**Strategy​ ​and​ ​algorithm​**

**Strategy**: With every phone number use hashtable to store data (key is phone number, value is array activation date-deactivationdate and sort type ASC with every array).

Find max deactivation date of the number,

if not exists then activationdate is min of list activation date.

If exists deactivation date, search minimum of activationdate (list of activation date > deactivation date).

**Algorithm**

Linear search, Merge sort (Class MergeSort).

**Data structure**: Hashtable(Class Hashtable), Dynamic array (Class Item).

**Memory complexity:**

Hashtable: space O(n) (https://en.wikipedia.org/wiki/Hash\_table)

Dynamic array: space O(n) (https://en.wikipedia.org/wiki/Dynamic\_array)

With N = total record,

With U = total unique phone number

With V = total activation date of phone number

Hashtable stored U key (value is dynamic array),

One Dynamic array has V element.

<=> complexity memory = V1 + V2 + ... + Vn = N.

(n = U)

**Time complexity**

Hashtable: O(n) (https://en.wikipedia.org/wiki/Hash\_table)

Dynamic array: O(n) (https://en.wikipedia.org/wiki/Dynamic\_array)

With N = total record,

With U = total unique phone number

With V = total activation date of phone number

Method createItem call N times.

Hashtable insert U key.

Dynamic array add V times.

<=> time insert = V1 + V2 + ... + Vn = N;

(n = U)

Method write has U loops.

With every loop call 2 times merge sort (VlogV) and one time linear search(V).

<=> write time = (2\*V1logV1 + V1) + ... + (2\*VnlogVn + Vn)

<=> write time = V1 (2\*logV1+1) + ... + Vn (2\*logVn+1)

With Vx < N

<=> write time < V1 (2\*logN+1) + ... + Vn (2\*logN+1)

<=> write time < (2\*logN+1) (V1 + ... + Vn )

(n = U)

<=> write time < N(2\*logN+1)

<=> time complexity < N(2\*logN+1) + N

<=> time complexity < 2(N\*logN + N)

**Unit test**

Using junit framework (java library, use maven to management dependency

library)

Class AppTest (method testApp)

With date input format YYYY-MM-DD, using compare String to sort.