%11=6 i=6 J=2 A[(6+2^2)%11]=>A[10]=>=10
H(11)=(2*11+5)%11=5 i=5 J=3 A[(5+3^2) %11]=>A[4]=>4
H(39)=(2*39+5)%11=6. I=6 J=4 A[(6+4^2)%11]=>A[0]=>0
H(20)=(2*20+5)%11=1
H(16)=(2*16+5)%11=4 i=4 J=3 A[(4+3^2)%11]=>A[2]=>2
H(5)=(2*5+5)%11=4. I=4. J=0 A[(4+0)%11]=4. Not empty
J=1 A[4+1)%11]=5. Not empty
J=2 A[(4+2^2)%11 =8 not empty
J=3. A[4+3^2)%11]=2 not empty
…..
J=11 A[4+11^2)%11]=4 not empty the quadratic probing is fails so we have to option either we need to change collision avoidance method or we need to create new array that increase by 50% of its original size then it will copy to the new array

R-2.22 What is the result of Exercise R-2.19 assuming collisions are handled by double hashing using a secondary hash function *h'(k)* = 7 – (*k* mod 7)?

11	23	20	16	39	44	94	12	88	13	5
0	1	2	3	4	5	6	7	8	9	10

H(12)=(2*12+5)%11 =7

H(44)=(2*44+5)%11=5

H(13)=(2*13+5)%11=9

H(88)=(2*88+5)%11=5 H'(88)= 7-88%7=3 then 3+5=8

H(23)=(2*23+5)%11=7 H'(23)=7-23%7=5 then 5+7 =12%11=1

H(94)=(2*94+5)%11=6

H(11)=(2*11+5)%11=5. H'(11)=7-11%7=3 then 3+5=8+3=11%11=0

H(39)=(2*39+5)%11=6 H'(39)=7-39%7=3 then 3+6=9=9+3=12%11=1+3=4

H(20)=(2*20+5)%11=1 H'(20)=7-20%7=7-6=1+1=2

H(16)=(2*16+5)%11=4 H'(16)=7-16%7=5=5+4=9+5=14%11=3

H(5)=(2*5+5)%11=4 H'(5)=7-5%7=2+4=6=>6+4=10

Give the pseudo-code description for performing a removal from a hash table that uses linear probing to resolve collisions. Why is it necessary to use a special marker to represent deleted elements?

```
Algorthim removeItem(k)
  Item:= findElement(k)
   insertItem("marked", value)
return item;
```

the markers needs to indictate not the last element when we needs to search the value if we get empty that means it is end of the values but if it marked that means it is not the last element so it helps us to keep search the values until he gets the empty values

C-4.10 Suppose we are given an n-element sequence S such that each element in S represents a different vote in an election, where each vote is given as an integer representing the ID of the chosen candidate. Without making any assumptions about who is running or even how many candidates there are, design an efficient algorithm to see who wins the election S represents, assuming the candidate with the most votes wins. Handle the possibility of **multiple winners** and **do this using a Dictionary**. Today specify your solution using pseudo code (tomorrow we will implement in JavaScript after discussing today's pseudo code solution).

```
Algorthim electionWinner(seq)
  D:=new HashTable()
  countElement(seq, D);
return findWinnerHelper(D)
Algorthim countElement(seq,D);
P:=seq.first()
Whiel !seq.isLast(p)) then do
  Key:=D.findValue(p.element())
    If(k===null)
     D.insertItem(p.element(),1)
  Else
     Key:=key+1;
     D.insertItem(p.element(),key)


Algorthim findWinner(D)
    Max:=findMax(D)
    J:=0;
     Winner:=new Array()
```

```
        For i:=0 to D.size() the do
            If(max===D[0].values()
                Winner[j++]=D[0]
  Return winner
```

Algorthim findMax(D)

```
     Max:=D[0].value();
   For i:=0 to D.size() the do
    If(max<D[0].values()
      Max=D[0].values
Return max
```