12/13 I/O

1. BufferReader

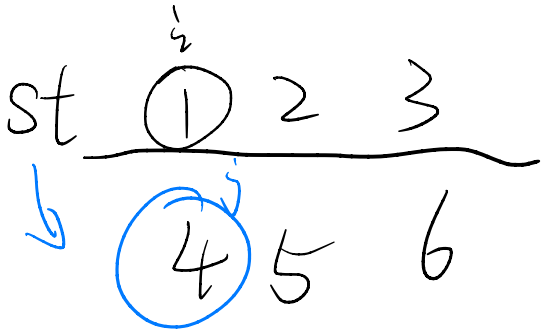
1) 1 2 3 4 — Integers

```
1 → StringTokenizer st =
        new StringTokenizer(f.readline()); (1)

int i = Integer.parseInt(st.nextToken());

      st =
→ new StringTokenizer(f.readline()); (2)

int j = In.....;
```



7

exe 8

2) String

1 2 3 4 5 6 7

S → abc de \n

S1 → ij ii \n

S = ?

S1 = ?

S.length() → 6

S1.length() → 5

charAt( )

3) char

a b c

st = new S..T...(f.readline())

char c = Character.ParseChar(St.)

2. PrintWriter.

BufferedWriter bw = new BufferedWriter
(new FileWriter(""));

PrintWriter out = new
PrintWriter(bw);

out.print("~"); → to file.

out.close();

ASCII

A - Z
a - z
0 - 9
! ? _ ~

→ ASCII

```
cook@pop-os:~$ ascii -d
    0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
    1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
    2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
    3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
    4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
    5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
    6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
    7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
    8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
    9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
   10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
   11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
   12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
   13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
   14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
   15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

char A → int

(int) A → 65

int A_i = (int) A − 64

65

0 1 2 3 4 5

COMETQ.

65 — 90

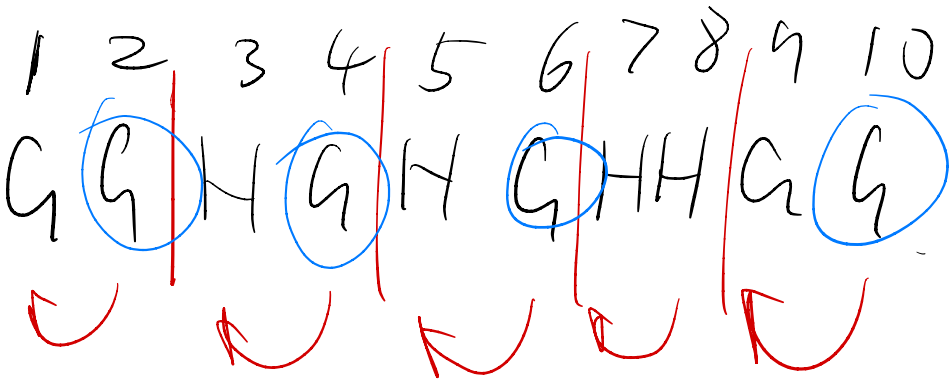
A - Z

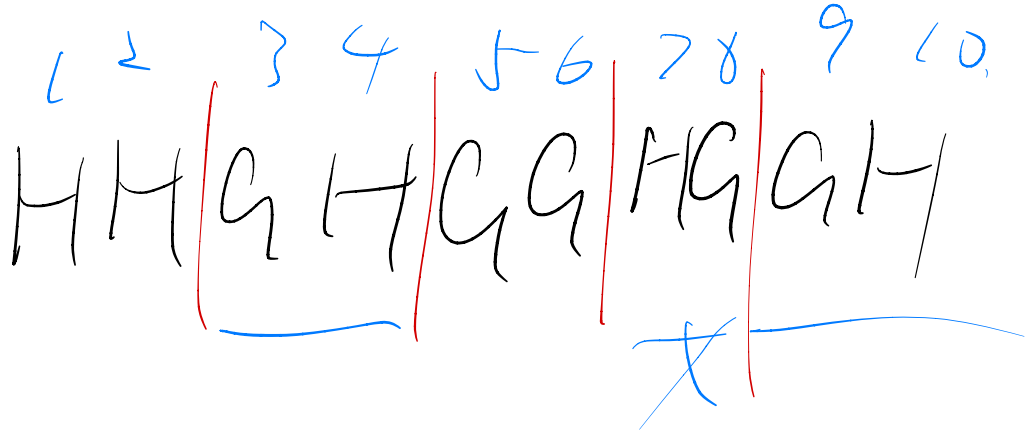
① charAt.

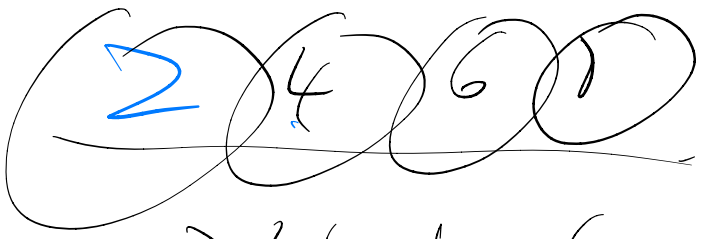
② (int) char − 64 1 — 26.

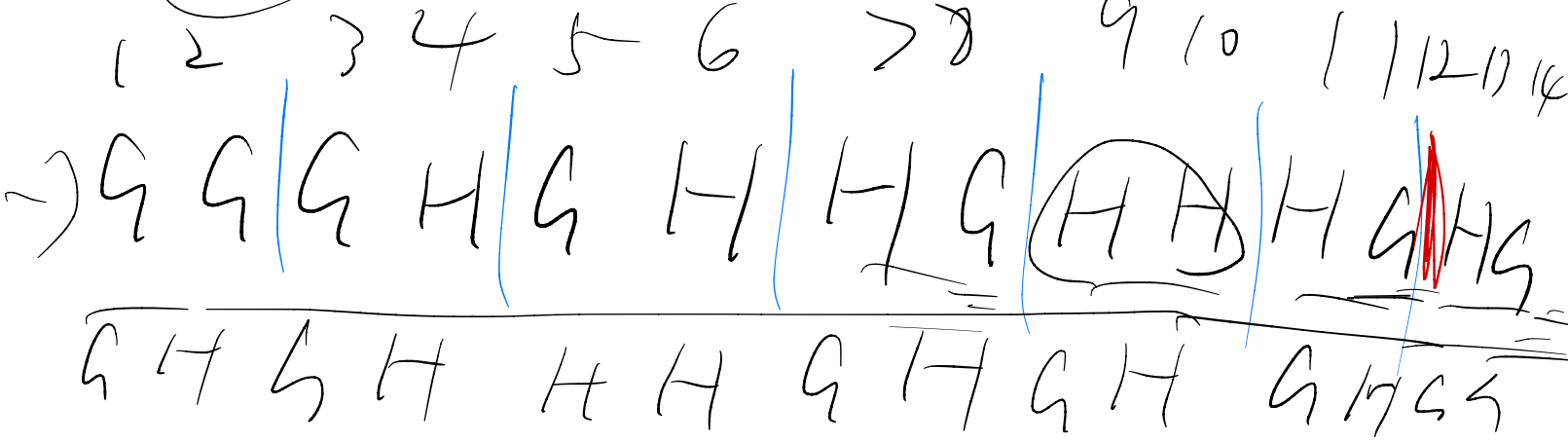
③ Calc.

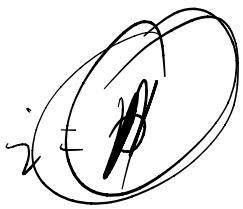
1.





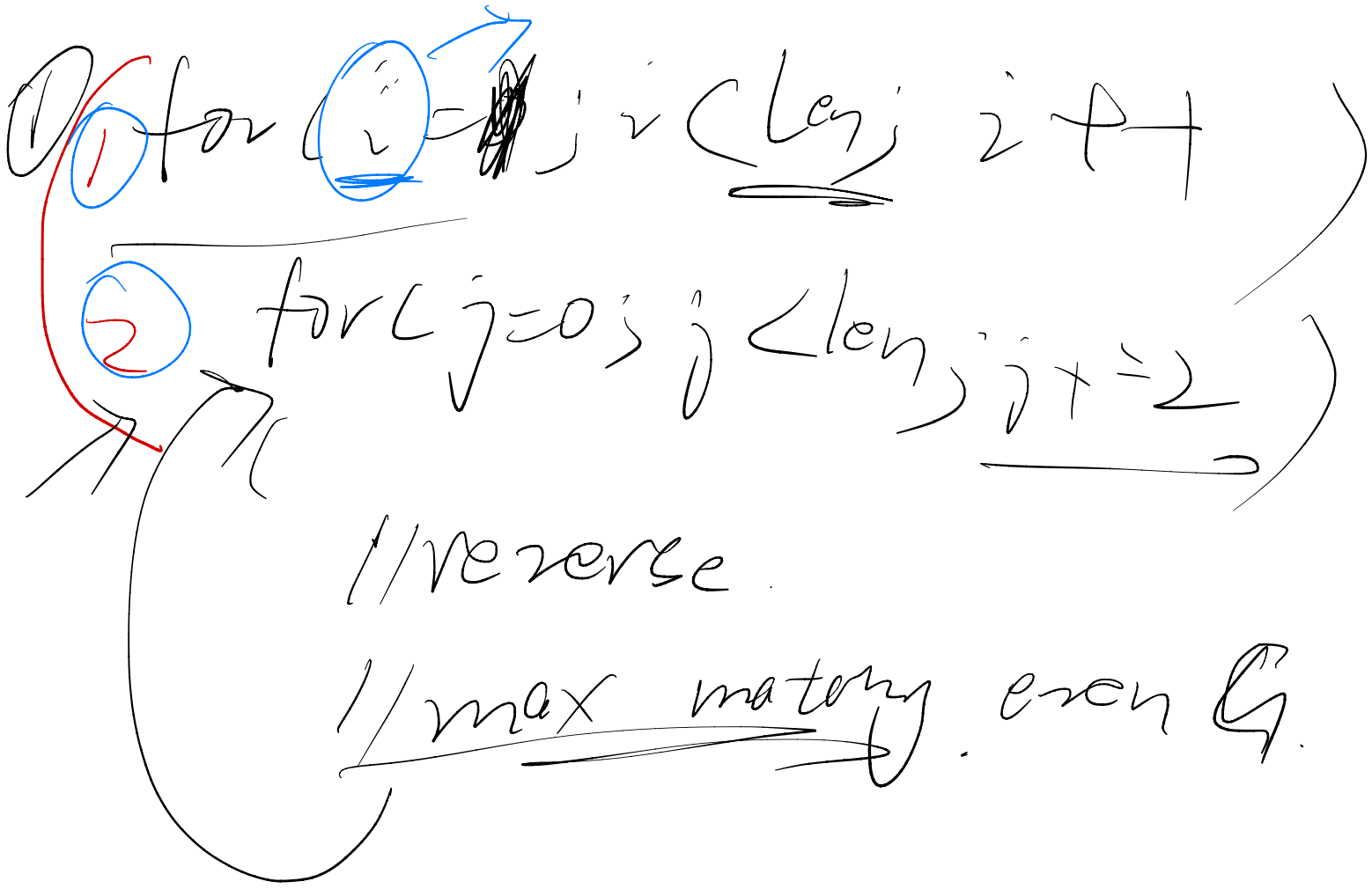






j=4. 6. 8 10. 12. 14.

max=4 6. 3. 3. 2 1.

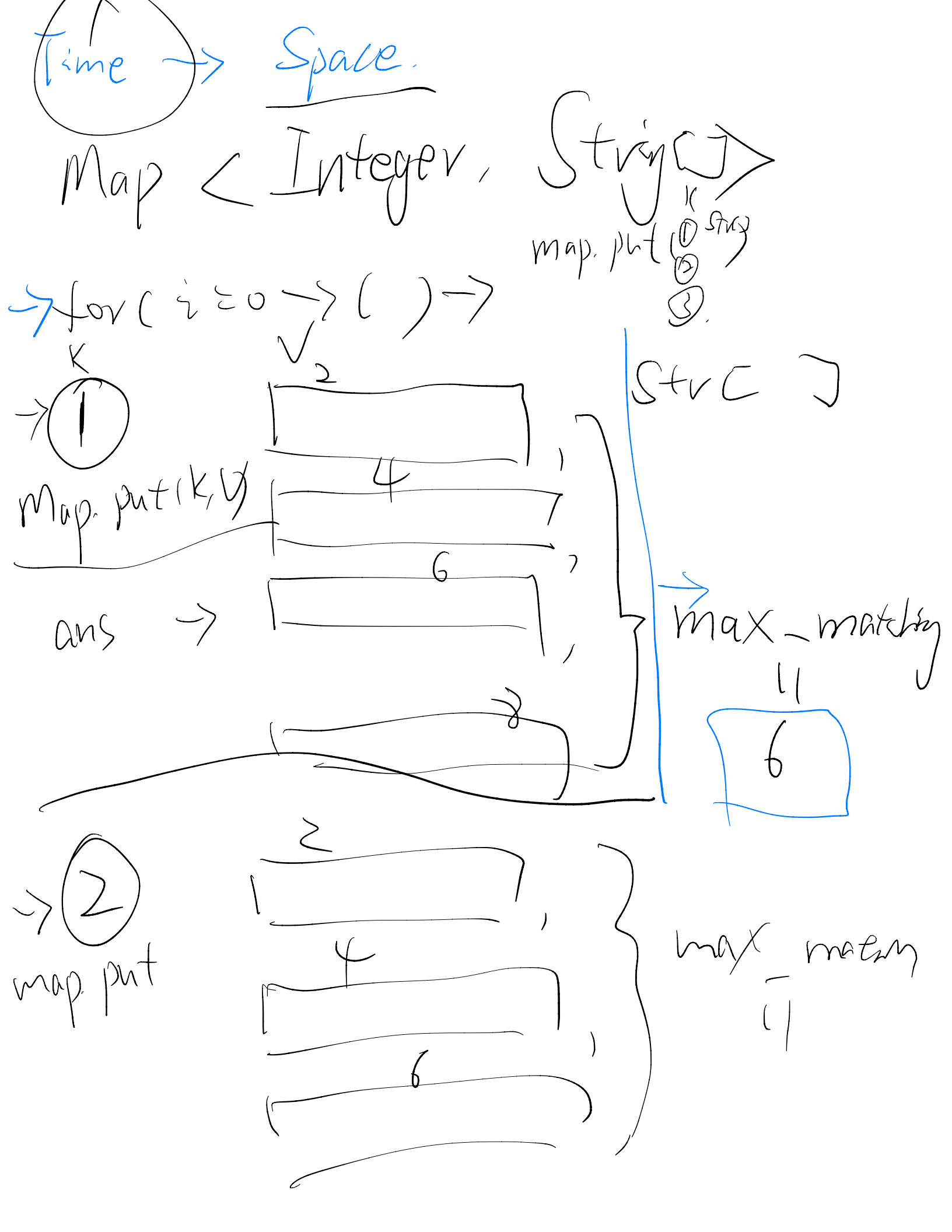


)

)

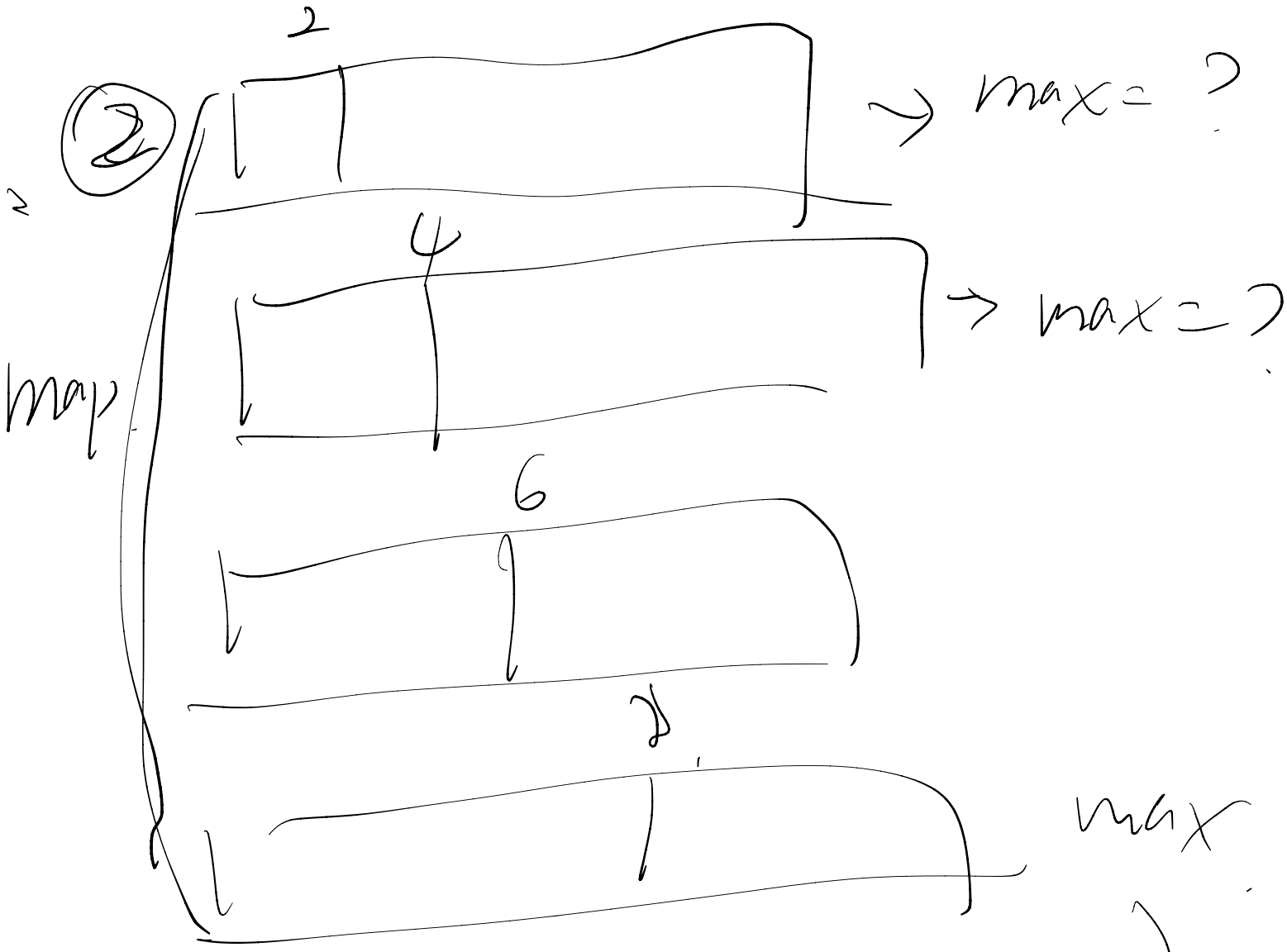
print(i)

Time exceeds the req


Time → Space.
Map < Integer, String >
map.put(1, str)
for(i=0 → ( ) →
K
V
1
2
4
6
8
Map.put(k,v)
ans →
Str[ ]
max_matching
11
6
2
map.put
2
4
6
max_matching
11

for (1 → ∞) { ← 2,

① max_match = 0,



② map.put (k, max) ←.

}

If (max_new <= max_old)
    break;
    return key.
}

G G G H G H H G H H G H G

G H H G H G G G

even not G → G max

ans = 2

2

ans : 0

for(j = len-2; j >= 0; j -= 2)

if(str[j] != str[j+1]){

str[j] == G) ==

(ans % 2 == 0.

ans++; }

)

12/14

$O(n)$ → while (i=0 → n)
For (i=0 → n)

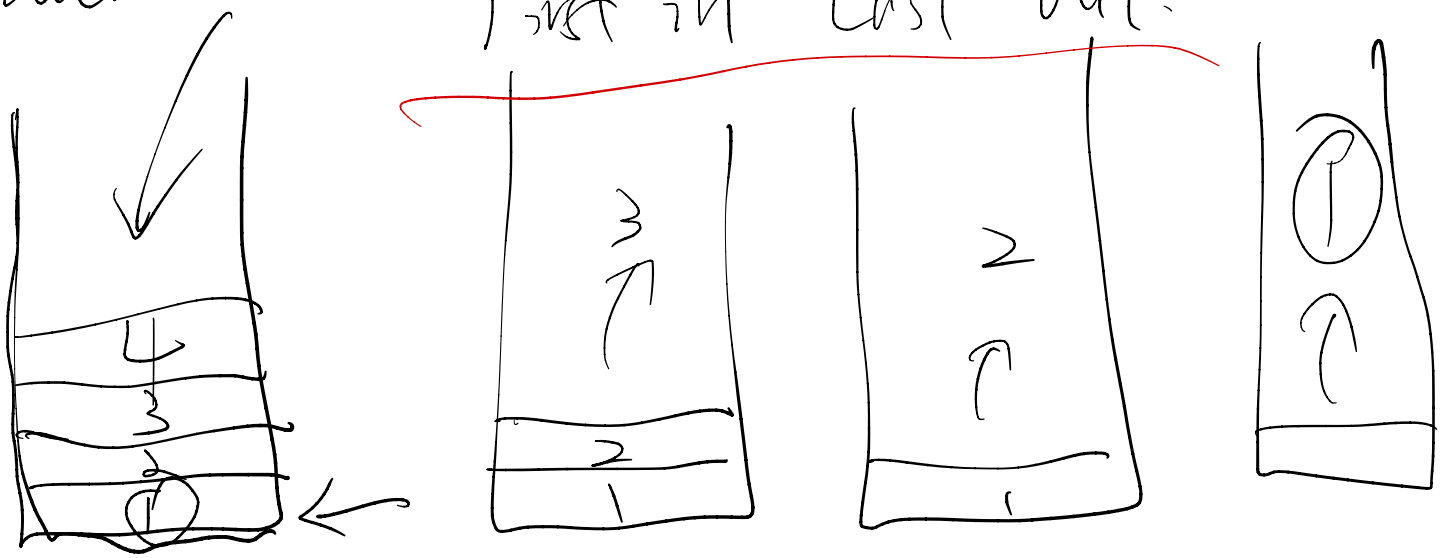
$O(n^2)$ → while (i=0 → n)
↳ For (i=0 → n)

Data Structure

arrays.
Array Lists → Stack
queue
double queue
linked list
map / set

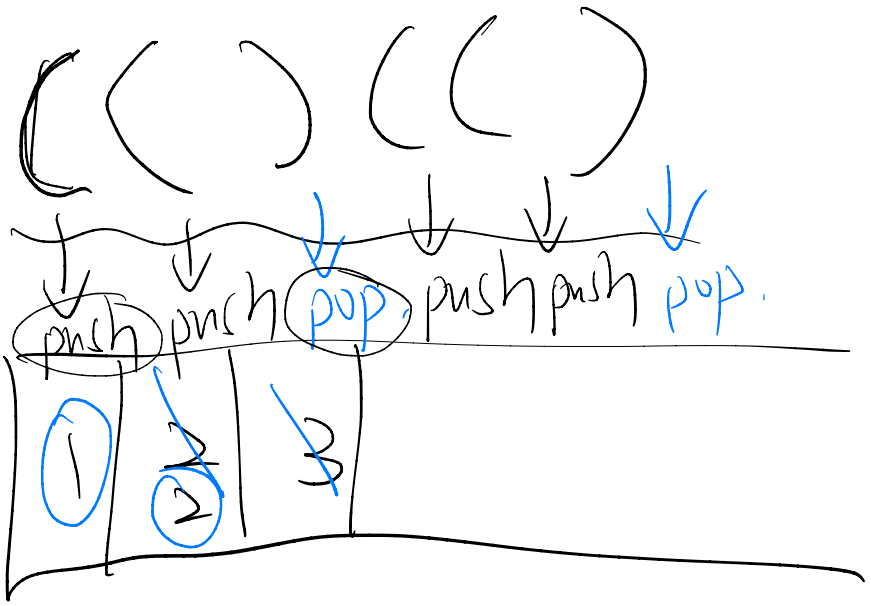
Stack

First in Last out.

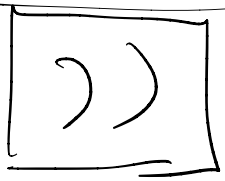


Stack → pop() push()

```
Stack s = new Stack();
s.pop();   s.push();
```

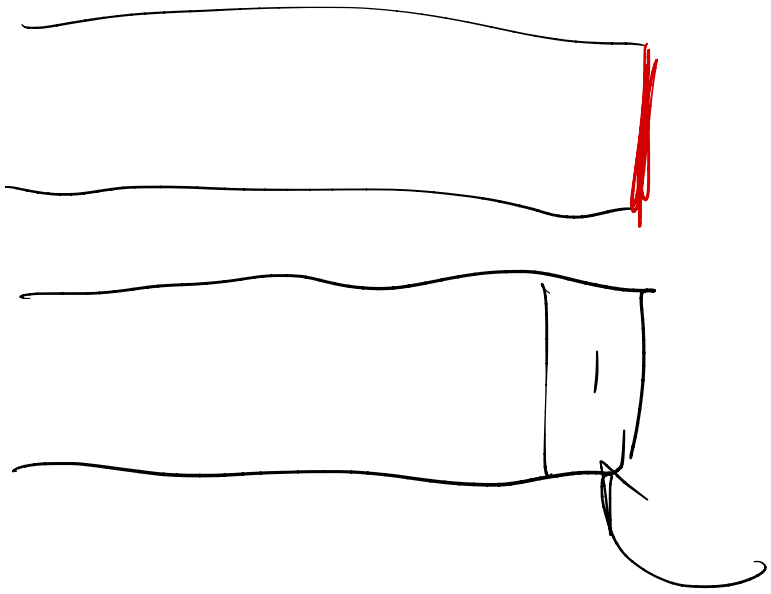


s.size()



O(n)

queue / DQ

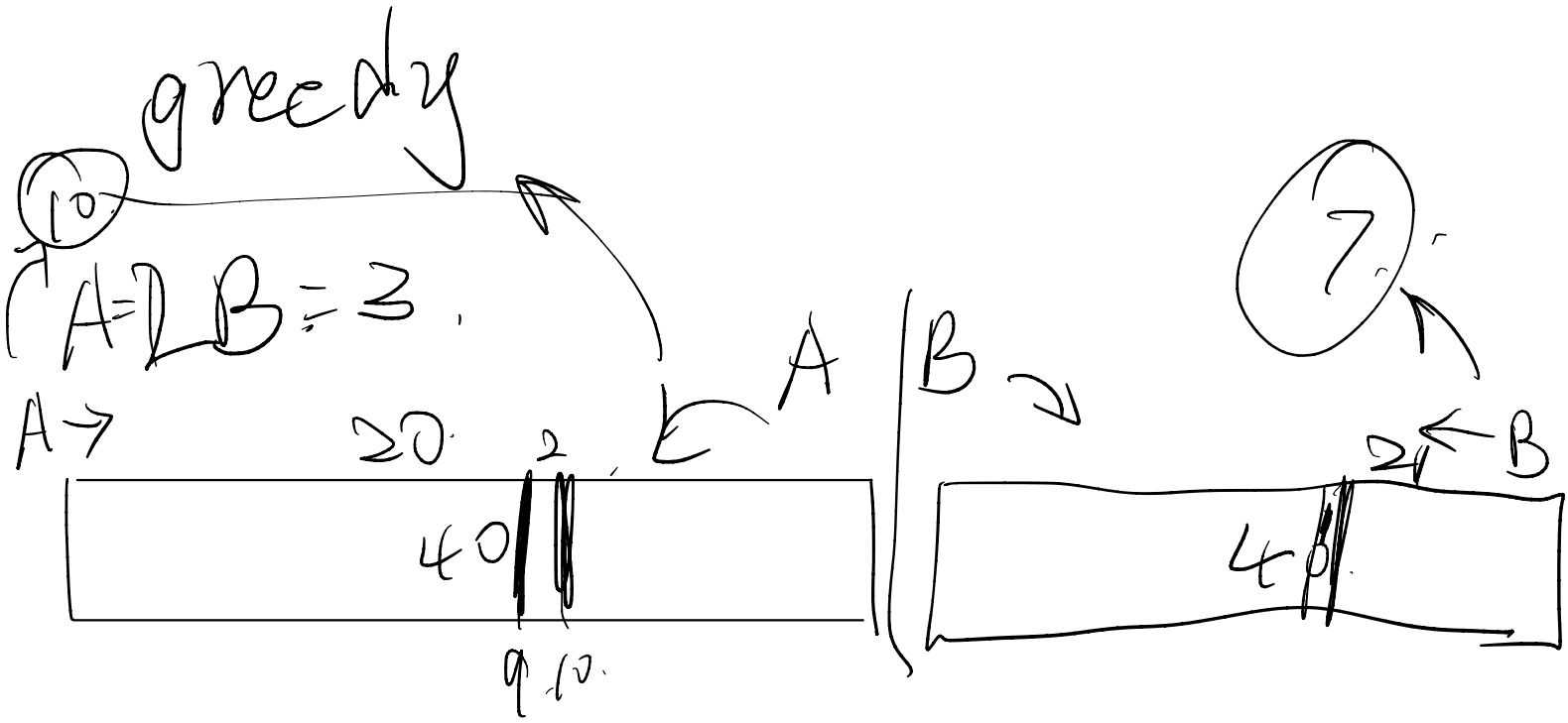


single-sided

q.insert()

q.delete()

Double-Ended queue



Map / Set.

dict (key : value) → Map.

HashMap <key, value> hm

String, Integer
Integer, Integer
Character Character.

= new HashMap<key, value>();

String → Integer.

Dave	0
L	0
B.	0

For ↓

Dave
200 ≥
L
B.

1. map.get(name)
2. map.put(name, a)
3. B.

12/15

① naive. record max matching

1. I/O., simple algo. O(4) == (even)

2. data structure. → Map.

① ubiquitous

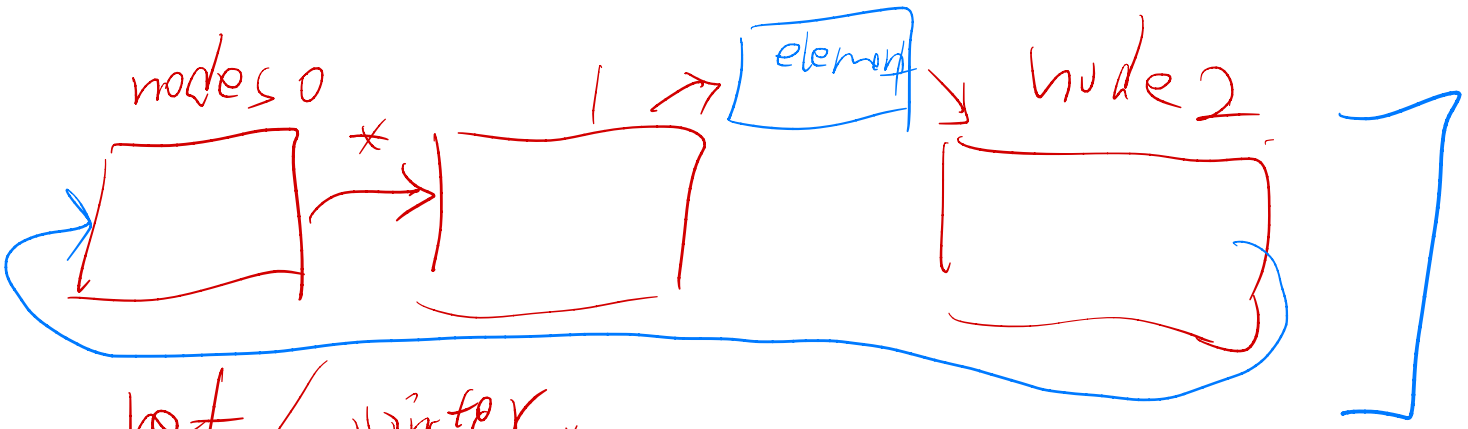
Time → mem.

3. array ←→ ArrayList.

★ func. / methods.

array Vs. ArrayList linked list.

List list = ArrayList<Integer>();



ref / pointer.

Java C/C++

add(1, Element)

array) ① primitive.

ArrayList.

[1, 2, 3, 4]

<1, 2, 5, 3, 4>

ArrayList

Collections

ArrayList<Integer> al = new AL<Integer>()

< 3 , 2 , 1 >

ascending.

Collections.sort(al);

< 1 , 2 , 3 >

bubble sort ←

merge sort. ←

<1000, 100, 8, 9, - - - - - - >

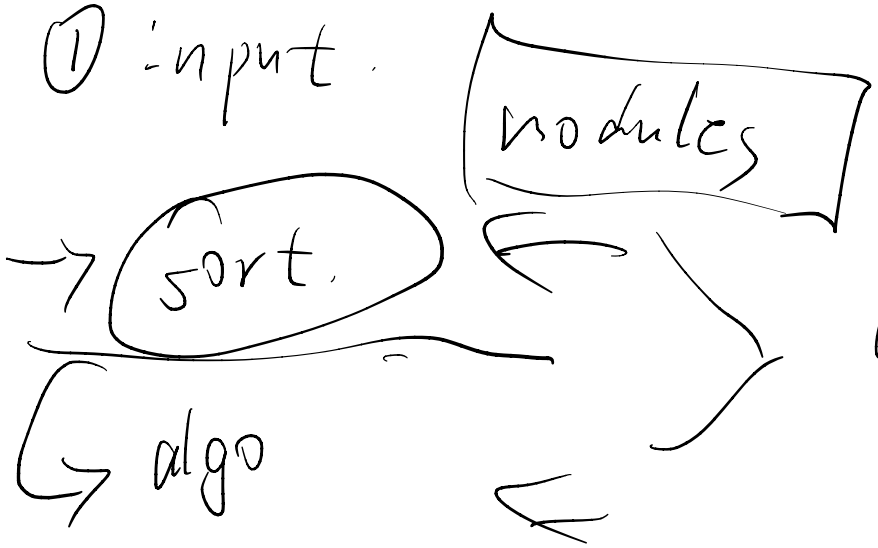
10^3.

Collections.sort(a1);

↓

O(nlogn). A*

① input.



algorithem seperation

function

(private) (int) func_name (int a, int b){

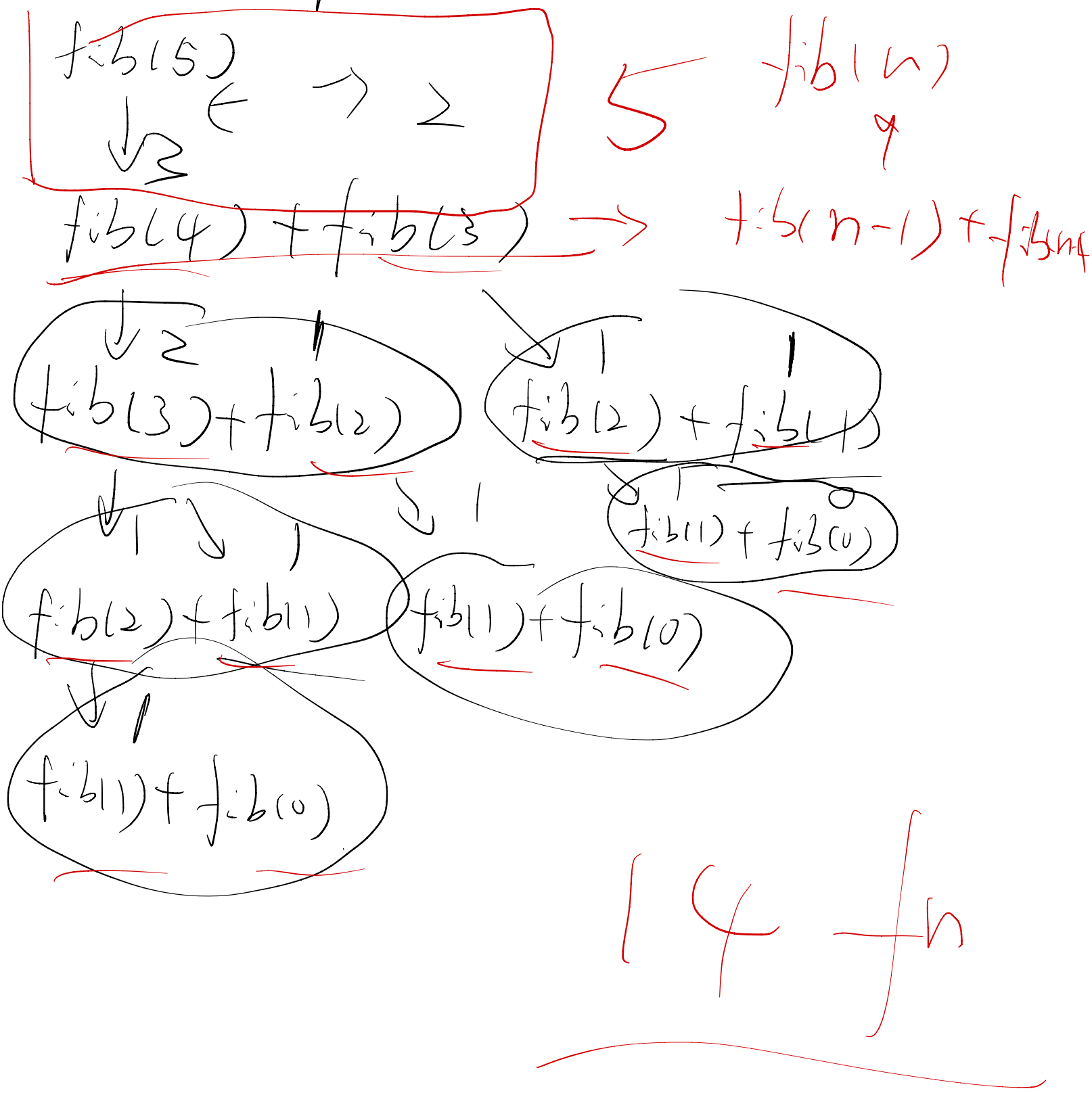
int ans1 ans2.

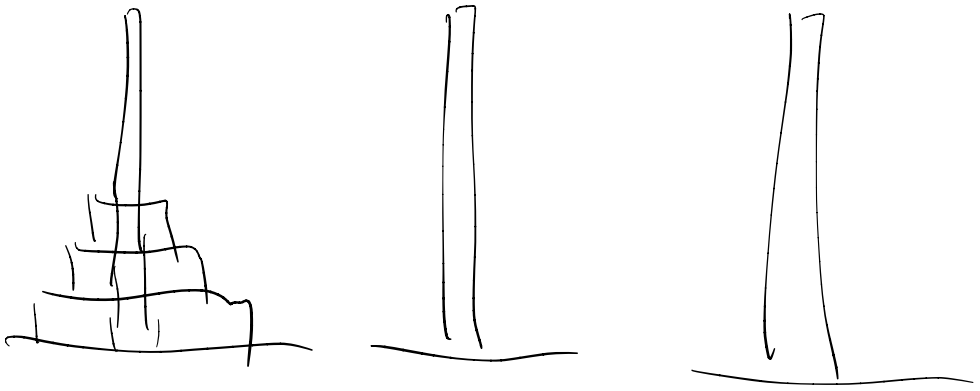
return ans1;

}

recursion

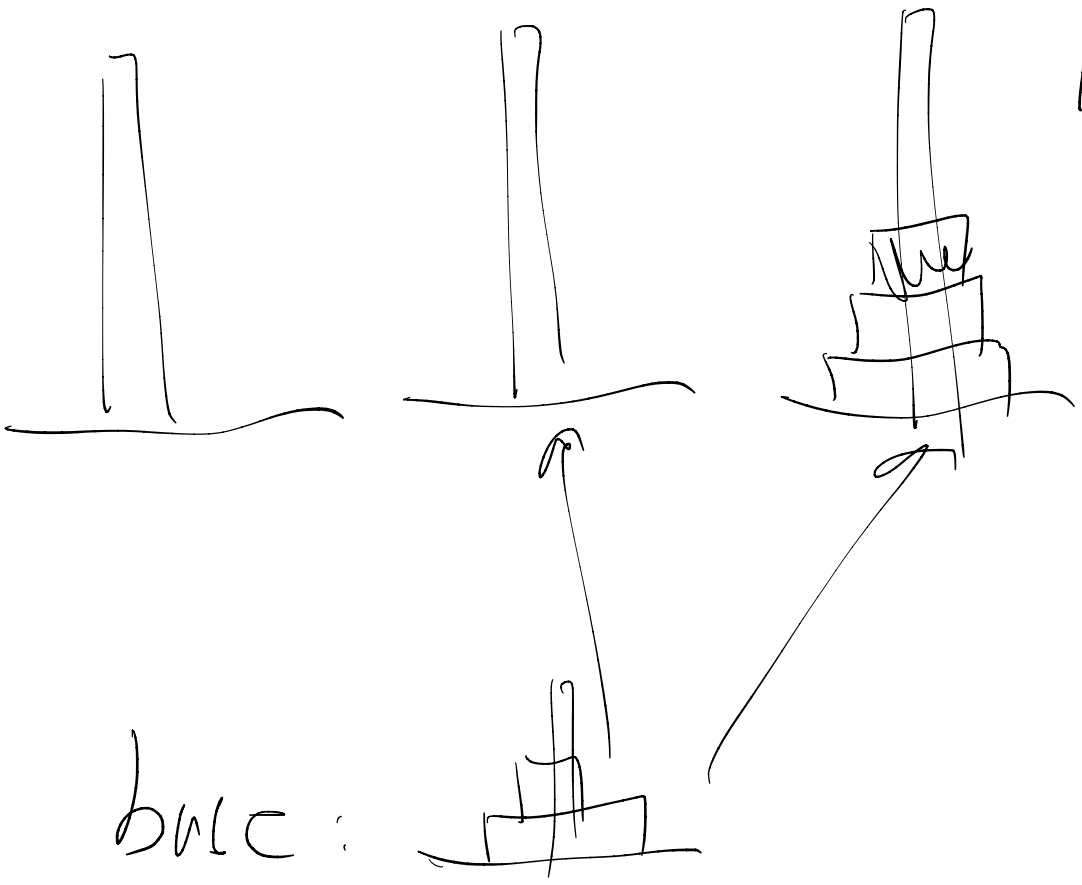
Time → space





rule:

always have small ones on large ones



base:

recf(n) (n-2) → 3rd
(n-1) → 2nd

func:

① separate.

2

L → ≤ G → ≥ least

L 3

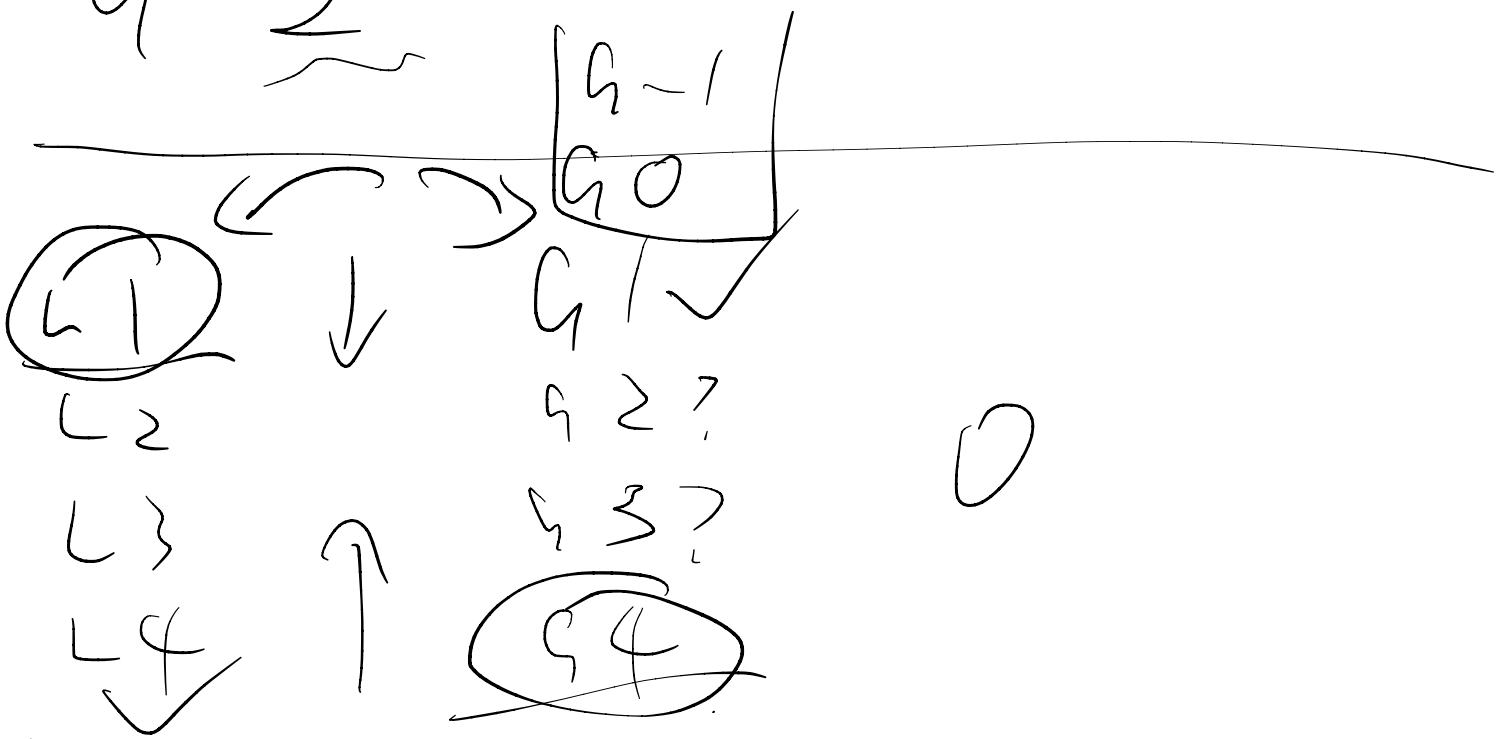
G 5

3

L 5

2 L 6 0

G 2



Data Struct ?

algo ?

ArrayList L, G

① read n,

2 → Nth: add L or G.

② Collections.sort( )

③ find out max_G

min_L

④ if(max_G < max_L)

if(min_G < min_L)