(b). Given a sequence of all books in a library (containing title, author, call number, and publisher) and another Sequence of 30 publishers, design an efficient algorithm to determine how many of the books were published by each publisher. (b) [5 points] What is the time complexity of your algorithm? Justify your answer.

Using sequnces

```
Algorthim bookPublisher(book,publisher)
    result:=new Sequnce();
   pub:=publisher.first()
       while(!publisher.isLast(pub))
              count:= countPublisher(book,pub)
               numOFBook:=new publisherNumber(pub,count)
              result.insertFirst(numOfBook)
            pub:=publisher.after(p)
 count:= countPublisher(book,pub)
 numOFBook:=new publisherNumber(pub,count)
result.insertFirst(numOfBook)

return result;

Algorthim countPublisher(book,pub)//
      cnt:=0
     P:=book.first()
        While(!book.isLast(p)) then do
              If(p.getPublisher()===pub)
                    cnt++;
         p:=book.after(p)
 If(p.getPublisher()===pub)
```

```
            cnt++;
return count;
```

Using has table

```
Algorthim countPublisher(book,publisher)
   result:=new Array()
    D:=new HashTable()
   insertToHasTable(book,D)
iterPub:=publisher.items()
 while(!iterPub.hasNext()) then do
       p:=iterPub.nextObject()
       cnt:=D.findValue(p)
            if(cnt===null)//if publisher did not have books
                    cnt=0;
     numOfBookPub:=new publisherNumber(p,cnt)
      result.push(numOfBookPub)
return result

Algorthim insertToHashTable(book,D)
   iterBook:=book.items()
  while(iterBook.hasNext()) then do
        b:=iterBook.nextObject()
       cnt:=D.findValue(b.getPublisher())
    if(cnt===null)
      D.inserItem(b.getPublisher(),1)
  else
```

cnt:=cnt+1

D.inserItem(b.getPublisher(),cnt)

This is my assumptions

```
Class librarayBook{
Constructor(title,author,callNumber,publisher){
   This._title=title;
  This._author=author
 This._callNumber=callNumber
 This.Publisher=publisher;

}
getPublisher(){
  return this._publisher
}
}
```

```
Class publisherNumber{

 Constructor(publisher,numberOfBook){
  This._publisher=publisher;
 This._numberOfBook=numberOfBooj;
}
}
```

1. Let book be a Sequence containing the words of a book in the same order they occur in the text of the book, i.e., one could read the book by traversing the Sequence book. Design a pseudo-code function fiveMostFrequent(book), that determines the five most frequently occurring words in book. The output is to be a list of

pairs; the pairs can each be a two-element array, e.g., a result of {[the, 10], [book, 7], [a, 5], [of, 3], [in, 3]}.

Extra credit bonus [+5] if you handle the case where there could be a tie for the fifth most, e.g., in the sample output, there could be one or more additional words that occur 3 times and would have to be included in the result.

Hint: Obviously, this can be done in many ways. The most important thing is that you come up with a design that works without bugs. However, what immediately jumps out at me is using a Dictionary followed by a Priority Queue (don't worry if this does not jump out at you, just design an algorithm that works).

```
Algorthim fiveMostFrequnt(book)
  BH:=new HastTable();
  insertToHasTable(book,BH)
 PQ:=new ProirtyQueue()
  insertToPQ(BH,PQ)
  seq:=new Sequnce()
   while(PQ.isEmpty()) then do
       bP:=PQ.removeMin();
          if(PQ.size()<5)
                seq.inserLast(bP)
return seq


Algorthim inserToHasTable(book,BH)
   iterB:=book.items()
  while(iterB.hasNext()) then do
    e:=iterB.nextObject();
```

```
cnt:=BH.findValue(e);
  if(cnt===null)
    BH.inserItem(e,1)
Else
  Cnt:=cnt+1
    BH.insertItem(e,cnt)
```

Algorthim insertToPQ(BH,PQ)

```
iterH:=BH.items()
while(iterH.hasNext())  then do
  e:=iterH.nextObject()
  PQ.insertItem(e.value(),e)
```

(a) [20 points] Given a Sequence B of thousands of credit card bills and another Sequence P of thousands of payments, design an efficient algorithm to create a Sequence of credit card bills that were not paid in full. The elements of Sequence B contain the credit card number, amount due, name, and address. The elements of Sequence P contain the credit number, amount paid, and name. The output of the algorithm should be a newly created Sequence containing the unpaid bills, i.e., **elements with the credit card number, name, address, amount due, and amount paid for those customers for which the amount paid is less than the amount due.** Note that you must handle the case where there is a bill, but there is no payment

B =>(creditCardNum,amountDue,name,address)

P=>(creditCardNum,amountPaid,name)

Out put+> (creditCardNum,name,address,amountDue,unPaid)

If amountPaid is less than amountDue

```
Algorthim unpadBills(B,P){
   iterB:=B.items();
  seq:=new Sequnce();
 while(iterB.hasNext()) then do{
       e:=iterB.nextObject()
     paidAmount:=paidBills(e.getCreditNumber(),P)
  if(paidAmount<e.getAmountDue()) then{
     bill:=new UnPiadBills(e.getCreditNum(),e.getName(),e.getAddress(),amountDue,paidAmount)
       seq.insertLast(bill)
}
}
return seq;
}
```

```
Algorthim paidBills(creditNum,P)
  iterP: =P.items()
        while(iterP.hasNext()) the do
            e:=iterP.nextObject()
            if(creditNum===e.getCreditNum())
                return e.getAmountPaid();
return 0;
```

```
//this is my assumptions
class UnPiadBills{
constructor(crdeitCardNum,name,address,amountDue,unPaid
){
   this._creditCardNum= crdeitCardNum;
   this._name=name;
 this._address=address;
```

```
    this._amountDue=amountDue;
   this._amountNotPaid=amountNotPaid
}
getCrdeitCardNum(){
return this._ crdeitCardNum
}
getAmountDue(){
return amountDue;
}
getAddress(){
return this._address
}}
```