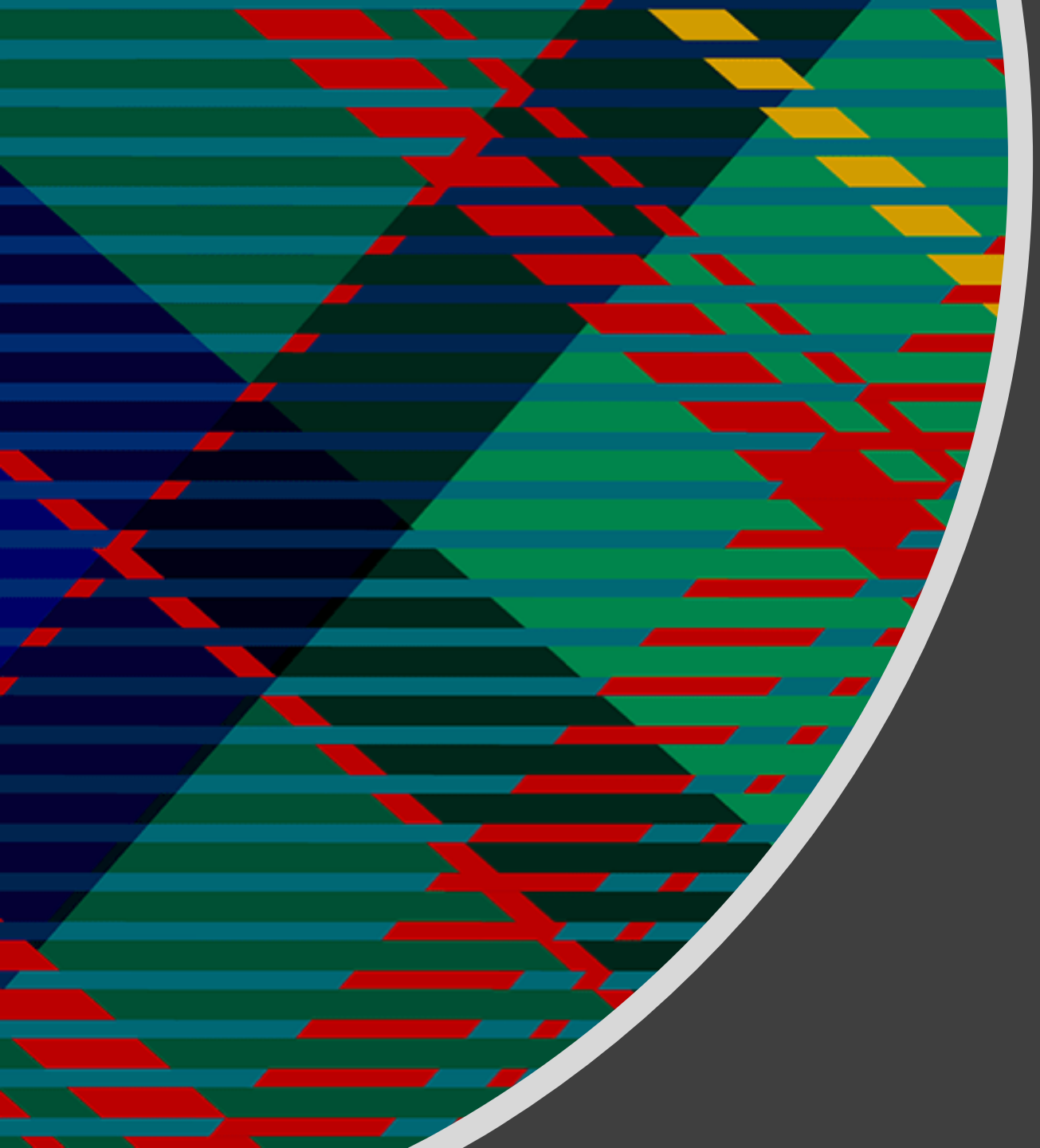


122
Binary Search

Last time

Cost

- Big ‘O’
- Complexity classes

Searching with order

Sorting

Question

What is the worst-case runtime complexity of selection sort?

(tightest, simplest bound)

A $O(\log n)$

B $O(n \log n)$

C $O(n)$

D $O(n^2)$

E Other

```
13 int find_min(int[] A, int lo, int hi)
14 {
15   int min_i = lo;
16   for (int i = lo+1; i < hi; i++) {
17     if (A[i] < A[min_i])
18       min_i = i;
19   }
20   return min_i;
21 }
22
23 void sort(int[] A, int lo, int hi)
24 {
25   for (int i = lo; i < hi; i++)
26   {
27     int min_index = find_min(A, i, hi);
28     swap(A, i, min_index);
29   }
30 }
```

Question

What is the worst-case runtime complexity of selection sort?

(tightest, simplest bound)

```
13 int find_min(int[] A, int lo, int hi)
14 {
15   int min_i = lo;
16   for (int i = lo+1; i < hi; i++) {
17     if (A[i] < A[min_i])
18       min_i = i;
19   }
20   return min_i;
21 }
22
23 void sort(int[] A, int lo, int hi)
24 {
25   for (int i = lo; i < hi; i++)
26   {
27     int min_index = find_min(A, i, hi);
28     swap(A, i, min_index);
29   }
30 }
```

Question

What is the *best-case* runtime complexity of selection sort?

(tightest, simplest bound)

A $O(\log n)$

B $O(n \log n)$

C $O(n)$

D $O(n^2)$

E Other

```
13 int find_min(int[] A, int lo, int hi)
14 {
15   int min_i = lo;
16   for (int i = lo+1; i < hi; i++) {
17     if (A[i] < A[min_i])
18       min_i = i;
19   }
20   return min_i;
21 }
22
23 void sort(int[] A, int lo, int hi)
24 {
25   for (int i = lo; i < hi; i++)
26   {
27     int min_index = find_min(A, i, hi);
28     swap(A, i, min_index);
29   }
30 }
```

Last time

Cost

- Big ‘O’
- Complexity classes

Searching with order

Sorting

Today

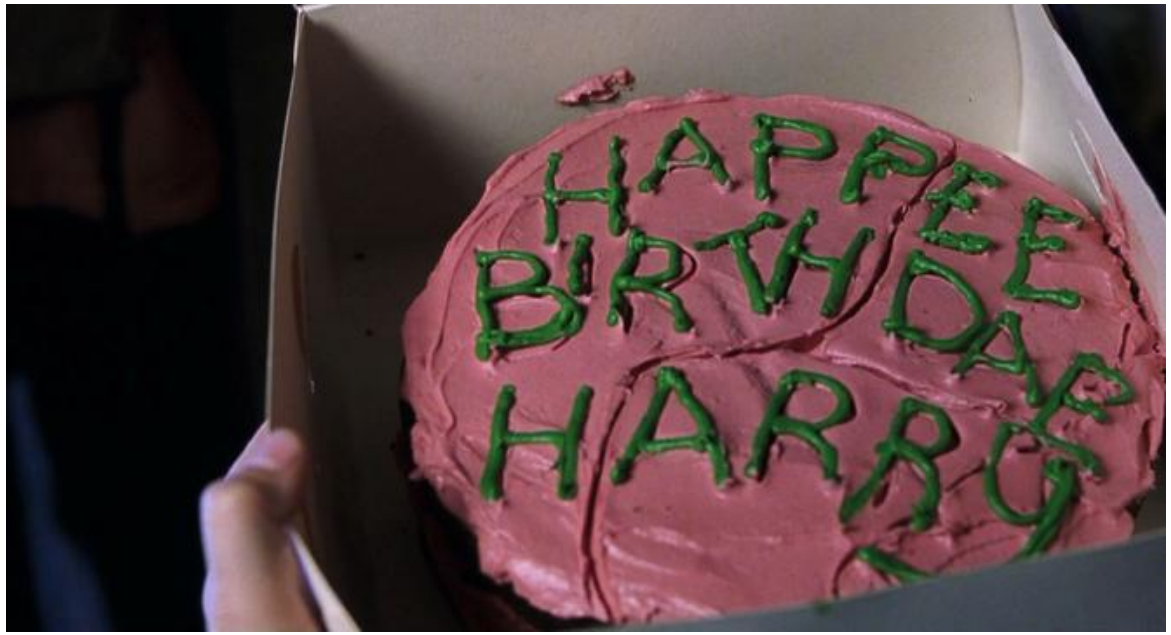
Faster search

- Binary search
- Divide and conquer
- Complexity

Next

Faster sort

Birthday 20 Questions



Guess the date of my birthday. I'll tell you if:

1) you're right
2) date should be earlier in the year
3) date should be later in the year

Question

I'm thinking of a number between 1 and 64.
After each guess, I'll tell you if you're correct or if my number is higher or lower.

What is the maximum number of guesses you'll need to play this game?

A 6

B 7

C 32

D 64

E Other

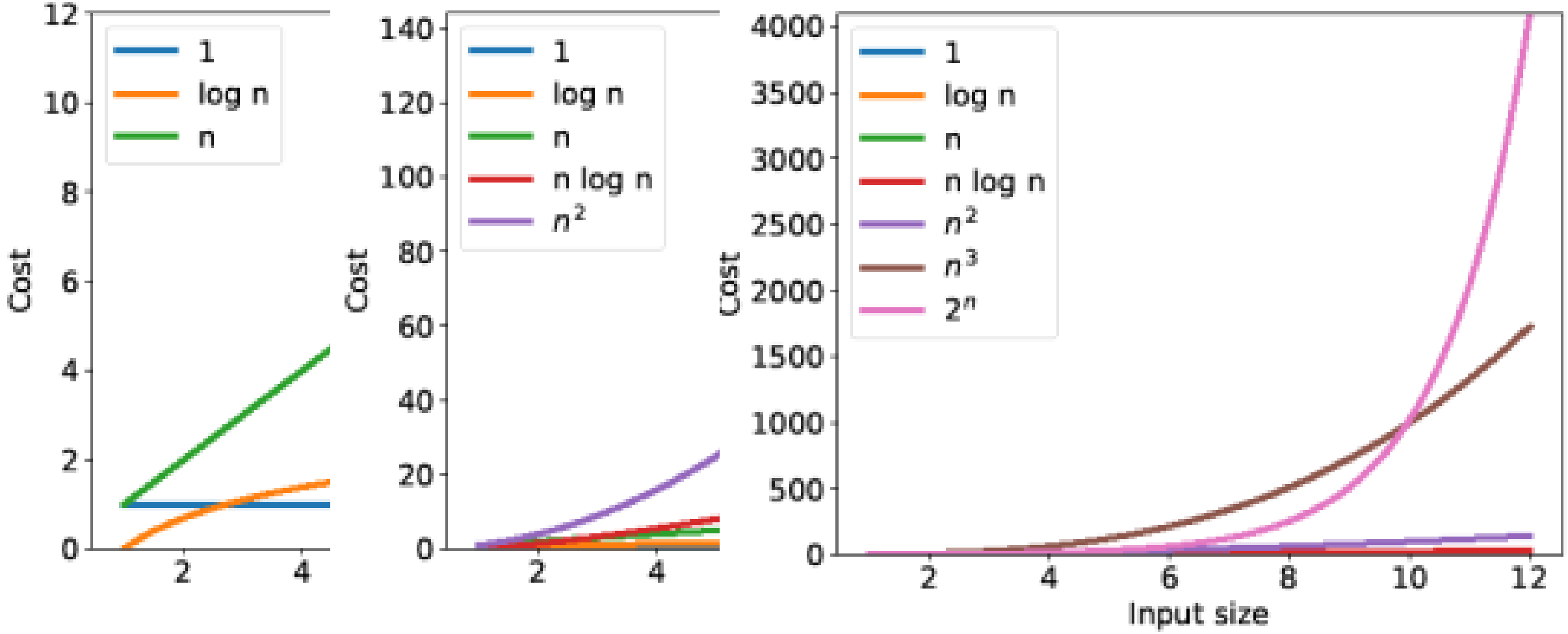
Question

I'm thinking of a number between 1 and N.
After each guess, I'll tell you if you're correct
or if my number is higher or lower.

What is the maximum number of guesses
you'll need to play this game?

N	10	100	1000	10K	100K	1M	10M	100M
$\log_2 N$	3.3	6.6	10.0	13.3	16.6	19.9	23.3	26.6
$\lfloor \log_2 N \rfloor + 1$	4	7	11	14	17	20	24	27

Complexity Classes



Searching sorted data

Binary search

Searching sorted data

Binary search

1) Check middle
    A) If found, return index
    B) Otherwise binary search lower/higher half

Searching sorted data

Seems simple

- https://reprog.wordpress.com/2010/04/19/are-you-one-of-the-10-percent
- http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html
- http://www.envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it

Contract for binary search

How should we modify the contract from sorted linear search?

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
```

Binary search

When should we stop and return -1?

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15
16   while
17   {
18
19   }
20   return -1;
21 }
```

Binary search

What do we know at an arbitrary iteration?

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15
16   while (lo < hi)
17   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
20   {
21   }
22   return -1;
23 }
```

Binary search

What do we know at an arbitrary iteration?

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15
16   while (lo < hi)
17   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
18   //@loop_invariant ... A[0, lo) < x ...
19   //@loop_invariant ... x < A[hi, n) ...
20   {
21   }
22   return -1;
23 }
```

Binary search

What do we know at an arbitrary iteration?

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15
16   while (lo < hi)
17   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
18   //@loop_invariant gt_seg(x, A, 0, lo);
19   //@loop_invariant lt_seg(x, A, hi, n);
20   {
21   }
22   return -1;
23 }
```

Binary search

Reasoning with an empty loop

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15
16   while (lo < hi)
17   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
18   //@loop_invariant gt_seg(x, A, 0, lo);
19   //@loop_invariant lt_seg(x, A, hi, n);
20   {
21   }
22   return -1;
23 }
```

Binary search

Reasoning with an empty loop

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15
16   while (lo < hi)
17   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
18   //@loop_invariant gt_seg(x, A, 0, lo);
19   //@loop_invariant lt_seg(x, A, hi, n);
20   {
21   }
22   return -1;
23 }
```

INIT:

- 0 <=lo by line 13
- lo <=hi by lines 13, 14, 7 and length(…) >= 0
- hi <= n by line 14
- gt_seg(x, A, 0, lo) by line 13 and x > empty seg
- lt_seg(x, A, hi, n) by line 14 and x < empty seg

PRES:

EXIT:

Binary search

Reasoning with an empty loop

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15
16   while (lo < hi)
17   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
18   //@loop_invariant gt_seg(x, A, 0, lo);
19   //@loop_invariant lt_seg(x, A, hi, n);
20   {
21   }
22   return -1;
23 }
```

INIT:

- 0 <=lo by line 13
- lo <=hi by lines 13, 14, 7 and length(…) >= 0
- hi <= n by line 14
- gt_seg(x, A, 0, lo) by line 13 and x > empty seg
- lt_seg(x, A, hi, n) by line 14 and x < empty seg

PRES:

EXIT:

- lo = hi by lines 16, 17
- gt_seg(x, A, 0, lo) by line 18 implies !is_in(x, A, 0, lo)
- lt_seg(x, A, lo, n) by line 19 implies !is_in(x, A, lo, n)
- !is_in(x, A, 0, n)

TERM??

Is this code safe?

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15   while (lo < hi)
16   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
17   //@loop_invariant gt_seg(x, A, 0, lo);
18   //@loop_invariant lt_seg(x, A, hi, n);
19   {
20     int mid = ...whatever...;
21
22     if (A[mid] == x)
23       return mid;
24     if (A[mid] < x) {
25       lo = mid+1;
26     }
27     else { //@assert A[mid] > x;
28       hi = mid;
29     }
30   }
31   return -1;
32 }
```

Is this code safe?

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15   while (lo < hi)
16   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
17   //@loop_invariant gt_seg(x, A, 0, lo);
18   //@loop_invariant lt_seg(x, A, hi, n);
19   {
20     int mid = ...whatever...;
21     //@assert lo <= mid && mid < hi;
22     if (A[mid] == x)
23       return mid;
24     if (A[mid] < x) {
25       lo = mid+1;
26     }
27     else { //@assert A[mid] > x;
28       hi = mid;
29     }
30   }
31   return -1;
32 }
```

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15   while (lo < hi)
16   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
17   //@loop_invariant gt_seg(x, A, 0, lo);
18   //@loop_invariant lt_seg(x, A, hi, n);
19   {
20     int mid = ...whatever...;
21     //@assert lo <= mid && mid < hi;
22     if (A[mid] == x)
23       return mid;
24     if (A[mid] < x) {
25       lo = mid+1;
26     }
27     else { //@assert A[mid] > x;
28       hi = mid;
29     }
30   }
31   return -1;
32 }
```

PRES:

LI1) `0 <= lo && lo <= hi && hi <= n`

A: If A[mid] == x

B: if A[mid] < x:

- lo' = mid+1 by line 25
- 0 < = lo by line 16
- lo <= mid by line 21
- 0 <= mid by math
- 0 <= mid+1??
- mid < hi by line 21
- mid < int_max()
- mid+1 <= int_max()
- 0 <= mid+1
  - so 0 <= lo'
  - lo' <= hi (just proved)
  - hi' <= n (hi unchanged)

C: if A[mid] > x:

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15   while (lo < hi)
16   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
17   //@loop_invariant gt_seg(x, A, 0, lo);
18   //@loop_invariant lt_seg(x, A, hi, n);
19   {
20     int mid = ...whatever...;
21     //@assert lo <= mid && mid < hi;
22     if (A[mid] == x)
23       return mid;
24     if (A[mid] < x) {
25       lo = mid+1;
26     }
27     else { //@assert A[mid] > x;
28       hi = mid;
29     }
30   }
31   return -1;
32 }
```

PRES:

LI1) `0 <= lo && lo <= hi && hi <= n`

A: If A[mid] == x

B: if A[mid] < x:

C: if A[mid] > x:

- 0 <= lo’ (lo unchanged)
- hi' = mid by line 28
- lo’ <= mid by line 21
- lo’ <= hi’
- mid <= n by lines 16, 21
- hi’ <= n by line 28

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15   while (lo < hi)
16   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
17   //@loop_invariant gt_seg(x, A, 0, lo);
18   //@loop_invariant lt_seg(x, A, hi, n);
19   {
20     int mid = ...whatever...;
21     //@assert lo <= mid && mid < hi;
22     if (A[mid] == x)
23       return mid;
24     if (A[mid] < x) {
25       lo = mid+1;
26     }
27     else { //@assert A[mid] > x;
28       hi = mid;
29     }
30   }
31   return -1;
32 }
```

PRES:

LI1) `0 <= lo && lo <= hi && hi <= n`

LI2) `gt_seg(x, A, 0, lo);`

```
 6  int search(int x, int[] A, int n)
 7  //@requires n == \length(A);
 8  //@requires is_sorted(A, 0, n);
 9  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
10      (0 <= \result && \result < n && A[\result] == x);
11    @*/
12 {
13   int lo = 0;
14   int hi = n;
15   while (lo < hi)
16   //@loop_invariant 0 <= lo && lo <= hi && hi <= n;
17   //@loop_invariant gt_seg(x, A, 0, lo);
18   //@loop_invariant lt_seg(x, A, hi, n);
19   {
20     int mid = ...whatever...;
21     //@assert lo <= mid && mid < hi;
22     if (A[mid] == x)
23       return mid;
24     if (A[mid] < x) {
25       lo = mid+1;
26     }
27     else { //@assert A[mid] > x;
28       hi = mid;
29     }
30   }
31   return -1;
32 }
```

PRES:

LI1) `0 <= lo && lo <= hi && hi <= n`

LI2) `gt_seg(x, A, 0, lo);`

LI3) `lt_seg(x, A, hi, n);`

Binary search

Computing the mid-point

```
 6  int search(int x, int[] A, int n)
 7 {
 8   int lo = 0;
 9   int hi = n;
10
11   while (lo < hi)
12   {
13     // int mid = lo;              // This is linear search!
14     // int mid = (hi + lo)/2;     // The bug Joshua Bloch found!
15     int mid = lo + (hi - lo)/2;
16     //@assert lo <= mid && mid < hi;
17
```