Binary Search

Bruce Nan

Problem: Search

- We are given a list of records.
- Each record has an associated key.
- Give efficient algorithm for searching for a record containing a particular key.
- Efficiency is quantified in terms of average time analysis (number of comparisons) to retrieve an item.

Serial Search

- Step through array of records, one at a time.
- Look for record with matching key.
- Search stops when
  - record with matching key is found
  - or when search has examined all records without success.

Pseudocode for Serial Search

```
// Search for a desired item in the n array elements
// starting at a[first].
// Returns pointer to desired record if found.
// Otherwise, return NULL
…
for(i = first; i < n; ++i )
        if(a[first+i] is desired item)
                return &a[first+i];

// if we drop through loop, then desired item was not found
return NULL;
```

Serial Search Analysis

- What are the worst and average case running times for serial search?
- We must determine the O-notation for the number of operations required in search.
- Number of operations depends on *n*, the number of entries in the list.

Worst Case Time for Serial Search

- For an array of n elements, the worst case time for serial search requires n array accesses: $O(n)$.
- Consider cases where we must loop over all n records:
  - desired record appears in the last position of the array
  - desired record does not appear in the array at all

Average Case for Serial Search

Assumptions:

1. All keys are equally likely in a search
2. We always search for a key that is in the array

Example:

- We have an array of 10 records.
- If search for the first record, then it requires 1 array access; if the second, then 2 array accesses. *etc.*

The average of all these searches is:

(1+2+3+4+5+6+7+8+9+10)/10 = 5.5

Average Case Time for Serial Search

Generalize for array size *n.*

Expression for average-case running time:

$(1+2+\ldots+n)/n = n(n+1)/2n = (n+1)/2$

Therefore, average case time complexity for serial search is $O(n)$.

Binary Search

- Perhaps we can do better than O(n) in the average case?
- Assume that we are give an array of records that is sorted. For instance:
  - an array of records with integer keys sorted from smallest to largest (e.g., ID numbers), or
  - an array of records with string keys sorted in alphabetical order (e.g., names).

Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\leq$ `value` $\leq$ `a[hi]`.

Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo														↑ hi

Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\leq$ `value` $\leq$ `a[hi]`.

Ex. Binary search for 33.

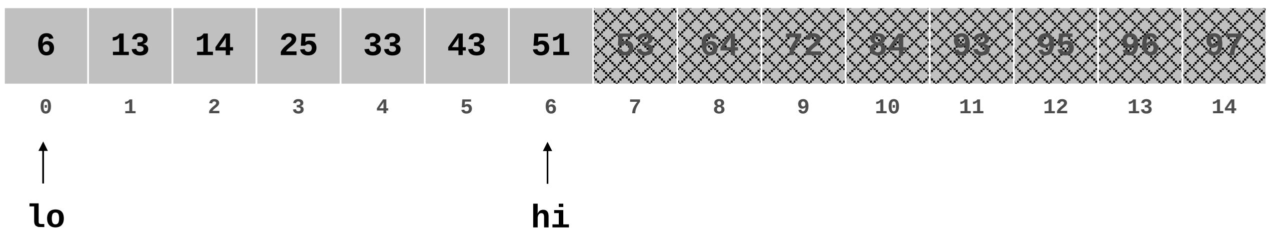
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo							↑ mid							↑ hi

Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\le$ `value` $\le$ `a[hi]`.

Ex. Binary search for 33.

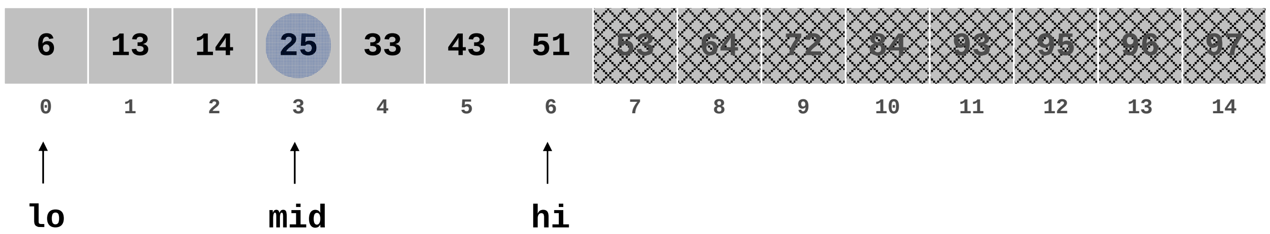


Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\leq$ `value` $\leq$ `a[hi]`.

Ex. Binary search for 33.



Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\leq$ `value` $\leq$ `a[hi]`.

Ex. Binary search for 33.

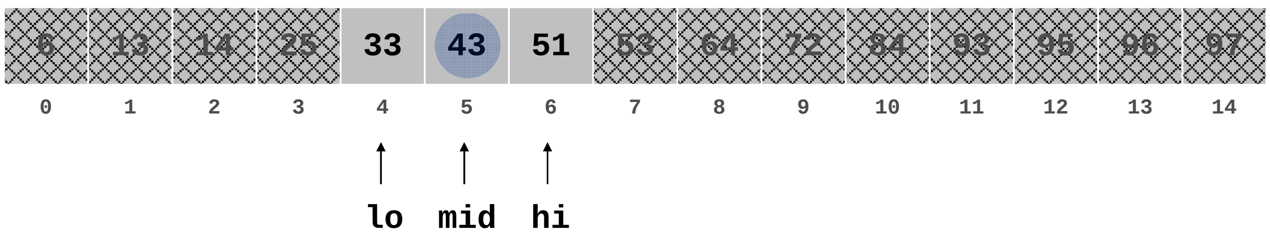
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
				↑ lo		↑ hi								

Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\leq$ `value` $\leq$ `a[hi]`.

Ex. Binary search for 33.

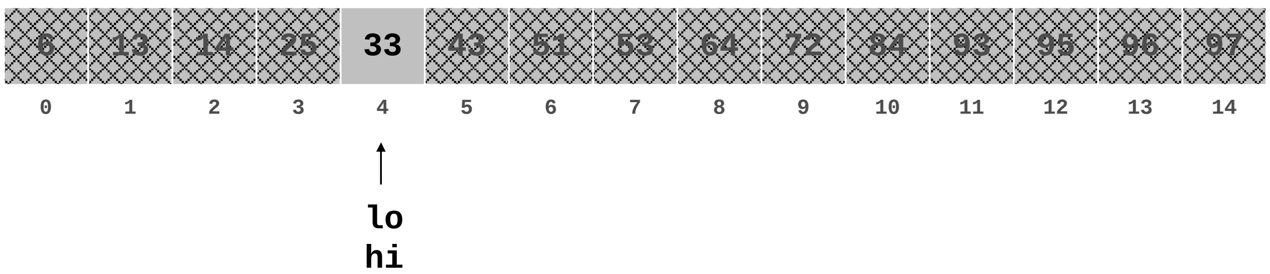


Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\leq$ `value` $\leq$ `a[hi]`.

Ex. Binary search for 33.

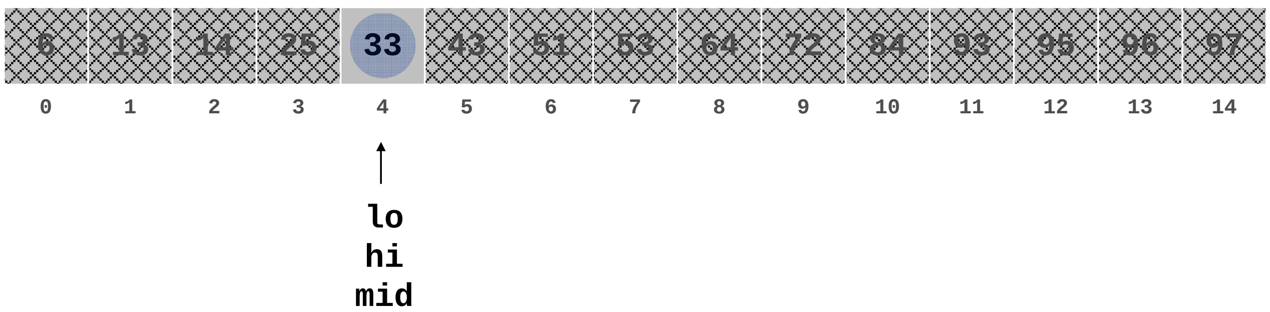


Binary Search

Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.

Invariant. Algorithm maintains a[lo] ≤ value ≤ a[hi].

Ex. Binary search for 33.



Binary Search

Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

Invariant. Algorithm maintains `a[lo]` $\leq$ `value` $\leq$ `a[hi]`.

Ex. Binary search for 33.