1)

I[even i] : beginning position of an interval $(i/2-1)$

I[odd i] : ending

I is the set of endpoints

I[0] is the valid size of the memory of I.

I[1] is the size of endpoints

I. b e

indexB, indexE

bsearch_int64_t

if indexB is odd #, b is within the $\{(indexB-1)/2\}^{th}$ interval

if indexB is even # & b ∈ I, b is within the $\{(indexB-2)/2\}^{th}$ interval

else b is out of any interval in I.

b, e inside some intervals in I

$$db = \begin{cases} \frac{indexB-1}{2} & \text{if indexB is odd.} \\ \frac{indexB-2}{2} & \text{if indexB is even.} \end{cases}$$

$$de = \begin{cases} \text{use indexE.} \end{cases}$$

$k = de - db.$

bB — b — bE / eB — e — eE

use pointers

bB = I[2*indexB + 2] / eB = I[2*indexE + 2]

bE = I[2*indexB + 3] / eE = I[2*indexE + 3]

Replace indexB/indexE with db/de.

if k = 0, make sure that *bB ≤ b ≤ *bE

*eB ≤ e ≤ *eE using assert.

else *bE = *eE ?

tailSize = sizeI − (de*2 + 1)

memmove(bE, eE, tailSize * sizeof(int64_t))

I[1] −= k;

after

(2)

II. i) 

$db = de$

e is out of intervals. even $de = (indexE - 2)/2$

int64_t *bB = I + 2*indexB + 2

bE = I + 2*indexB + 3

eE = I + 2*indexE + 3

eB = I + 2*indexE + 4

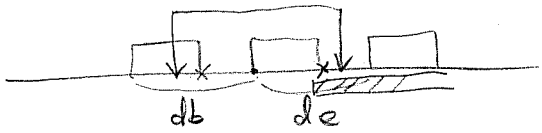
if $2*indexE+4 < sizeI + 2$, else eB does not exist. or eB = NULL

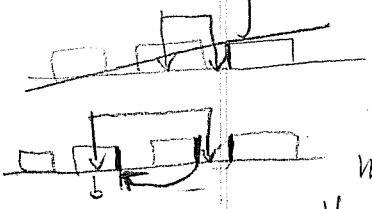
if k=0 if eB exists, *bE = e

Assert

$*bB \leq b \leq *bE$

$*eE < e < *eB$

if k>0, 

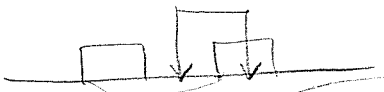


tailSize = sizeI − (2*de+1)

memmove (bE, eE, tailSize * sizeof (int64_t))

then *bE = e

I[1] -= k

III. 

if $2*indexB+1 < 2$, bE = NULL

bE = I + 2*indexB + 1

bB = I + 2*indexB + 2

eB = I + 2*indexE + 2

eE = I + 2*indexE + 3

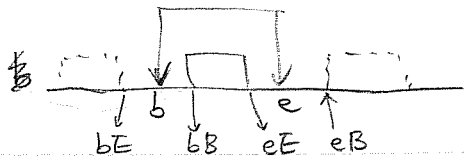
if k=0, *eB = b

else, tailSize = sizeI − (2*de)

memmove (bB, eB, tailSize * sizeof (int64_t))

I[1] -= k

③ Ⅳ.



bE = I + 2*indexB + 1 if 2*indexB+1 > 2, otherwise NULL

bB = I + 2*indexB + 2

eE = I + 2*indexE + 1

eB = I + 2*indexE + 3 if 2*indexE+3 < sizeI+2, otherwise NU

compute db, de

k = de - db tailSize = sizeI - (2*de+2)

if k = 0, memmove(eB, bB, tailSize * sizeof(int_64_t))

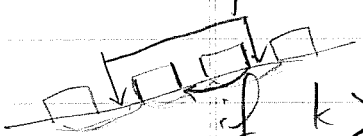
*bB = b

*eE = e

I[1] += 1.

if k = 1, *bB = b

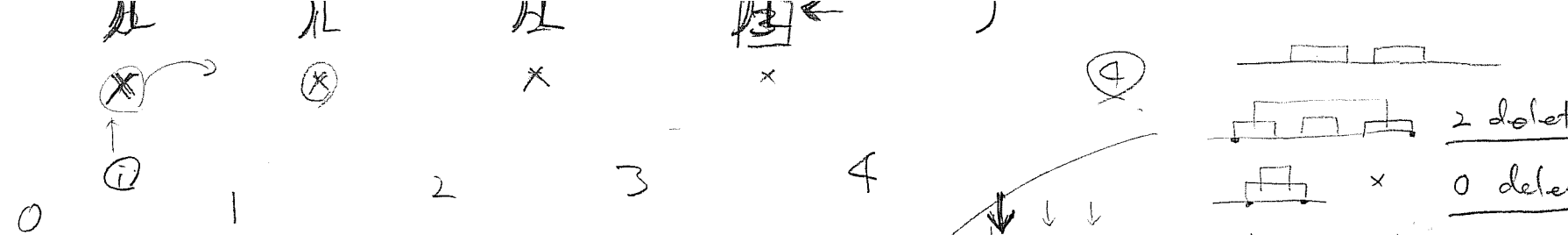
*eE = e



if k > 1, tailSize = sizeI - (2*de+1)

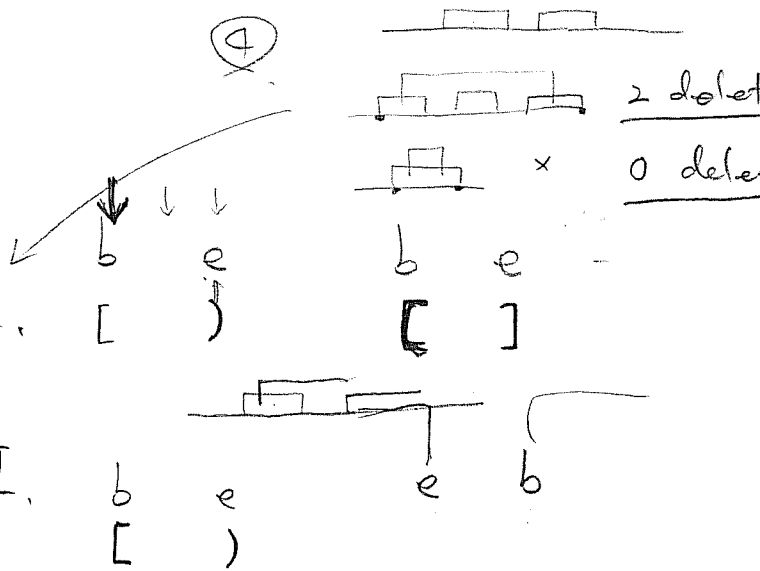
memmove(bB+1, eE, tailSize * sizeof)

I[1] -= (k-1);



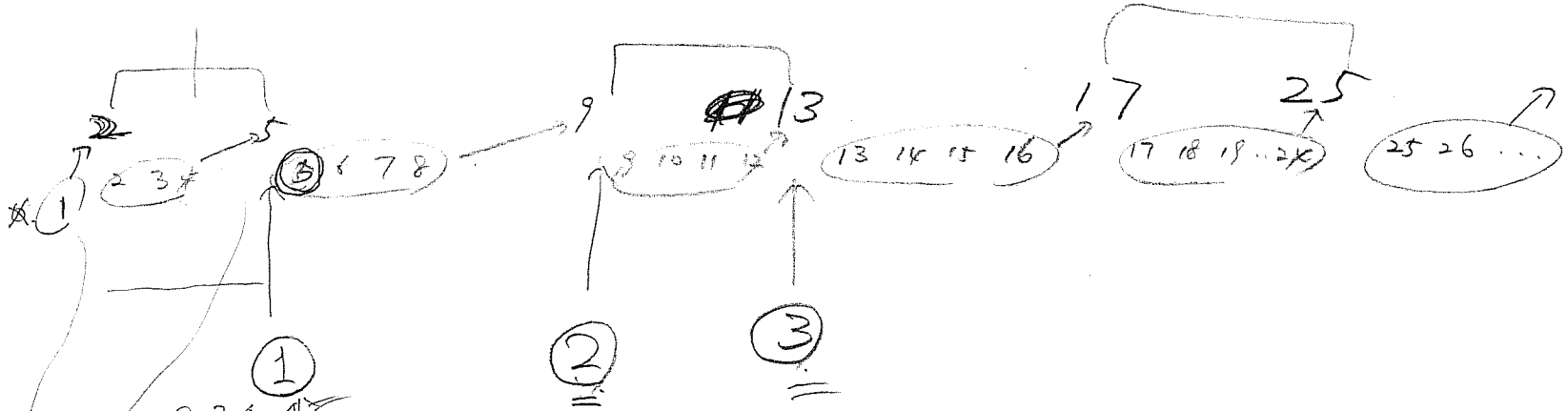
$V[r-1] \leq key < V[r]$

if $r < nel$



if $key < min(V)$, $r = 0$

if $key \geq max(V)$, $r = nel$.

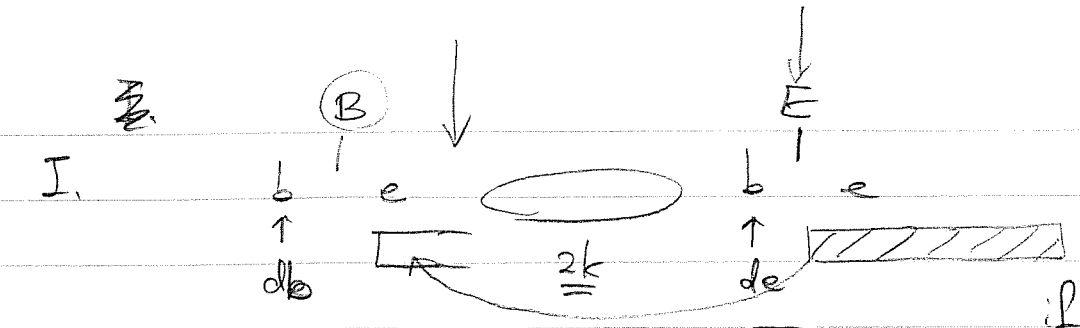


$\begin{pmatrix} 2* \\ 2*+1 \end{pmatrix}$ deardb

if odd # index, within an interval $(index+1)/2$ ← interval index

if even # index & key ∈ endpoints, with an interval $(index-2)/2$

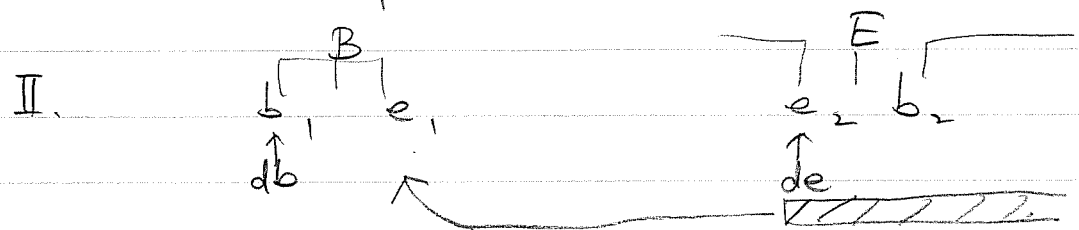
else $(index-2)/2$



$de - db = 2k$ $\boxed{k \geq 0}$

if $k=0$, Nothing.
if $k=1$, delete.

sizeEndpoint -= $2k$.

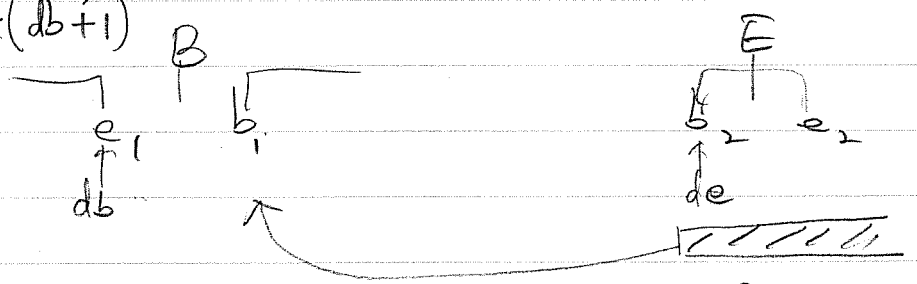


$de - db = 2k+1$ $k \geq 0$

if $k=0$, $e_1 = e_2$
if $k \geq 1$, delete.

~~B~~ $e \in E$

$*(db+1)$

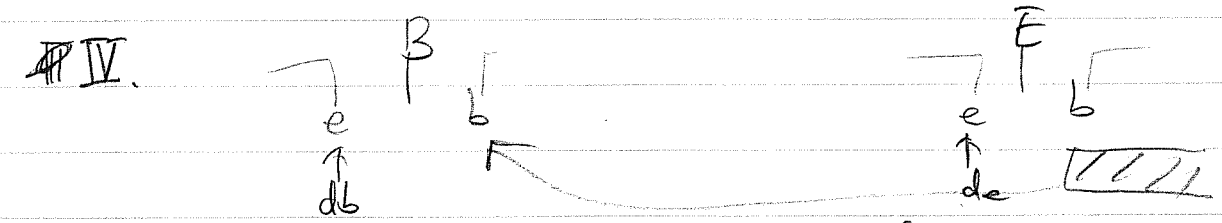


$de - db = 2k+1$ $k \geq 0$

if $k=0$, Nothing. $b_1 = b$
if $k=1$, delete

~~B~~ $b \in B$

$*(db+1)$



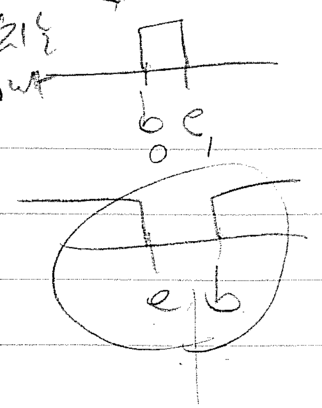
$de - db = 2k$ $k \geq 0$

if $k=0$ insert two points
if $k=1$ use the two poin
if $k \geq 2$ delete

$*(db+1) \in B$

$*(de) \in E$

except ① 정리는 pow ②



I. No left special, No right special

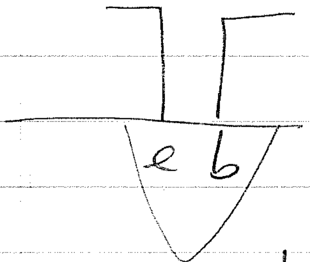
II. No left special, de → last element.

III. db = −1, No right special.

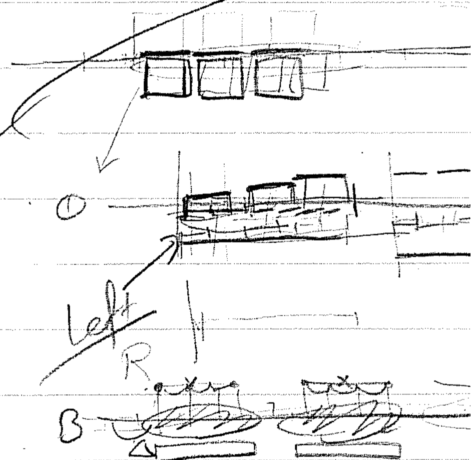
IV.

① db de ⇒ 없애는 경우 or search 고치는 경우

②



two intervals ~~diff.~~



① ~~Use R to build interval~~

② Build interval tree using B.

③ Periodic points in R are used as query to the interval tree

singbase diff