implement a bidirectional Map (over a Hash)

init $\Theta(1)$

search (k) => v -> $\Theta(1)$ on average

reverse_search (v) => k

insert (k,v) => $\Theta(1)$ on average and amortized

remove (k) => v -> $\Theta(1)$ on average

(k, v)

TKey, TValue

NULL_TKey, NULL_TValue

- Keys are unique, also the values

BSMAP:

m : integer

key_table : (↑ BSM Node) [ ]

value_table : (↑ BSM Node)[ ]

h_key : TFunction

h_value : TFunction

BSM Node:

K : TKey

V : TValue

next_key : ↑ BSM Node

prev_key : ↑ BSM Node

next_value : ↑ BSM Node

prev_value : ↑ BSM Node

search (, k) →

⇒ Function findKey (bdm , k) ⇒ returns pNode → pointer to the node

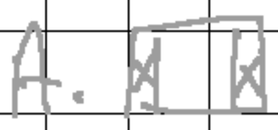
reverseSearch (, v)

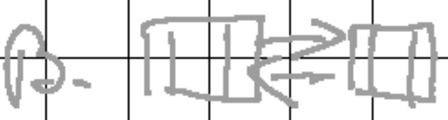
```
Function findkey ( hdm , k)
    pos ← hdm - h_key(k)
    pNode ← hdm . key_table [pos]
    while (pNode ≠ NIL) AND ([pNode. k ≠ k]) execute
        pNode ← [pNode]. next_key
    endwhile
    findkey ← pNode
end_function
```

remove (, K) ← removeNode (, pNode)

```
Function remove (hdm, k)
    node ← findkey (hdm, k)
    if node = NIL
        val = NULL_TValue
    else
        val ← [node].k
        removeNode (hdm, node)
    remove ← val
end function
```

```
Subalg   removeNode (hdm , pNode)
    pos ← hdm . h_key ([pNode].K)
    if ([pNode]. prev-key ≠ NIL) AND
       ([pNode]. next-key ≠ NIL) then
        pprev ← [pNode]. prev-key
        pnext ← [pNode]. next-key
        [pprev]. next-key ← pnext
        [pnext]. prev-key ← pprev
    else if ([pNode]. prev-key = NIL) then
        if [pNode]. next-key = NIL then
            hdm. key-table [pos] = NIL
        else
            [[pNode]. next-key ]. prev-key ← NIL
            hdm. key-table [pos] ← [pNode ]. next-key
        endif
    else //c
        [[pNode]. prev-key ]. next-key ← NIL
    endif
    @ same for value list (& hashtable)
        free (pNode)
endSubalg
```

A. 

B. 

C. 

D. 

Sortiod Map – represented on a hash table.

Sortiod Map:

m: integer

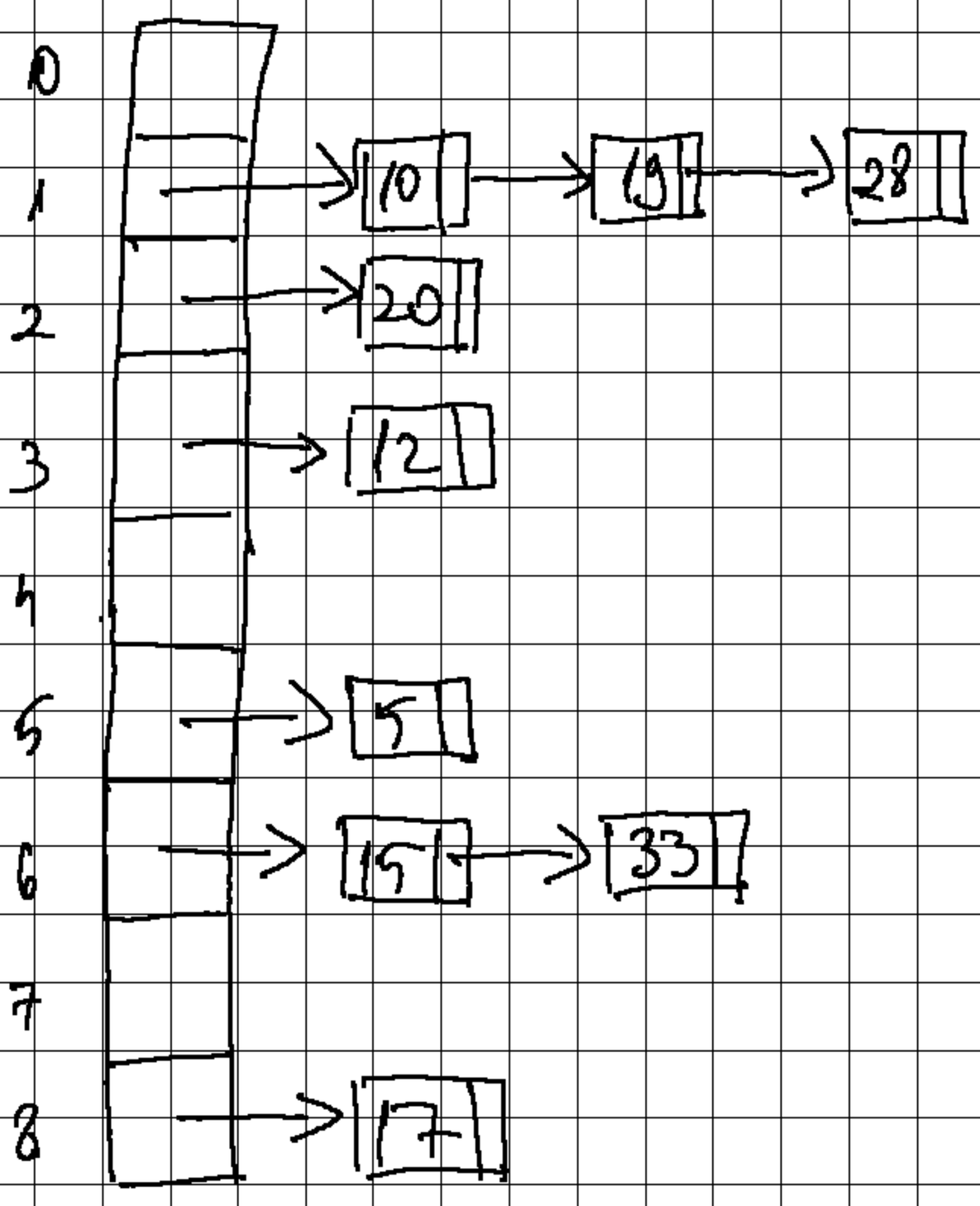
T: (↑Node) [ ] ← key list sortiod

h: Function

rel: relation

m = 9

k	h(k)
5	5
28	1
13	1
15	6
20	2
33	6
12	3
17	8
10	1



- merge : list1 + list2 => list12

list12 + list34 => ....

=> $\Theta(m*n)$

MIN. heap: → n log n

Node: linked

K: TKey

next: ↑Node

nextSorted: ↑Node

prevSorted: ↑Node

SortedMap:

m: integer

T: (↑Node)[ ] ← key list sorted

h: TFunction

rel: relation

head_sorted: ↑Node