Binary Search :-

Earlier lecture we have seen linear search, its running time was $O(n)$.

elem : x
array : A
length of A : n

← { A is sorted in ascending order if $A[0] \leq A[1] \leq \dots \leq A[n-1]$.

binsearch ← $O(\log n)$.

$x = 2, n = 9$

A =		1	2	3	4	5	6	7	8	9
indices:	0	1	2	3	4	5	6	7	8	

$A[4] = 5$, $x = 2$.

$\frac{8+0}{2} = 4$

1 2 3 4 ~~5~~ 6 7 8 9

$\neq 2$

4 → 6 7 8 9

0 → 3

$x = 2$ ↑

$\{0, 1, 2, 3 \dots 8\} = [0, 9)$

If x present in A[0,9)
then return its index in A,
else return -1.

1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8

search x in A[0,9) - lo

hi

mid = $\frac{9+0}{2}$ = 4

A(4) ≠ 2.

Both the lower & higher indices we are reading in can change so lets change the question to:
is x ∈ A[lo, hi) ??

mid = (lo+hi)/2.

A[mid] > x ?

5 > 2 ?

{lo, ..., mid-1} = [lo, mid).

$A[mid] < x$?

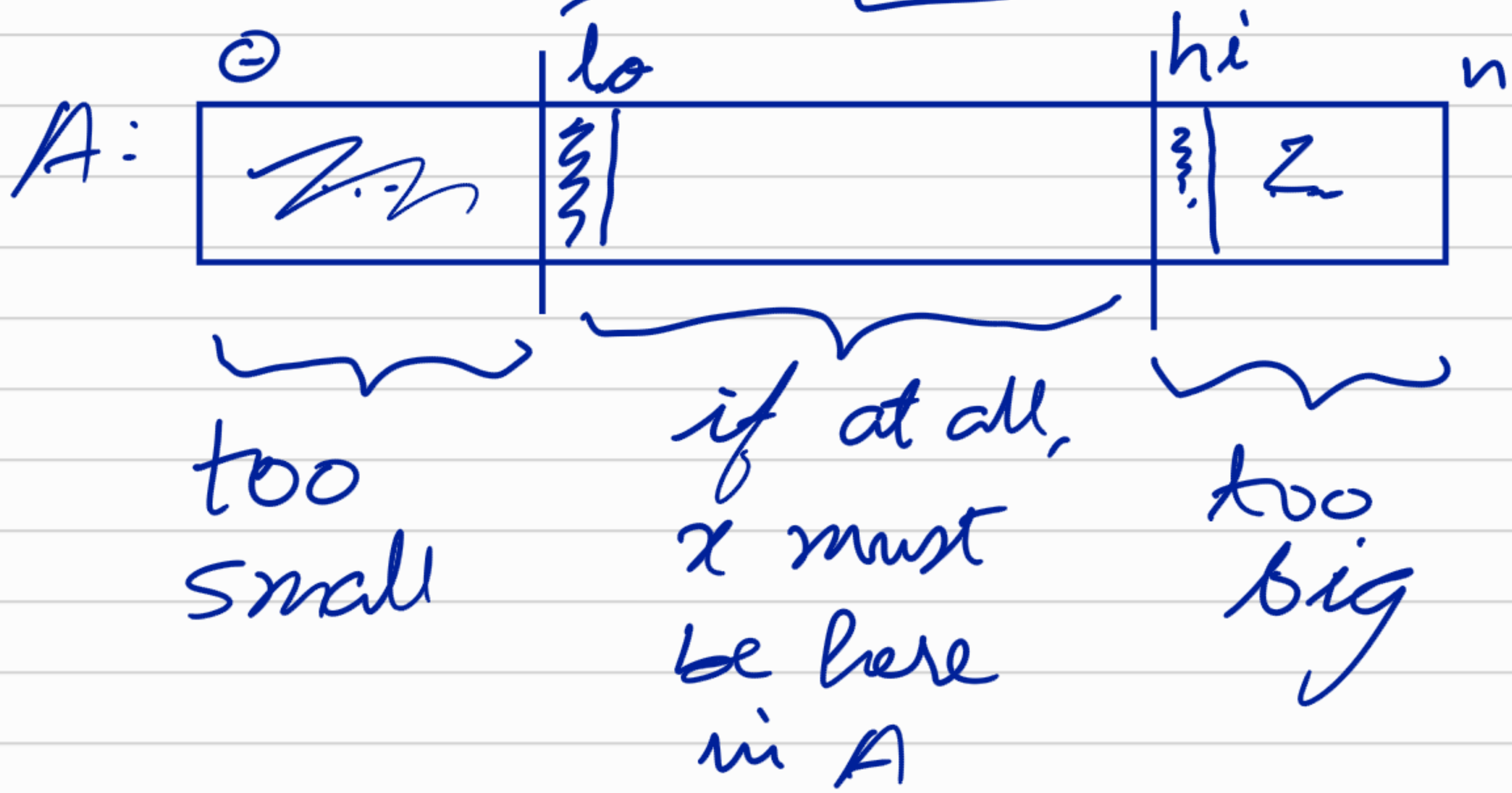
$\{mid+1, \ldots, hi-1\}$

"

$[mid+1, hi)$

At any arbitrary iteration of binary search :-

$[lo, hi)$



too small

if at all, x must be here in A

too big

$A[0, n) =$

$A[0, lo) \cup A[lo, hi) \cup A[hi, n)$

```
int binsearch(int x, int[] A, int n)
//@requires n == \length(A);
//@requires is_sorted(A, n);
/*@ensures       ( \result == -1                  && !is_in(x, A, 0, n) )
          ||    ( 0 <= \result && \result < n  && A[\result] == x );
@*/
{
    int lo = 0;
    int hi = n;
    //search for x in A[lo, hi)
    //recall that [lo, lo) is empty.
    while (lo < hi)
int binsearch(int x, int[] A, int n)
//@requires n == \length(A);
//@requires is_sorted(A, n);
/*@ensures       ( \result == -1                  && !is_in(x, A, 0, n) )
          ||    ( 0 <= \result && \result < n  && A[\result] == x );
@*/
{
    int lo = 0;
    int hi = n;
    while (lo < hi)
    /*@loop_invariant       0  <= lo
                    &&    lo <= hi
                    &&  hi <= n;   @*/
    //Q1: contract loop invariant:        A[0, lo) < x
    //Q2: contract loop invariant:        x < A[hi, n)
```

```
    {
    }
    return -1; //make sure that x is not in A[0,n)
}
```

Assuming that

```
LI1. /*@loop_invariant       0  <= lo
                     &&     lo <= hi
                     &&  hi <= n;    @*/
LI2.    //Q1: contract loop invariant: A[0, lo) < x
LI3.    //Q2: contract loop invariant:          x < A[hi, n)
```

We want to prove that whenever the program returns -1, it must be the correct answer.

Exit condition :- out of the program

assuming we break loop does return -1 correctly.

LG false, LI are true.

$lo \geq hi$; $lo \leq hi$.

$lo = hi$

CI2: $A[0, lo) < x$.

CI3: $A[hi, n) > x$.

$lo = hi$

$A[0, hi) < x$

$A[0, n) = A[0, hi) \cup A[hi, n)$

$A[0, hi) < x < A[hi, n)$

$A[hi-1] < x < A[hi]$

this implies that $x \notin A[0, n)$

hence we returned -1 correctly.

```
int binsearch(int x, int[] A, int n)
//@requires n == \length(A);
//@requires is_sorted(A, n);
/*@ensures      ( \result == -1                  && !is_in(x, A, 0, n) )
            ||  ( 0 <= \result && \result < n  && A[\result] == x );
@*/
{
    int lo = 0;
    int hi = n;
    while (lo < hi)
    /*@loop_invariant       0  <= lo
                        &&  lo <= hi
                        &&  hi <= n;     @*/
    //Q1: contract loop invariant: A[0, lo) < x
    //Q2: contract loop invariant:         x < A[hi, n)
    {
    }
    return -1;
}
```

Just note we haven't proved termination of the loop. So we may never reach return -1 statement.

n=4 a[i] b[i]

a → 0 1 1 1

b → 1 0 1 0

ans: 0 0 0 1

i'k. 1 0

0, 1, 2, 3

11 = 10

tmp = carry + a[i'] + b[i'].

carry	a[i]	b[i']
0	0	0
0	0	1

tmp

0 : 0 ← ans[i'].
0 ← carry

1 : 1 ← ans[i']
0 ← carry

2 (10) : 0 ← ans[i]
1 ← carry.

carry ← ans[i']

3 (11)

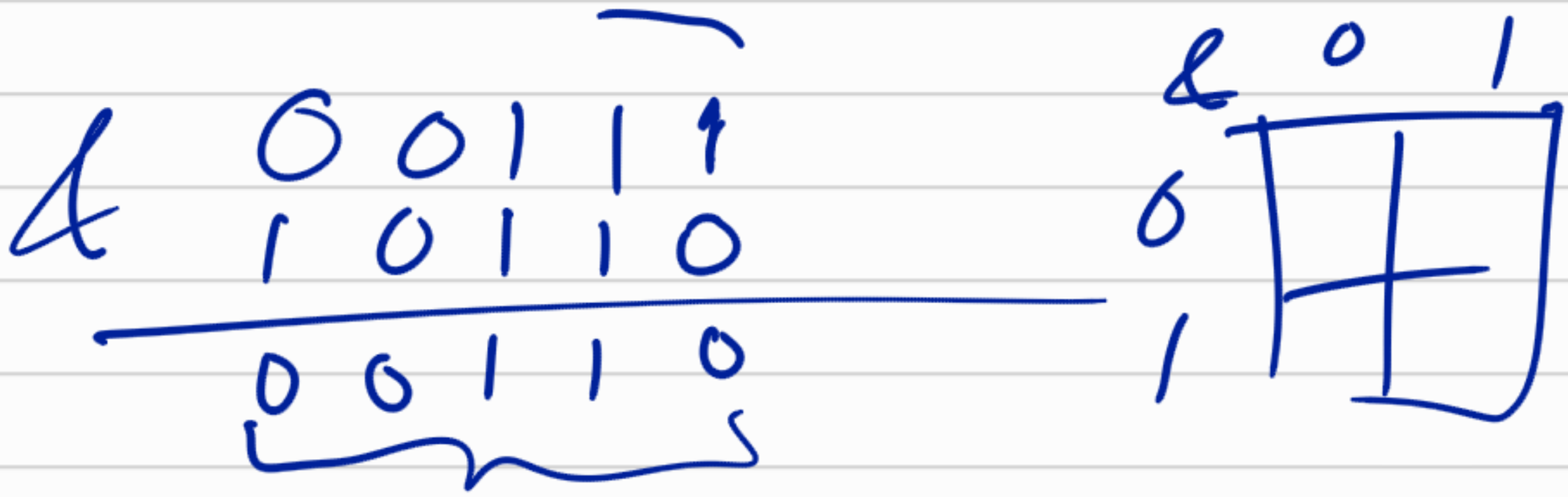
ith step :-

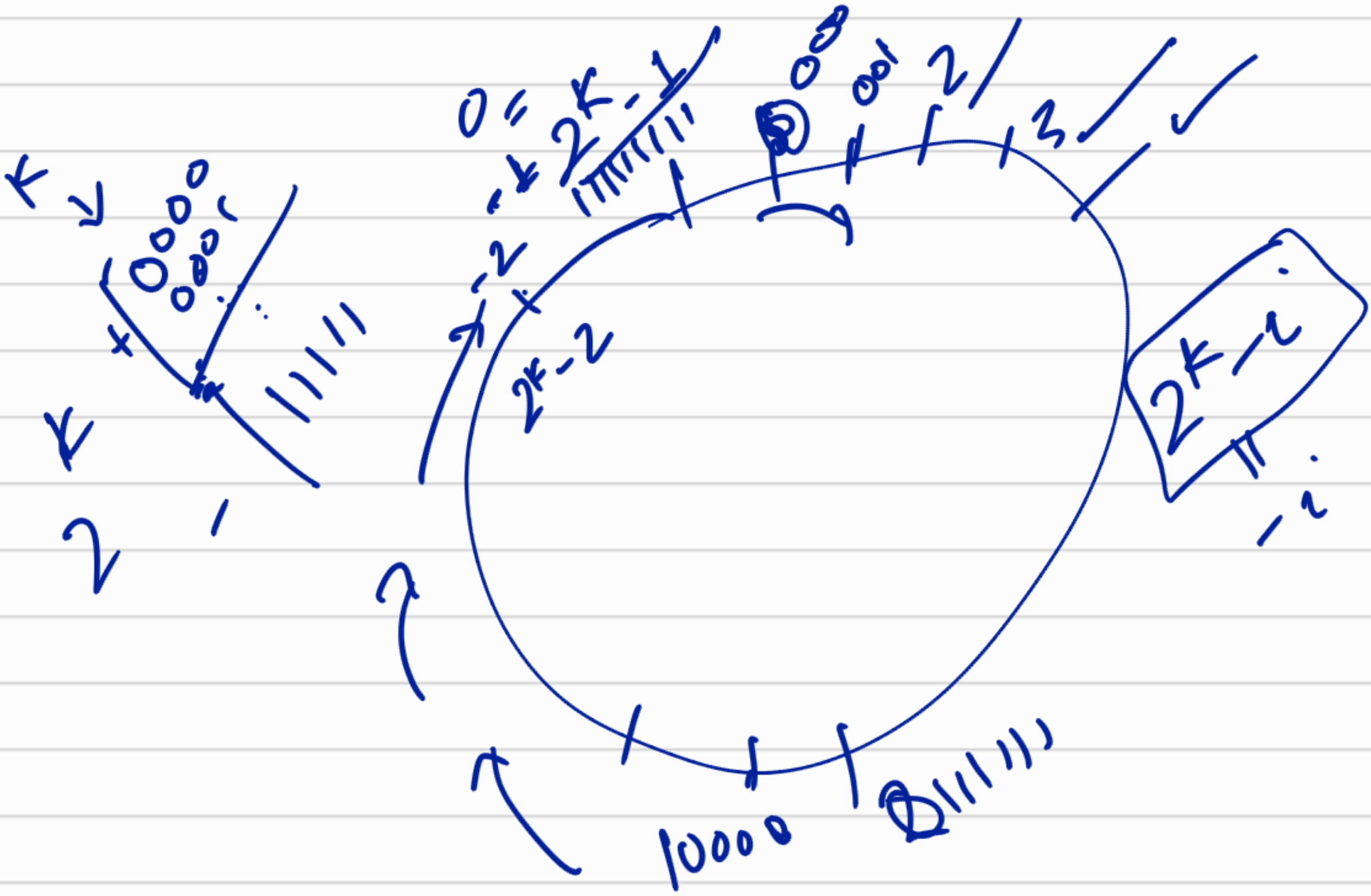
* tmp = carry + a[i] + b[i]
carry = tmp / 2;
ans[i] = tmp % 2;

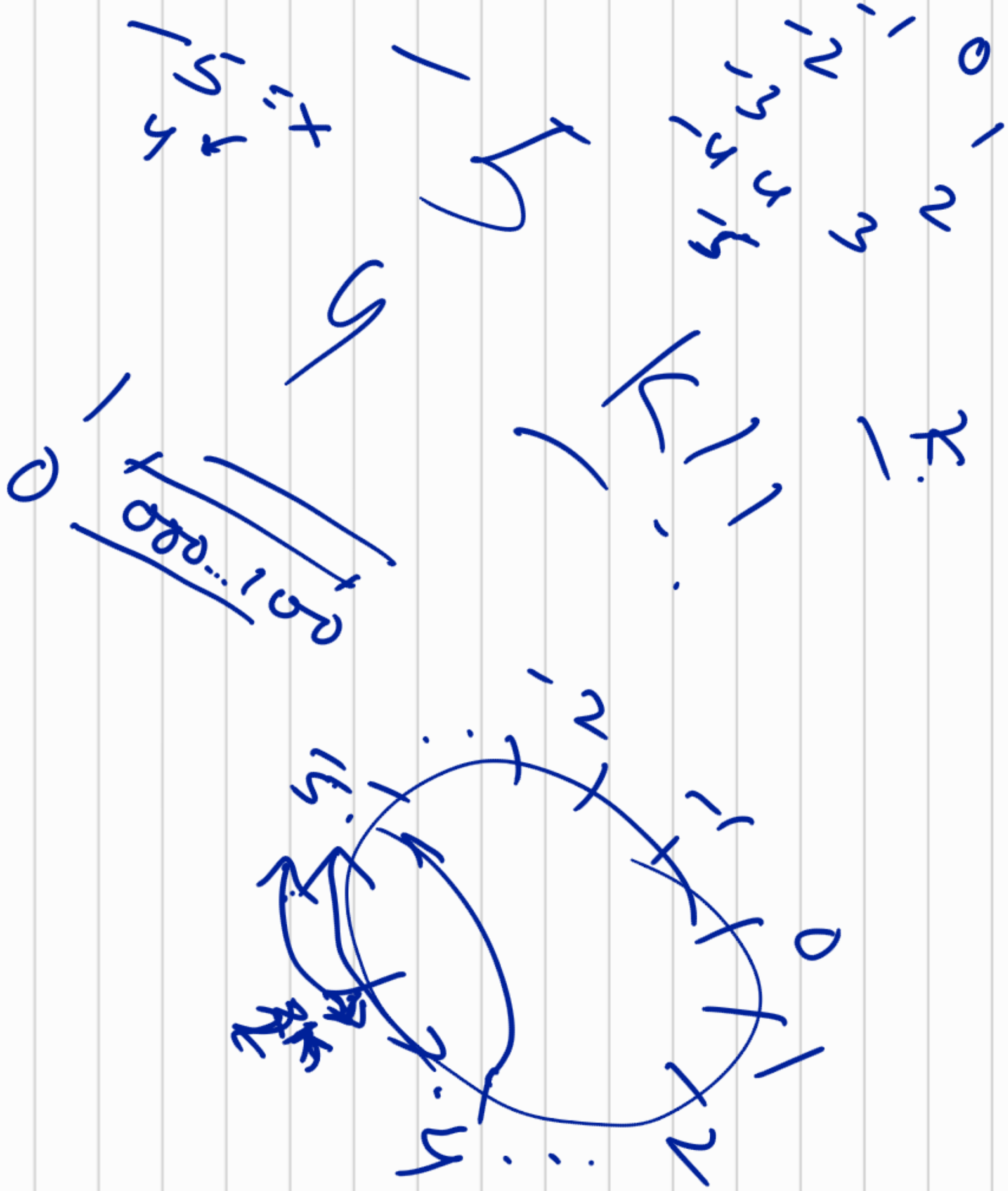
$c \in xy = 0$

i^{th} step :- $\in$.

*


0 0 1 1 1
1 0 1 1 0
0 0 1 1 0
& 0 1
0
1


K
00000
00001
11111
0 ~ 2K-1
00000
00001
2
3
2K-2
2K-2
2K-2
10000
11111



```
int binsearch(int x, int[] A, int n)
//@requires n == \length(A);
//@requires is_sorted(A, n);
/*@ensures      ( \result == -1                    && !is_in(x, A, 0, n) )
            ||  ( 0 <= \result && \result < n   && A[\result] == x );
@*/
{
    int lo = 0;
    int hi = n;
    while (lo < hi)
    /*@loop_invariant       0  <= lo
                        &&  lo <= hi
                        &&  hi <= n;    @*/
    //Q1: contract loop invariant: A[0, lo) < x
    //Q2: contract loop invariant:        x < A[hi, n)
    {
        int mid = (lo~~wer~~ + ~~upper~~) / 2; //@assert lo<=mid && mid<hi;
        if ( A[mid] == x ) {
            return mid;
        }
        if (A[mid] < x) {
            lo = mid+1;
        }
        else
        {//here we know A[mid] > x;
            hi = mid;
        }
    } //@assert lo==hi; // A[lo-1]<x<A[lo];
    return -1;
}
```

hi

A[

New things in the loop :-

```
while ( lo < hi ) {
    int mid = (lo+hi)/2 ;
    if ( A[mid] == x ) return mid;
    if ( A[mid] < x ) { lo = mid+1; }
    else { hi = mid; }
}
```

Proving the loop invariants :-

① LI 1 : $0 \le lo \le hi \le n$.

E.P. :
$0 \le lo$
$lo \le hi'$

INIT :-
1) $lo = 0$ (L8)
2) $hi = n$ (L9)
3) $lo \le hi$ ①+②.
④ $hi \le n$ -

```
1   int binsearch(int x, int[] A, int n)
2   //@requires n == \length(A);
3   //@requires is_sorted(A, n);
4   /*@ensures      ( \result == -1                    && !is_in(x, A, 0, n) )
5               ||  ( 0 <= \result && \result < n  && A[\result] == x );
6   @*/
7   {
8       int lo = 0;
9       int hi = n;
10      while (lo < hi)
11      /*@loop_invariant       0  <= lo
12                          &&  lo <= hi
13                          &&  hi <= n;    @*/
14      //Q1: contract loop invariant: A[0, lo) < x
15      //Q2: contract loop invariant:        x < A[hi, n)
16      {
17          int mid = (lo + hi) / 2;
18          //@assert lo <= mid && mid < hi;
19          if ( A[mid] == x ) {
20              return mid;
21          }
22          if (A[mid] < x) {
23              lo = mid+1;
24          }
25          else
26          {
27              //@assert A[mid] > x;
28              hi = mid;
29          }
30      }
31      //@assert lo == hi;
32      /* The correctness proof gave us the following*/
33      //@assert A[lo-1] < x && x < A[lo];
34      return -1;
35  }
```

} LI 1
← LI 2
← LI 3
← A1, L18
← L19
← L22
}

PRES: lo, hi before loop & lo', hi' after loop.
To show
if $0 \le lo \le hi' \le n$ then $0 \le lo' \le hi' \le n$

A[mid] == x
If LI9 is true, then we actually exit the loop, nothing to prove here as no lo' & hi' exist.

If A[mid] < x : (L22)

① $lo' = mid + 1$ by L23

② $hi' = hi$

T.P. $0 \le lo' \le hi' \le n$.

③ by LI1 $0 \le lo$

④ by assertion A1 (L18)

(HA) $lo \le mid$ ←

(HB) $mid < hi$

$0 \le lo'$

$0 \le lo$ by assumption

+ ① $lo' = mid + 1$

+ (HA) $lo \le mid$.

$lo < mid$

$lo < mid + 1$

$0 \le lo < lo'$

$lo' \leq hi'$

$mid < h_i' = h_i'$ —

$lo' = mid + 1$

$lo' - 1 \leq mid$ —

$lo' - 1 < hi'$

$lo' < hi' + 1$

$\leq$

Case 3)

If $A[mid] > x$:

← exercise → .

2). LI2: $A[0, lo) < x$

$lo = 0$

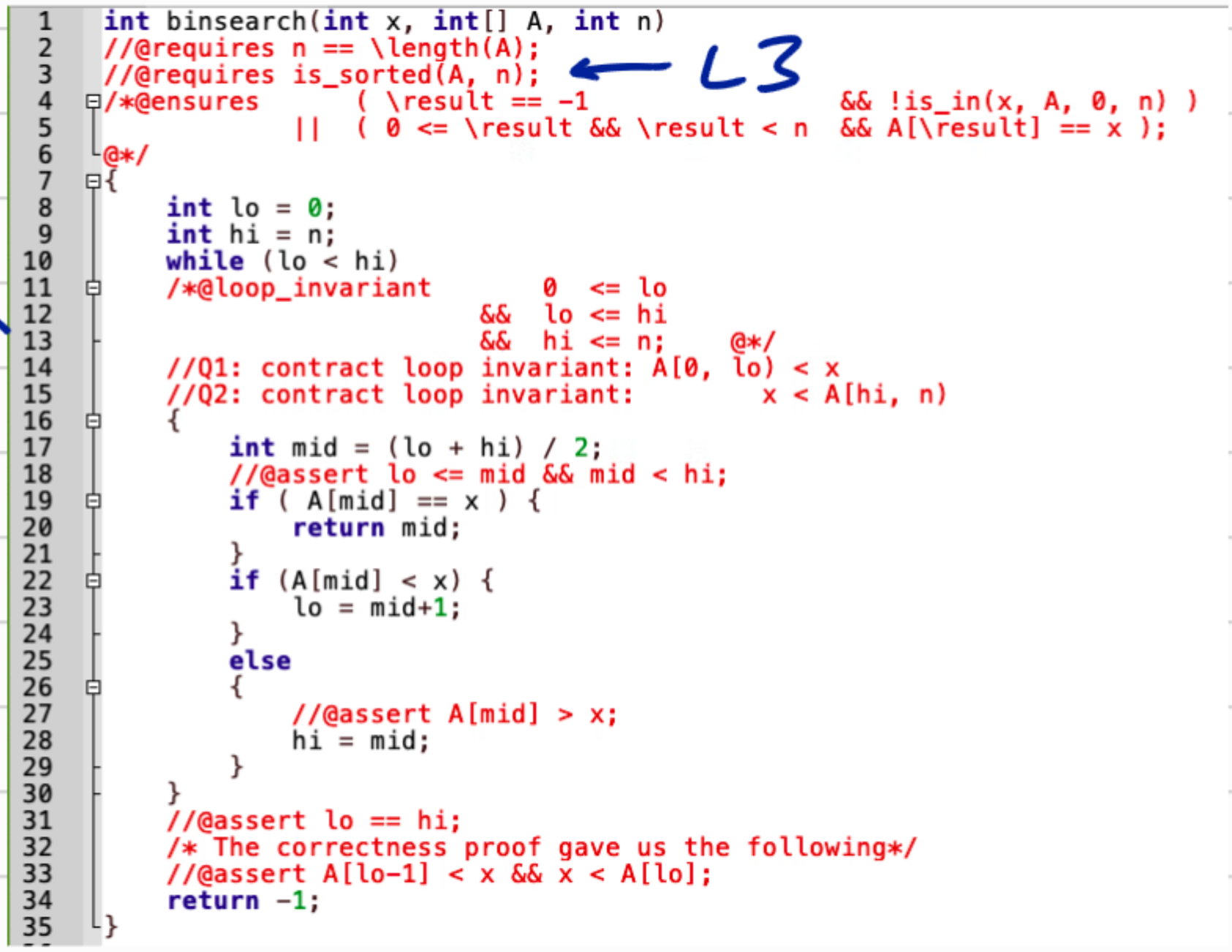
INIT:-

$A[0,0) < x$

$A[0,0) = \{\}$

for any element y

: $y \in A[0,0) = \{\}$



```
1  int binsearch(int x, int[] A, int n)
2  //@requires n == \length(A);
3  //@requires is_sorted(A, n);
4  /*@ensures      ( \result == -1                    && !is_in(x, A, 0, n) )
5              ||  ( 0 <= \result && \result < n  && A[\result] == x );
6  @*/
7  {
8      int lo = 0;
9      int hi = n;
10     while (lo < hi)
11     /*@loop_invariant        0  <= lo
12                         &&  lo <= hi
13                         &&  hi <= n;    @*/
14     //Q1: contract loop invariant: A[0, lo) < x
15     //Q2: contract loop invariant:        x < A[hi, n)
16     {
17         int mid = (lo + hi) / 2;
18         //@assert lo <= mid && mid < hi;
19         if ( A[mid] == x ) {
20             return mid;
21         }
22         if (A[mid] < x) {
23             lo = mid+1;
24         }
25         else
26         {
27             //@assert A[mid] > x;
28             hi = mid;
29         }
30     }
31     //@assert lo == hi;
32     /* The correctness proof gave us the following*/
33     //@assert A[lo-1] < x && x < A[lo];
34     return -1;
35 }
```

← L3

: $y < x$.

PRES: If $A[0,lo) < x$

then $A[0,lo') < x$

(1) when $A[mid] == x$, nothing to do.

Case ② when $A[mid] < x$ is true.

$lo = mid + 1$ — ②

[By L3 ; $A[0, n)$ is sorted.

$\Rightarrow A[0, mid) \leq A[mid] < x$

$A[0, mid+1) < x$

$A[0, lo) < x$ by ②.

~~Case~~ ③ $A[mid] > x$ is true

← Exercise →.

$[x, y)$

$= \{x, \ldots (y-1)\}$.

$[A[x, y) = \{A[x], A[x+1]$

$\ldots A[y-1]\}$.

LI1 ✓.

LI2 ✓

ex: LI3 is similar to LI2.

=

① Proved all the loop invariants.

② binsearch is correct when return -1 is executed.

③ loop is terminated ← not proved.

[lo, hi)

lo ≤ hi

length of [lo, hi) = abs(hi-lo).

= hi-lo

$$hi' - lo \sim \frac{1}{2}(hi - lo)\ \checkmark$$

$$\downarrow$$

$$\underline{lo} \qquad \underline{mid} \qquad \underline{hi'}$$

$$\uparrow$$

$lo \hookrightarrow hi'$

(II) $lo \leq hi'$ (*)

// Observations :-

① $hi - lo$ strictly decreases in each iteration.

② $hi - lo$ never falls below zero. by (*)

∴ ① + ② the seq. $hi - lo$ is finite.

```
1  int binsearch(int x, int[] A, int n)
2  //@requires n == \length(A);
3  //@requires is_sorted(A, n);
4  /*@ensures      ( \result == -1                  && !is_in(x, A, 0, n) )
5              ||  ( 0 <= \result && \result < n   && A[\result] == x );
6  @*/
7  {
8      int lo = 0;
9      int hi = n;
10     while (lo < hi)
11     /*@loop_invariant        0  <= lo
12                          &&  lo <= hi
13                          &&  hi <= n;    @*/
14     //Q1: contract loop invariant: A[0, lo) < x
15     //Q2: contract loop invariant:         x < A[hi, n)
16     {
17         int mid = (lo + hi) / 2;
18         //@assert lo <= mid && mid < hi;
19         if ( A[mid] == x ) {
20             return mid;
21         }
22         if (A[mid] < x) {
23             lo = mid+1;
24         }
25         else
26         {
27             //@assert A[mid] > x;
28             hi = mid;
29         }
30     }
31     //@assert lo == hi;
32     /* The correctness proof gave us the following*/
33     //@assert A[lo-1] < x && x < A[lo];
34     return -1;
35 }
```

L23

① if $0 < hi - lo$

then. $0 \leq \underbrace{hi' - lo'} < hi - lo$

Case 1:- if A[mid] == x

then we exit the loop

so nothing to prove.

Case 2: if A[mid] < x

L23

①. $hi' - lo' = hi - (mid+1)$

$= hi - mid - 1$

$< hi - mid$

A.L) $lo \leq mid < hi$

$hi - mid$

$\leq hi - lo$

$hi' - lo' < hi - lo$

$hi' - lo' = hi - (mid+1)$ ②

$> hi - (hi+1)$

$hi' - lo' > -1.$

$hi' - lo' \geq 0$ ①.

This tells us that the loop terminates.

We still need to show that $lo \leq mid < hi$

$$mid = \frac{lo + hi}{2}$$

T.P. $mid < hi$

fact: As we are in the loop $lo < hi$ (LG is true)

$\Rightarrow lo \leq hi - 1$

$$mid = \frac{lo + hi}{2} \leq \frac{hi - 1 + hi}{2}$$

$$\leq \frac{2hi - 1}{2}$$

$$\leq hi - 1$$

$\therefore mid < hi$ ——

T.P: $lo \leq mid$.

$mid = \frac{lo + hi}{2}$ & $lo \leq hi - 1$

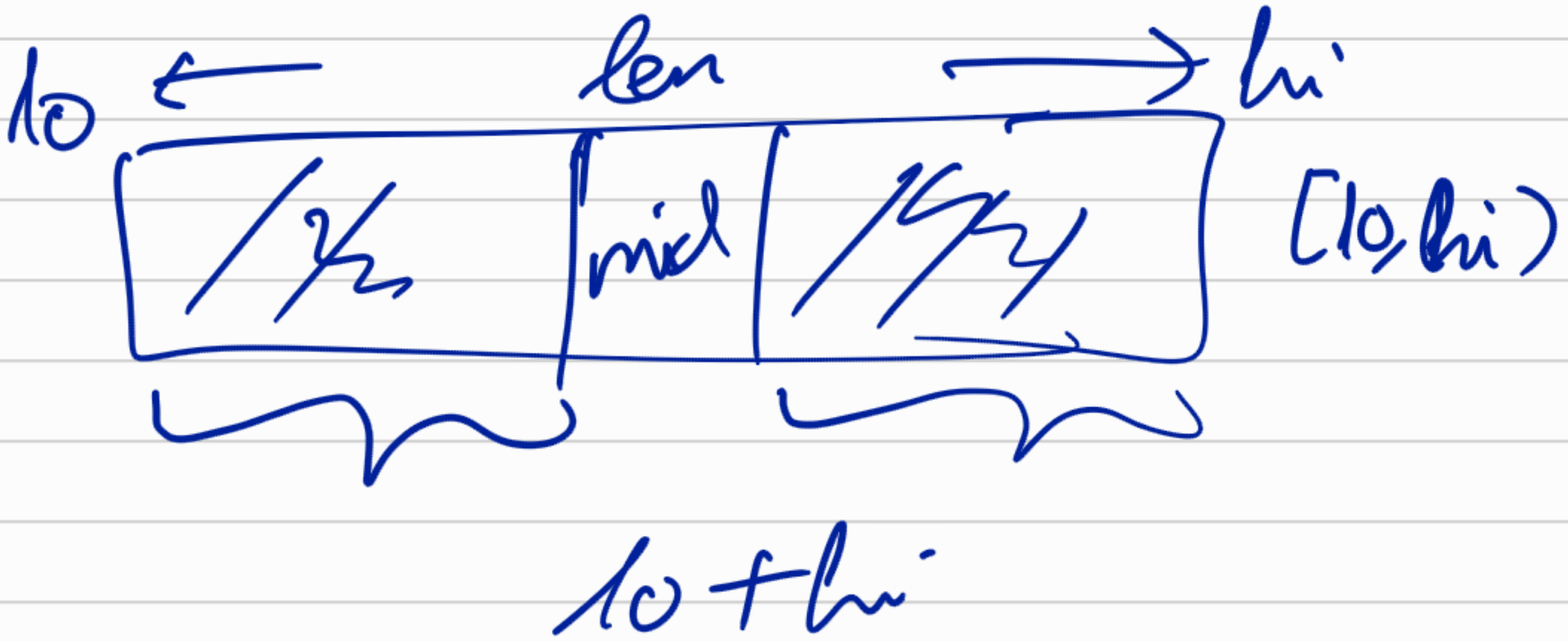
$lo + 1 \leq hi$

$mid = \frac{lo + hi}{2} \geqslant \frac{lo + lo + 1}{2}$

$\geqslant \frac{2lo + 1}{2}$

$mid \geqslant lo$

We proved binsearch to be safe and correct.



$(hi - lo) \leq \frac{hi - lo}{2}$ ∨.

If we start with an array of size 2^k then in the next step if we haven't found the element x then size to search in gets reduced to

$2^k \to 2^{k-1} \to 2^{k-2} \ldots$

$\log 2^k$ $\quad$ $\log 2^{k-1}$

k $\quad$ $k-1$

exercice 1

$\rightarrow 1 + 1 + \log n.$

exercice:

$O(\log n)$