a) Efficienty — time + space

User-friendly.

understandble.

Security.

I ~~think~~.

Efficieny. is time+space.

User-friendly: user-can use it easily

understandable: it is all about Komogorova

So ~~people~~ programmers can find the errors quicky

They are important because a software system is to provide service to users and as programmer we should check everything is understandable.

b)

CASE tools stand for Computer Aided Software Engineering. it is basically following the process like requirements -> Design -> implementation -> verification -> maintance following these instructions, the software develper can manage the process better.

c

c) Analsis effort method :: it is the method

flow input SFC.

pros very fast and convieniens

cons: not flexible

② parametric Estimating: it is a branch

of statics. pros: very accurate theoretically
cons: very complicated

③ I.K.T method

: evaluate complexity and cost..
pros: accurate
cons: Time consuming.

④ MBCE : use a Unique model Base.

Cost engineering Technology..

pros: accurate

cons: complicated

d)

First generations : machine langugae
very fast but not understandble

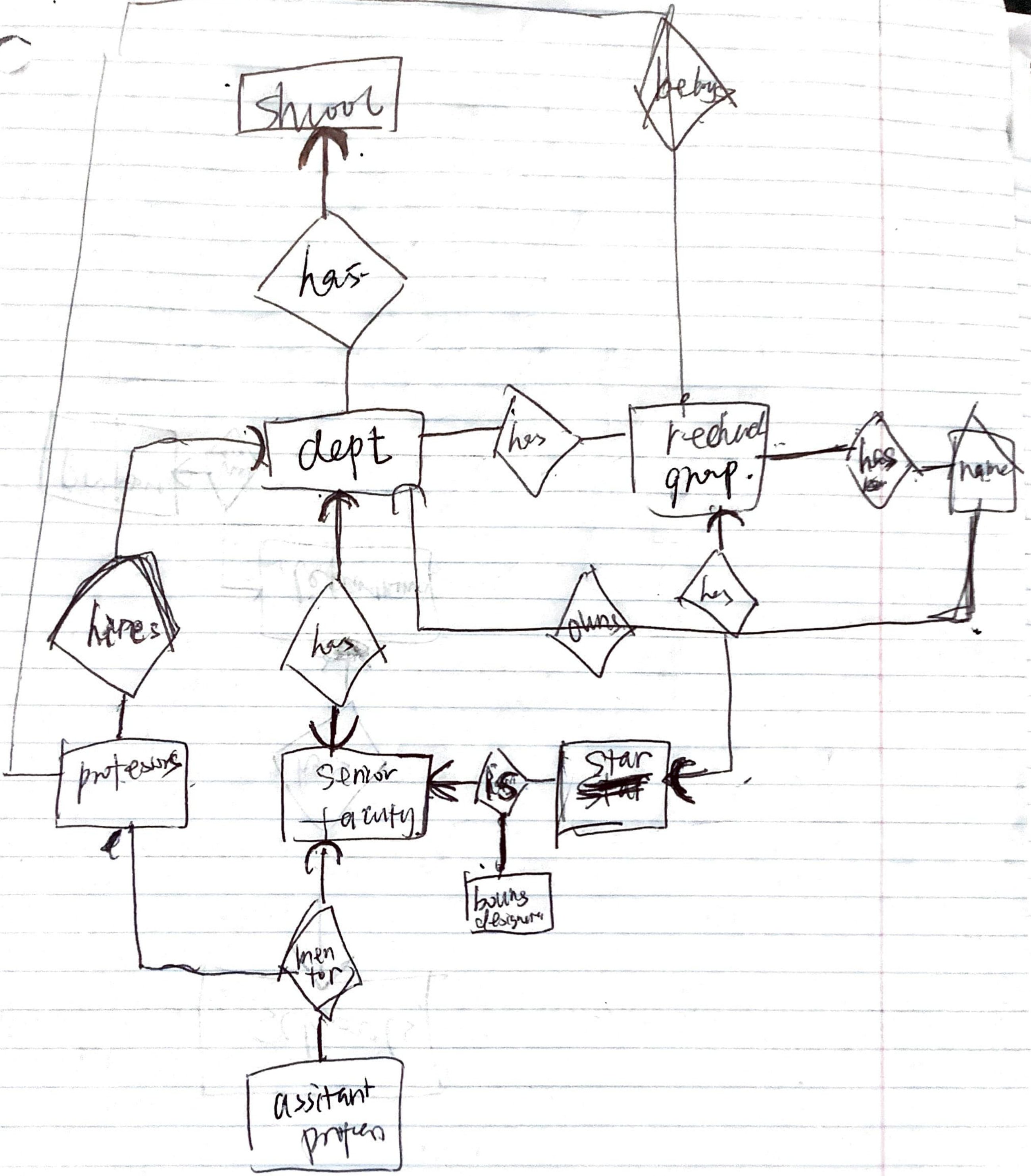
Second Generation :
Assembly — it is little further from binary
So this is like APD AX, 01.
it is more understandble, small size
but need assemble.

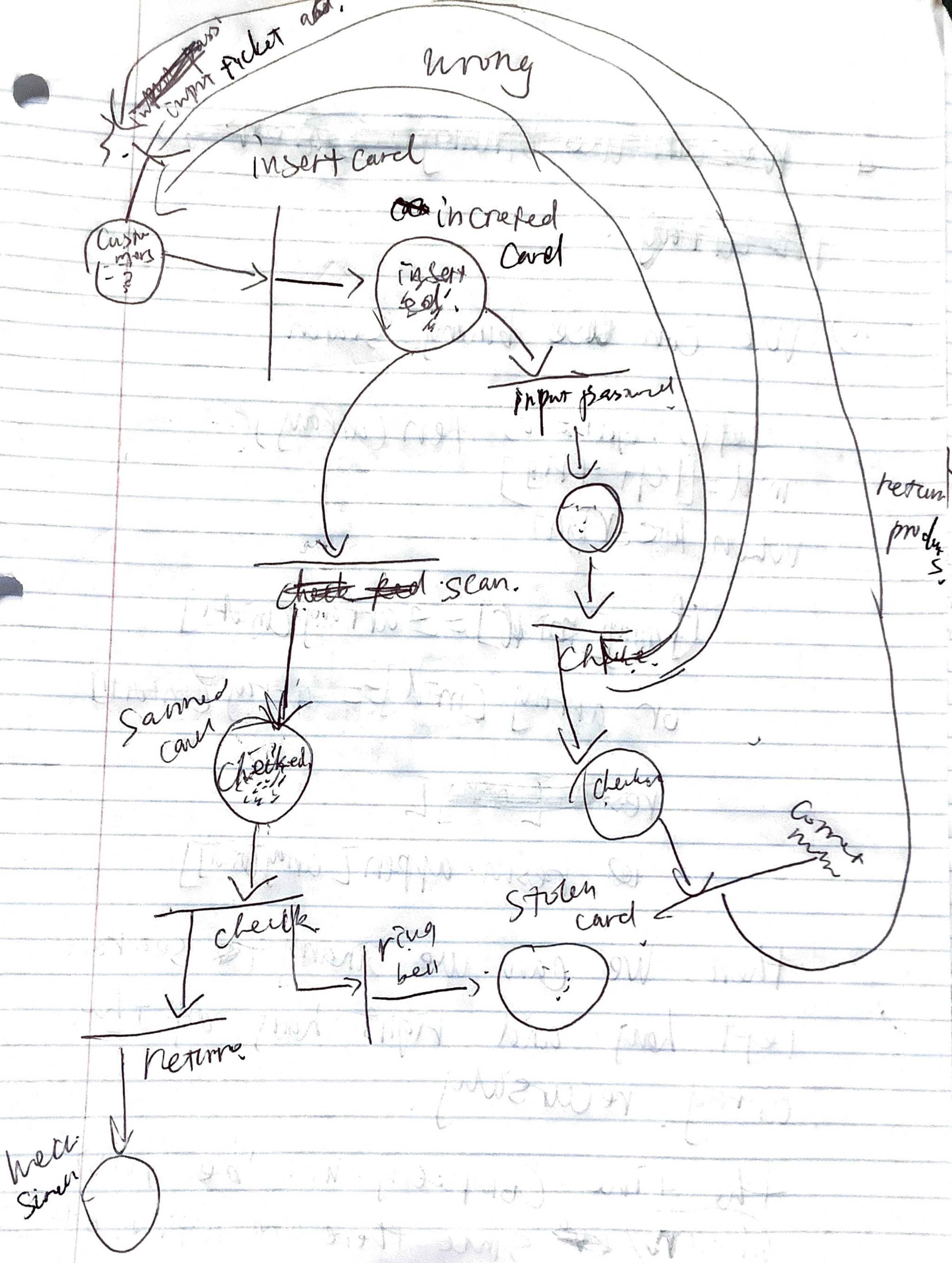
Third condition
procedure — mathementicians found out
that coding PL can be Turring machine
equavalent.

Fourth generation something like. OO and SQL.

using OO you can use creat modeles easy
abstraction easy data analysis but
the most important is because of inheritance

using SQL follows functional programing.


beby
shool
has
dept
has
reseach grup.
has
name
hires
has
owns
has
professors
Senior faculty
is
Star
~~Staf~~
bouns designation
men tor
assitant profess


~~with pass~~ input ticket ~~and~~.
wrong
insert card
~~oo~~ increted card
Customers - ?
insert ~~card~~
input pasword
~~check read~~ scan.
Chek.
return produs.
Scanned card
Checked
checker
Correct num
stolen card
check
ring bell
return.
well sined

~~search to~~

~~the array~~.

a) We can use binary search

```
  left, right = 0, len(array)
mid = [left + rig]
when left <= right

    if array[mid] == array[mid-1]
     or array[mid] == array[mid+1]

      res~~ult [array]~~ [

      result.appen[array[mid]]
```

then we can use binary to search left half and right half of the array. recursily.

the time complexity will be just nlog n since there are n times of binary search

So we could

design a base case for the algrithm. say:

```
when the n==0: retun 1
        then if n % 2==1:
            we cand do a* femotion (a, n-1)
        else:
                p= function(a, n/2)
            return p*p
```

the time complicity will be smaller then O(n) because in some steps will dived the exponent so, it will be

$\log n \leq$ ~~O(log~~ time complexity $\leq n$.