

So far in the course Important basic data structures Arrays, Lists, Sequences, Trees, Priority Queues, Heaps, Dictionaries, Hash Tables, and Binary Search Trees Search Trees Important algorithms Sorting (insertion, heap, PQ, merge, Quick, bucket, radix) Searching (Dictionary: binary search, hash table, BST) Selection (Quick, deterministic) not covered this time Design strategies Exhaustive Search, Divide-and-Conquer, Prune-and-Search, and randomization Solution to recurrences Amortized analysis (average behavior over a large number of times running the algorithm)

Reasoning About Loops ◆Make sure the loop has a goal and it is progressing toward that goal each time through the loop Make sure the loop invariant holds every time at the start and end of the loop body Make sure the loop terminates Make sure the loop is making progress toward the terminating condition Check boundary conditions ■ E.g., check size 0, 1, n

What is the loop invariant? An assertion that is necessarily true immediately before and immediately after each iteration of a loop Could be false part way through the loop, but must be re-established before the end of the loop body The invariant at termination of the loop should imply the goal of the loop has been achieved!!!!

4

3

Binary Search Algorithm (What's wrong) Algorithm BinarySearch(S, k): Input: key k and Sequence S storing n items, sorted by item.key() Output: the value associated with key k or NO\_SUCH\_KEY low ← 0  $high \leftarrow S.size() - 1$ while low < high do  $mid \leftarrow (low + high)/2$ if k = key(S.elemAtRank(mid)) then return value(S.elemAtRank(mid)) if k < key(S.elemAtRank(mid)) then high ← mid - 1 else  $low \leftarrow mid + 1$ return NO\_SUCH\_KEY

Error in binary search Does not handle the case when low equals high (boundary condition) ■ When the segment is size 1, the key may not be found because we do not enter the loop

5 6

```
Binary Search Algorithm
(Corrected)
  Algorithm BinarySearch(S, k):
  Input: key k and Sequence S storing n items, sorted by item key()
   Output: the value associated with key k or NO_SUCH_KEY
   low \leftarrow 0
   high ← S.size() - 1
   while low ≤ high do
      mid \leftarrow (low + high)/2
      if k = key(S.elemAtRank(mid)) then
        return value(S.elemAtRank(mid))
      if k < kev(S.elemAtRank(mid)) then
        \mathsf{high} \leftarrow \mathsf{mid} - 1
      else
        low \leftarrow mid + 1
   return NO_SUCH_KEY
```

7

Introducing Errors through Copy-Paste

- We wish to have only one key comparison during each iteration of the
- So we copy from above version then modify as described
  - Move the check for equality after the loop
  - Now we do not exit the loop early however we do half the key comparisons during each iteration

8

Binary Search Algorithm (what's wrong? Look at red) Algorithm BinarySearch(S, k): Input: An ordered Sequence S storing n items, accessed by keys() Output: An element of S with key k.  $low \leftarrow 0$ high ← S.size() - 1 while low ≤ high do  $mid \leftarrow (low + high)/2$ if k < kev(S.elemAtRank(mid)) then // one key comparison per iteration hiah  $\leftarrow$  mid - 1 else if k = key(S.elemAtRank(mid)) then // done once outside the loop now return value(S elemAtRank(mid)) return NO\_SUCH\_KEY

**Errors**  Does not handle a Sequence with 0 mid is not initialized since loop is not entered and, further, it cannot be initialized to handle an empty Sequence Does not handle a Sequence with 1 element that matches k The else eliminates mid when it hasn't yet been eliminated, so delete the + 1 from the else branch

10

9

Binary Search Algorithm (better, but what else is wrong?) Algorithm BinarySearch(S, k): Input: An ordered Sequence S storing n items, accessed by keys() Output: An element of S with key k. low ← 0 high ← S.size() - 1 while low ≤ high do  $mid \leftarrow (low + high)/2$ if k < key(S.elemAtRank(mid)) then // one key comparison per iteration high  $\leftarrow$  mid - 1 low ← mid // eliminate + 1 because mid has not been eliminated yet if S.size() > 0 /\ k = key(S.elemAtRank(mid)) then // handles empty S return value(S.elemAtRank(mid)) return NO\_SUCH\_KEY

Error: loop does not terminate Does not handle a Sequence with 1 item (or a segment with 1 item) when its key matches k ■ The loop does not terminate Modify the loop condition from ≤ to < so the loop terminates when high = low since low does not change when the key of the item equals k The rank mid may not contain the item with the key after fixing the loop's terminating condition ■ Either low or high will contain the key if it is in the Sequence Fixing this eliminates the need to initialize mid before the loop since mid will only used inside the loop now

11 12

```
Binary Search Algorithm
(red shows corrections)
Algorithm BinarySearch(S, k):
 Input: An ordered Sequence S storing n items, accessed by keys()
 Output: An element of S with key k.
 low \leftarrow 0
 high ← S.size() - 1
 while low < high do
                                 // needs to be < to terminate
    mid \leftarrow (low + high + 1)/2
                                // needs to be the ceiling to terminate
    if k < key(S.elemAtRank(mid)) then // one key comparison per iteration
      high \leftarrow mid - 1
     else
      low ← mid // + 1 because mid has not been eliminated yet
 if S.size() > 0 \( \Lambda \) k = key(S.elemAtRank(high)) then // handles empty S
   return value(S.elemAtRank(high)) // high or low contain matching key
    return NO_SUCH_KEY
```

Binary Search Algorithm (change < to < in the loop) Algorithm BinarySearch(S, k): Input: An ordered Sequence S storing n items, accessed by keys() Output: An element of S with key k.  $low \leftarrow 0$ high ← S.size() - 1 while low < high do // needs to be < to terminate  $mid \leftarrow (low + high + 1)/2$ // needs to be the ceiling to terminate if k ≤ key(S.elemAtRank(mid)) then // change to ≤ instead of < high ← mid // changed due to change of condition else low ← mid + 1 // changed due to change of condition if S.size() > 0 /\ k = key(S.elemAtRank(high)) then // handles empty S return value(S.elemAtRank(high)) // high or low contain matching key return NO\_SUCH\_KEY

13 14

```
Errors

◆ The loop does not always terminate

■ mid needs to be the floor of the expression otherwise mid and high do not/cannot change which causes non-termination
```

Binary Search Algorithm (loop now terminates) Algorithm BinarySearch(S, k): Input: An ordered Sequence S storing n items, accessed by keys() Output: An element of S with key k.  $low \leftarrow 0$ high ← S.size() - 1 while low < high do // needs to be < to terminate // needs to be the floor to terminate  $mid \leftarrow (low + high)/2$ if k ≤ key(S.elemAtRank(mid)) then // changed to ≤ instead of < high ← mid // removed – 1 (since mid is not eliminated) else low ← mid + 1 // changed if S.size() > 0  $\land$  k = key(S.elemAtRank(low)) then // handles empty S return value(S.elemAtRank(low)) // high or low contain matching key else return NO\_SUCH\_KEY

15 16

```
Why a third version?

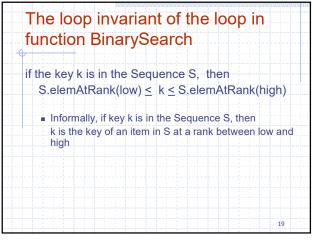
Depends on the purpose
The third version is an improvement in the binary search used by the Lookup Table
```

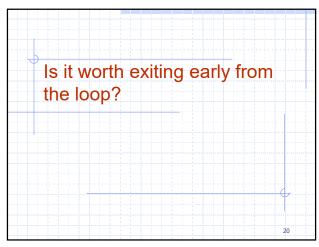
Errors (none)

Handles a Sequence with 0 elements
Handles a Sequence with 1 element that matches the key k

We do not want the ceiling((high+low)/2) this time
The loop terminates
mid is initialized correctly with the floor of the expression (does not add 1)
Handles a Sequence with 2 elements (or a segment with 2 elements) with one matching the key k
Two cases: first and second element
Finds the key when it is in the Sequence by using rank low although could have left it as high

17 18





19 20

Binary Search Algorithm (Two comparisons per iteration) Algorithm BinarySearch(S, k): Input: An ordered Sequence S storing n items, accessed by keys() Output: An element of S with key k. low ← 0 high ← S.size() - 1 while low ≤ high do  $mid \leftarrow (low + high)/2$ if k = key(S.elemAtRank(mid)) then {exit early from the loop} return value(S.elemAtRank(mid)) else if k < key(S.elemAtRank(mid)) then high ← mid - 1 else  $low \leftarrow mid + 1$ return NO\_SUCH\_KEY

Binary Search Algorithm

(One comparison per iteration)

Algorithm BinarySearch( S, k):
Input: An ordered Sequence S storing n items, sorted by keys()

Output: An item of S with key k and rank between low & high.

low ← 0
high ← S.size() - 1
while low < high do
mid ← (low + high + 1)/2
if k < key(S.elemAtRank(mid)) then {always does log n comparisons}
high ← mid - 1
else
low ← mid I/ + 1 does not yet eliminate mid
if S.size() > 0 Λ k = key(S.elemAtRank(low)) then
return value(S.elemAtRank(low))
else
return NO\_SUCH\_KEY

21 22

Homework

Both algorithms make O(log n) key comparisons

Which algorithm makes fewer actual key comparisons when the key is not in S?

Which makes fewer comparisons, on average, when the key is in S, assuming all keys are equally likely to be searched?

What's Wrong with this

In Place Version of Partition

Algorithm inPlacePartition(S, lo, hi)
Input Sequence S and ranks lo and hi,  $0 \le lo \le hi \le S$ .size()
Output the pivot is now stored at its sorted rank  $p \leftarrow a$  random integer between lo and hi
SawapElements(S.atRank( lo), S.atRank(p))  $j \leftarrow lo + 1$   $k \leftarrow hi$ while  $j \le k$  do
while  $k > j \land S$ .elemAtRank(k)  $\ge pivot$  do  $k \leftarrow k - 1$ while  $j \le k \land S$ .elemAtRank(j)  $\le pivot$  do  $j \leftarrow j + 1$ if  $j \le k$  then
SswapElements(S.atRank(j), S.atRank(k)) {move pivot to sorted rank}
return k

23 24

