Checkpoint1

代码

```
int binsearch_final(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A, 0, n);
/*@ensures (-1 == \result && !is_in(x, A, 0, n))
|| ((0 <= \result && \result < n) && A[\result] == x);
@*/
{
    int lower = 0;
    int upper = n - 1;
    while (lower < upper)
    //@loop_invariant 0 <= lower && upper <= n - 1;
    //@loop_invariant (lower == 0 || A[lower - 1] < x);
    //@loop_invariant (upper == n - 1 || A[upper + 1] > x);
    {
        int mid = lower + (upper-lower)/2;
        //@assert lower <= mid && mid < upper;
        if (A[mid] == x) return mid;
        else if (A[mid] < x) lower = mid+1;
        else /*@assert(A[mid] > x);@*/
        upper = mid - 1;
    }
    return -1;
}
```

證明上面代碼中的三個循環不變量：

```
//@loop_invariant 0 <= lower && upper <= n - 1;         --1
//@loop_invariant (lower == 0 || A[lower - 1] < x);     --2
//@loop_invariant (upper == n - 1 || A[upper + 1] > x); --3 你好
```

證明：

1.在初始情況下，有lower=0 和 upper = n-1，而且函數有前置條件：0<=n&&n<=\length(A)，所以循環不變量都成立。
2.假設在某次進入循环前，3個循環不變量都成立，即有：

- 0<=lower && upper<= n-1
- lower==0 || A[lower-1]<x
- upper==n-1 || A[upper+1]>x
- lower < upper

分三種情況跳轉

- A[mid]==x：跳出循環，不改變，即循環不變量成立
- A[mid]< x：則lower=mid+1
    - 因爲lower <= mid < upper，則滿足循環不變量1
    - 因爲A[lower-1]=A[mid] < x，所以滿足循環不變量2
    - upper不改變，所以滿足循環不變量3
- A[mid]>x:則upper=mid-1
    - 因爲lower<=mid<upper，所以滿足循環不變量1
    - 因爲lower不變，所以滿足循環不變量2
    - 因爲A[upper+1]=A[mid]>x，所以滿足循環不變量3

證明循環會終止：

每次循環后，upper-lower必然減少，而且有下界1，故必然終止。

Checkpoint2

代码

```
int binsearch_final(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A, 0, n);
/*@ensures (-1 == \result && !is_in(x, A, 0, n))
|| ((0 <= \result && \result < n) && A[\result] == x);
@*/
{
    int lower = 0;
    int upper = n;
    while (lower < upper)
    //@loop_invariant 0 <= lower && lower <= upper && upper <= n;
    //@loop_invariant (lower == 0 || A[lower-1] < x);
    //@loop_invariant (upper == n || A[upper] > x);
    //@loop_invariant (is_in(x,A,0,n) == is_in(x,A,lower,upper));
    {
        int mid = lower + (upper-lower)/2;
        //@assert lower <= mid && mid < upper;
        if (A[mid] == x) return mid;
        else if (A[mid] < x) lower = mid+1;
        else /*@assert(A[mid] > x);@*/
        upper = mid;
    }
    return -1;
}
```

证明新的循环不变量

```
//@loop_invariant (is_in(x,A,0,n) == is_in(x,A,lower,upper));
```

初始

在初始情况下，有lower=0 和upper=n，所以is_in的输入都一致，那么结果肯定一致

循环中

假设某次进入循环前，满足循环不变量，那么分三种情况跳转

- A[mid]==x：则跳出循环，之后不会再进入循环，循环不变量成立
- A[mid] < x:则lower = mid+1。因为A数组有序，而且A[mid]< x，则代表[lower,mid]中没有x，那么只可能是[mid+1,upper)中有，则循环不变量满足
- A[mid] > x:则代表[mid,upper)中没有x，那么只可能是在[lower,mid)中有x，所以循环不变量满足。

验证循环不变量隐含后置条件

经过手动测试多组数据，并未发现后置条件报错。测试数据如下

- x=3,n=5,A[]={1,2,3,4,5}
- x=3,n=5,A[]={3,3,3,3,3}
- x=3,n=0

Checkpoint3

代码

```
int binsearch_final(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A, 0, n);
/*@ensures (-1 == \result && !is_in(x, A, 0, n))
|| ((0 <= \result && \result < n) && A[\result] == x && !is_in(x,A,0,\result));
@*/
{
    int lower = 0;
    int upper = n;
    while (lower < upper - 1)
    //@loop_invariant 0 <= lower && lower <= upper && upper <= n;
    //@loop_invariant (lower == 0 || A[lower] < x);
    //@loop_invariant (upper == n || A[upper] >= x);
    {
        int mid = lower + (upper-lower)/2;
        //@assert lower <= mid && mid < upper;

        if (A[mid] < x) lower = mid;
        else /*@assert(A[mid] >= x);@*/
        upper = mid;
        // printint(lower);printint(upper);
    }
    if(lower < n && A[lower] == x){
        return lower;
        //@assert lower == 0;
    }
    if(lower + 1 < n && A[lower+1]==x)return lower+1;
    return -1;
}
```

证明循环不变量

循环不变量

```
    //@loop_invariant 0 <= lower && lower <= upper && upper <= n;
    //@loop_invariant (lower == 0 || A[lower] < x);
    //@loop_invariant (upper == n || A[upper] >= x);
```

初始情况

初始情况下，lower=0,upper=n，自然满足循环不变量

循环时

假设某一循环时满足循环不变量，考虑对于下一个循环的影响。

- A[mid] < x：那么只有lower改变，下一个循环前的A[lower']=A[mid]< x，满足循环不变量2，又因为lower<=mid<=upper，自然满足循环不变量1
- A[mid] >= x:那么只有upper改变，下一个循环前的A[upper']=A[mid]>= x,满足循环不变量3，又因为lower<=mid<=upper，自然满足循环不变量1

证明后置条件

```
/*@ensures (-1 == \result && !is_in(x, A, 0, n))
|| ((0 <= \result && \result < n) && A[\result] == x && !is_in(x,A,0,\result));
@*/
```

因为满足前置条件和循环不变量，所以可得

- lower==0 || A[lower]<x
- upper==n || A[upper]>=x
- lower < upper-1
  所以考虑以下几种情况
- lower==0时，那么只可能是A[lower],A[lower+1]几种，判断是否越界即可
- A[lower]<x时，那么只可能是A[lower+1]，判断一下是否越界即可。

即我们实现的是找到包含小于x的最右边的数，然后判断下一位是否是x即可。

综上，当x存在时总能找到最左边的x，x不存在时返回-1，所以满足后置条件。

Checkpoint4

代码

```
int binsearch_before(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A, 0, n);
/*@ensures (-1 == \result && !is_in(x, A, 0, n))
|| ((0 <= \result && \result < n) && A[\result] == x);
@*/
{
    int lower = 0;
    int upper = n;
    while (lower < upper)
    //@loop_invariant 0 <= lower && lower <= upper && upper <= n;
    //@loop_invariant (lower == 0 || A[lower-1] < x);
    //@loop_invariant (upper == n || A[upper] > x);
    {
        int mid = lower + (upper-lower)/2;
        //@assert lower <= mid && mid < upper;
        if (A[mid] == x) return mid;
        else if (A[mid] < x) lower = mid+1;
        else /*@assert(A[mid] > x);@*/
        upper = mid;
    }
    return -1;
}
```

解答

约定中的

```
//@assert lower <= mid && mid < upper;
```

会告警。因为加法可能溢出为负数。如lower=2^{30-1},upper=2^{30}+1时，mid=(lower+upper)/2 = (-2^31)/2 = -2^30，不满足约定

Checkpoint5

代码

```
//...loop_invariant elided...
{
    lower = lower;
    upper = upper;
}
```

证明

- 初始时 lower = 0,upper = n 自然满足循环不变量
- 循环结束后，lower upper不变，自然也满足下个循环的不变量
  循环不变量确定的时不变的关系，无法保证应该改变的关系正确改变，所以无法正确实现二分查找。原因在于循环不会终止。