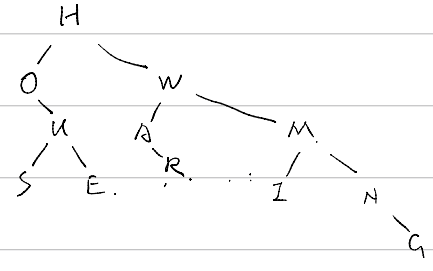
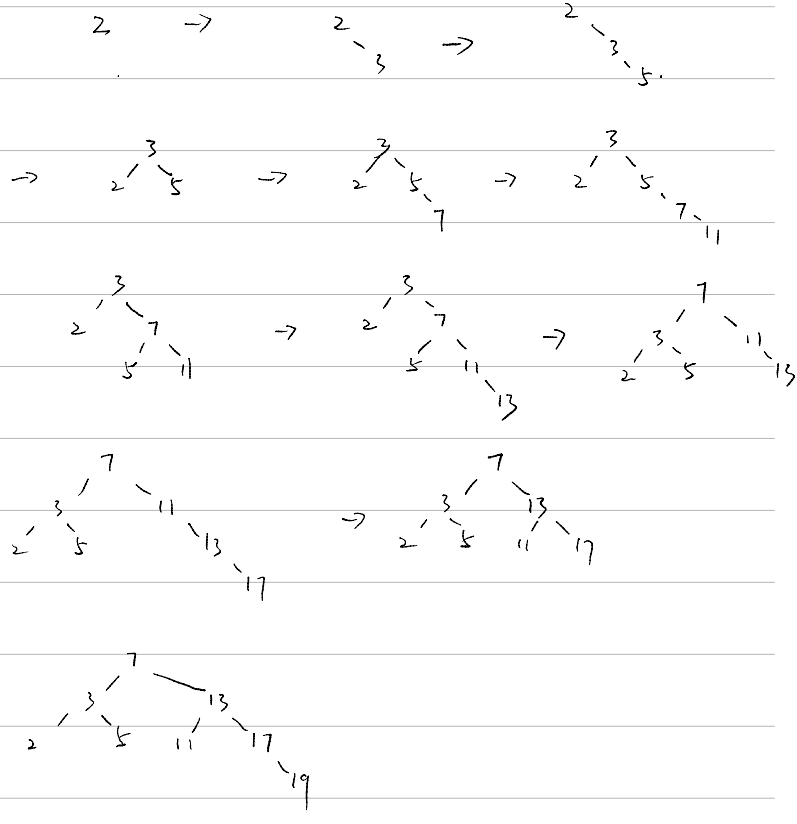
日期: / 3:20

1.(a)



1.(b)



日期: /

这题 push 和 pop 有没有 heapify 的操作啊，还是单纯插入和输出。（还是自己写一遍锻炼一下）

1. (c) Algorithm:

~~Build_Min_Heap (A):~~
~~for i <- [n/2]+1 to 0~~
~~Min-heapify (A, i)~~

```
Build_Min_Heap (A):
    res = list()
    for val in A:
        res.push(val)
    return res
```

```
Select_kth_Min (A, k)
    A <- Build_Min_Heap (A)
    for i in 0 to k-1
        kth_min = A.pop()
    return kth_min
```

Complexity: Buld_Min_Heap = O(n)

select_kth_Min = O(k·(logn) + O(n)

2. (a)

```
combine (cur, new):
    cur.next = new.
    new.pre = cur
    return new

Merge (L, R):
    if L.key < R.key:
        cur = L, head = cur
        lnew = L.next
        rnew = R.
    else:
        cur = R, head = cur
        lnew.=L, rnew = R.next
```

日期: /

```
while lnew not None or rnew not None:
    if lnew is None:
        cur = combine(cur, rnew)
        rnew = rnew.next
    elif rnew is None:
        cur = combine(cur, lnew)
        lnew = lnew.next
    else:
        if lnew.key < rnew.key
            cur = combine(cur, lnew)
            lnew = lnew.next
        else:
            cur = combine(cur, rnew)
            rnew = rnew.next.
return head
```

2(b)

```
Check:
    stack = list()
    for bracket in sequence:
        if bracket is "(":
            stack.push("(")
        else:
            symbol ← stack.pop()
            if symbol is ")":
                raise error
```

日期: /

```
if    stack.size() > 0 :
      raise error
print( is matched)
```

3.(a) (i)

```
select r.person_id
from Review r
inner join
(select movie_id from Movie
where publisher = "KEVIN FEIGE") m
on r.movie_id = m.movie_id
group by r.person_id
having count(*) >= 2;
```

(ii)

```
select Avg(m.budget)
from Movie m
inner join
(select r.movie_id
from Role r
inner join
( select person_id
  from Person
  where name = "CHRISTIAAN BALE") p
on r.person_id = p.person_id) r2
on m.movie_id = r2.movie_id
```

3.(b) (i)

$M1 \leftarrow \Pi_{movie\_id}\ \sigma_{publisher\ !=\ "KEVIN\ FEIGE"}(Movie)$

$\Pi_{person\_id}(Review) - \Pi_{person\_id}(Review \bowtie M1)$

日期: /

(ii) $M2 \leftarrow \Pi_{movie\_id}\, \sigma_{publisher = \text{"KEVIN FEIGE"}}(Movie)$

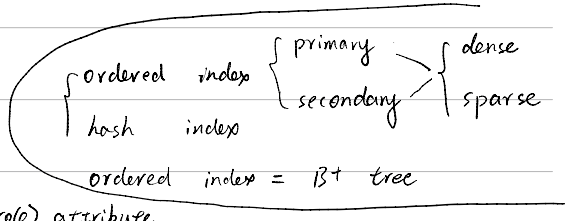
$R\_watch \leftarrow \Pi_{person\_id,\, movie\_id}(Review) \bowtie M2$

$R\_unwatch \leftarrow \Pi_{person\_id,}(Review) \times M2 - R\_watch$

$P \leftarrow \Pi_{person\_id}(Review) - \Pi_{person\_id}(R\_unwatch)$

$\Pi_{name}(P \bowtie \Pi_{person\_id, name}(Person))$

3.(c)



Role save as dense primary index on (role) attribute,
(movie_id, person_id) as secondary index

Person save as

dense primary index on (name) attribute.
secondary index on (person_id) attribute.

Review save as
hash index, compute hash values on (person_id, movie_id).

日期: /

4(a) △ doctor: (doctor_id , name, room_number, gender)

△ user : (user_id , name , gender , date_of_birth)

△ email : (email_address , user_id)

since email is multi. value attribute, it need 2 attributes as primary key.

△ appointment: (doctor_id , user_id , appointment_datetime)

since appointment is (many to many) relationship, it need 2 attribute as primary key.

日期: /

4 (b) ① 1 attribute :

$A^+ = (A, C, E)$ not candidate key.

$B^+ = (B, D)$ no

$C^+ = (C)$, $D^+ = (D)$, $E^+ = (E)$ no

② 2 attributes :

$(AB)^+ = (A, B, C, E, D)$, yes

$(CD)^+ = (ABCDE)$, yes

$(AD)^+ = (ABCDE)$, yes

$(BC)^+ = (ABCDE)$, yes

③ 3 attribute :

if attribute set doesn't contain A,

it must contain $\{CD\}$, so it can infer to A.

but $\{CD\}$ is candidate key.

consider $\{ACE\}^+ = (ACE)$, no

In conclusion (AB), (CD), (AD), (BC).

日期: /

4(c) check: $A \to C$, it's not trivial, A is not superkey of R

so R is not BCNF.

decompose: ① consider R:

$A \to C$ not trival, $A \cap C = \phi$,

A is not superkey

set = $\{(A, B, D, E), (AC)\}$

② consider $(ABDE)$

$A \to E$ not trivial, $A \cap E = \phi$.

A is not super key.

set = $\{(ABD), (AE), (AC)\}$

③ consider (ABD)

$B \to D$ not trivial, $B \cap D = \phi$

B is not super key.

set = $\{(AB), (BD), (AE), (AC)\}$

BCNF := $\{(AB), (BD), (AE), (AC)\}$