CS11212 - **Spring** 2022

Data Structures & Introduction to **Algorithms**

Analysis of Algorithms

Searching & Sorting: Part 1

Ibrahim Albluwi

Problem Description

Problem. Given a list of n elements and a key, check if the key is in the list.
Common variant. Find the position of the key in the list or report failure.

Problem Description

Problem. Given a list of n elements and a key, check if the key is in the list.
Common variant. Find the position of the key in the list or report failure.

Most straightforward algorithm. Linear Search.

```
int search(int a[], int k, int n) {

   for (int i = 0; i < n; i++)
      if (a[i] == k)
         return i;

   return -1;
}
```

3	0	12	5	99	32	49	71	66	82	90	21	1
0	1	2	3	4	5	6	7	8	9	10	11	12

search for 49
return 6

3	0	12	5	99	32	49	71	66	82	90	21	1
0	1	2	3	4	5	6	7	8	9	10	11	12

search for 2
return -1

Problem Description

Problem. Given a list of n elements and a key, check if the key is in the list.
Common variant. Find the position of the key in the list or report failure.

Most straightforward algorithm. Linear Search.

```
int search(int a[], int k, int n) {

   for (int i = 0; i < n; i++)
      if (a[i] == k)
         return i;

   return -1;
}
```

case	key comparisons	explanation
best	1	if k is the first element in the list.
worst	n	if k is not in the list.
average	$\frac{1}{2}(n+1)$	see algorithm analysis slides

Linear Search runs in $O(n)$ time

Searching Sorted Arrays

What if the elements are stored in a sorted array?

Example. Search for 34.

0	1	2	3	4	5	6	7	8	9	10	11	12
0	3	5	12	21	32	49	66	71	82	90	97	99

Searching Sorted Arrays

What if the elements are stored in a sorted array?

```
// precondition: a[] is sorted in ascending order
int search(int a[], int k, int n) {

   for (int i = 0; i < n; i++)
      if       (k == a[i]) return i;
      else if (k <  a[i]) return -1;

   return -1;
}
```

Example. Search for 34. Stop at index 6 (34 < 49) : 34 can't appear after 49.

0	3	5	12	21	32	49	66	71	82	90	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12

Searching Sorted Arrays

What if the elements are stored in a sorted array?

```
// precondition: a[] is sorted in ascending order
int search(int a[], int k, int n) {

   for (int i = 0; i < n; i++)
      if       (k == a[i]) return i;
      else if (k <  a[i]) return -1;

   return -1;
}
```

Example. Search for 34. Stop at index 6 (34 < 49) : 34 can't appear after 49.

0	3	5	12	21	32	49	66	71	82	90	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12

case	key comparisons	explanation
best		
worst		

Searching Sorted Arrays

What if the elements are stored in a sorted array?

```
// precondition: a[] is sorted in ascending order
int search(int a[], int k, int n) {

   for (int i = 0; i < n; i++)
      if        (k == a[i]) return i;
      else if (k <  a[i]) return -1;

   return -1;
}
```

Example. Search for 34. Stop at index 6 (34 < 49) : 34 can't appear after 49.

0	3	5	12	21	32	49	66	71	82	90	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12

case	key comparisons	explanation
best	1	if `k == a[0]`
worst	$2n$	if `k > a[n-1]`

not much improvement over searching in an unsorted array!

This algorithm runs in $O(n)$ time

A Better Solution: Binary Search

Example. Search for 31.

0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

A Better Solution: Binary Search

Example. Search for 31.

`[lo, hi]` is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

examine the middle element first!

A Better Solution: Binary Search

Example. Search for 31.

`[lo, hi]` is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

because `31 < 32`, we can ignore the elements `>= 32`

lo						hi									
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6									

A Better Solution: Binary Search

Example. Search for 31.

[lo, hi] is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

because 31 < 32, we can ignore the elements >= 32

lo			mid			hi									
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6									

A Better Solution: Binary Search

Example. Search for 31.

`[lo, hi]` is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

because 31 < 32, we can ignore the elements >= 32

lo			mid			hi									
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6									

because 31 > 12, we can ignore the elements <= 12

				lo		hi									
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
				4	5	6									

A Better Solution: Binary Search

Example. Search for 31.

`[lo, hi]` is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

because `31 < 32`, we can ignore the elements `>= 32`

lo			mid			hi									
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6									

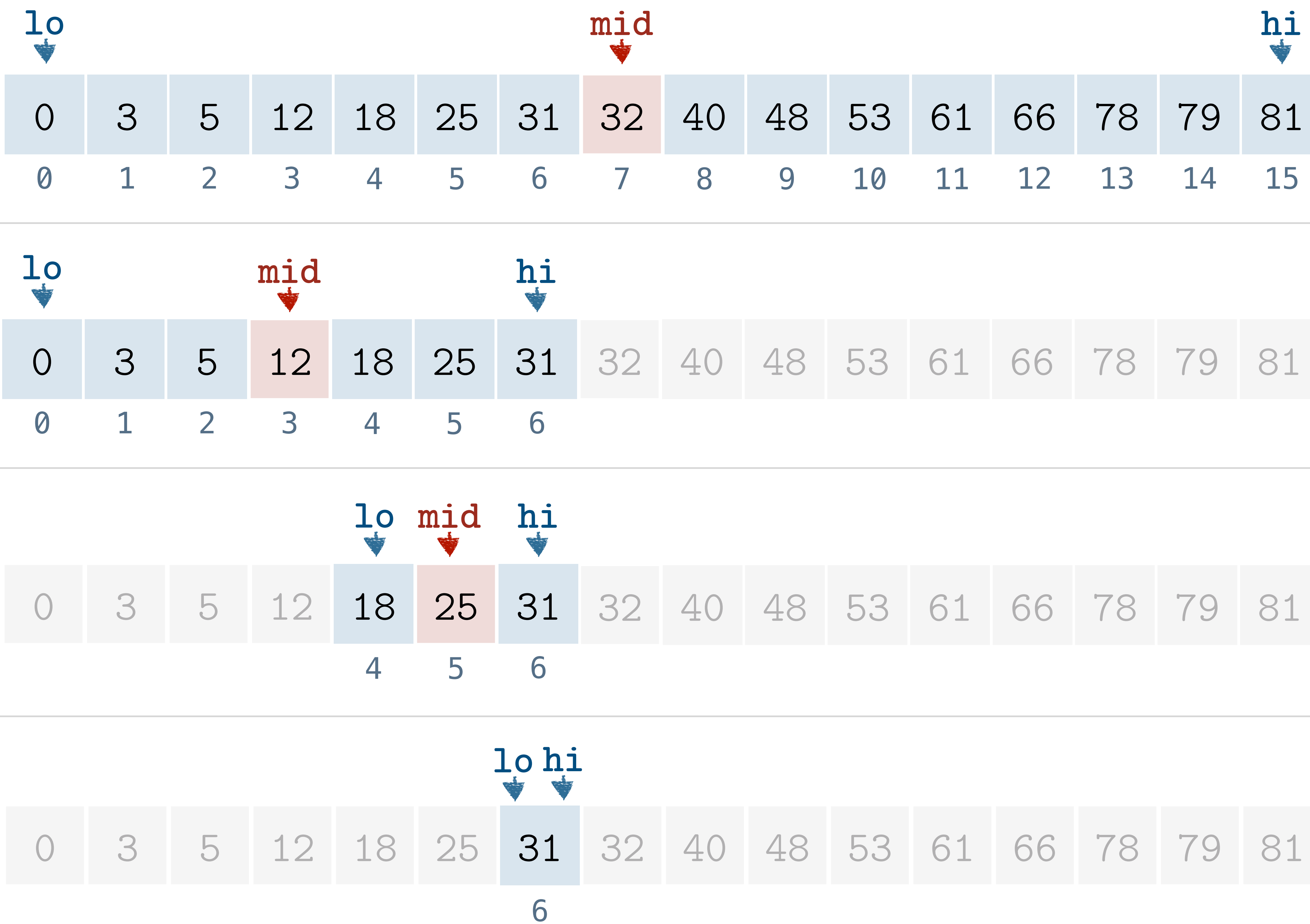
because `31 > 12`, we can ignore the elements `<= 12`

				lo	mid	hi									
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
				4	5	6									

A Better Solution: Binary Search

Example. Search for 31.

`[lo, hi]` is the range of elements we are searching in.



because 31 < 32, we can ignore the elements >= 32

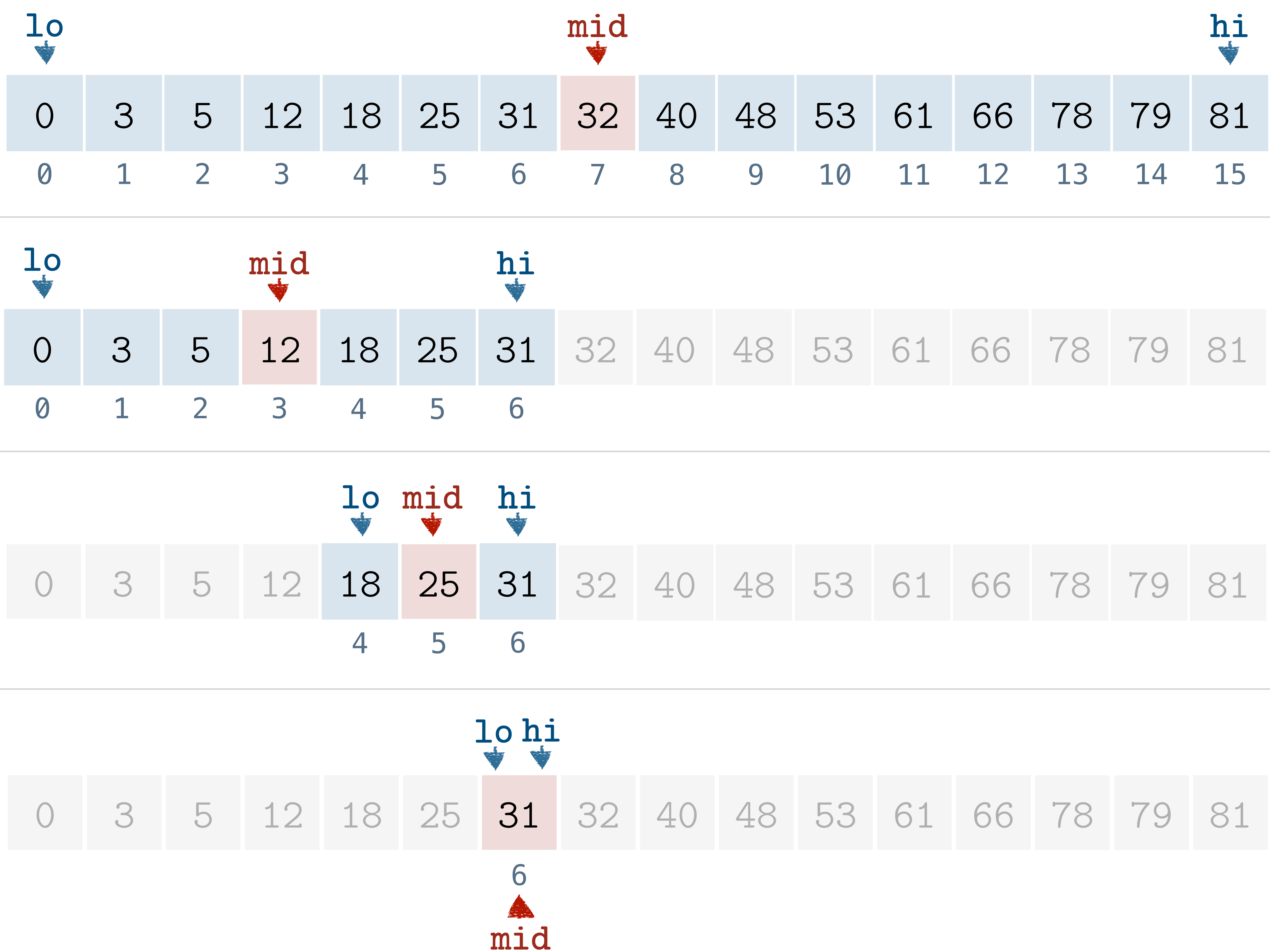
because 31 > 12, we can ignore the elements <= 12

because 31 > 25, we can ignore the elements <= 25

A Better Solution: Binary Search

Example. Search for 31.

[lo, hi] is the range of elements we are searching in.



because 31 < 32, we can ignore the elements >= 32

because 31 > 12, we can ignore the elements <= 12

because 31 > 25, we can ignore the elements <= 25

31 found!

A Better Solution: Binary Search

Example. Search for 41.

`[lo, hi]` is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

A Better Solution: Binary Search

Example. Search for 41.

`[lo, hi]` is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

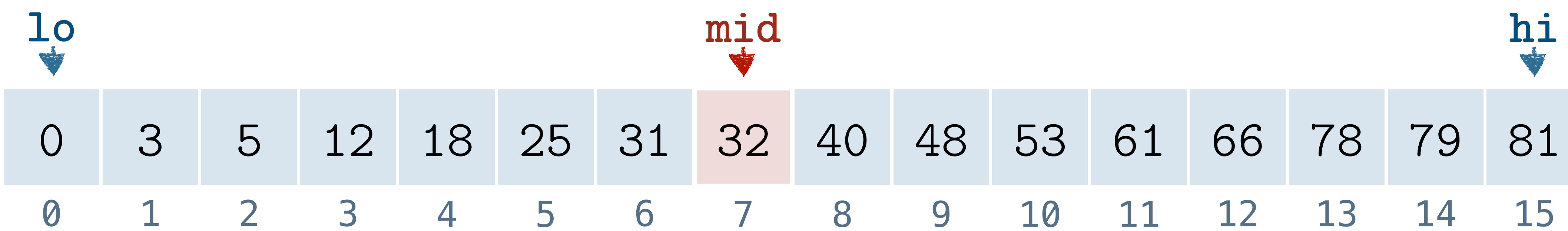
because `41 > 32`, we can ignore the elements `<= 32`

								lo							hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
								8	9	10	11	12	13	14	15

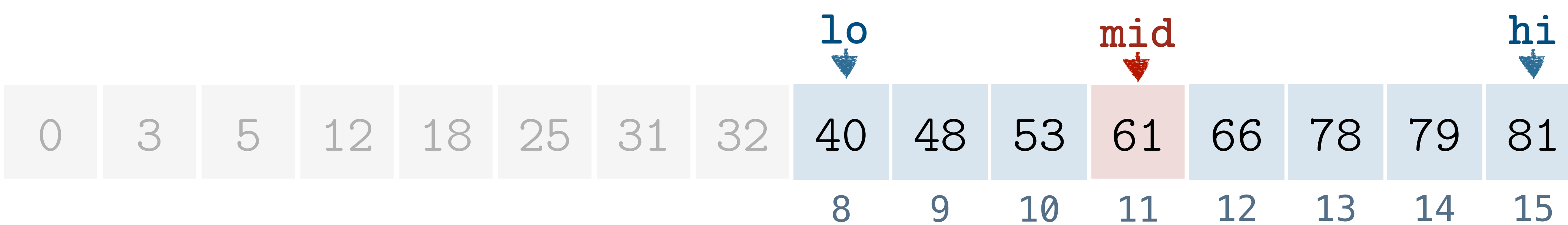
A Better Solution: Binary Search

Example. Search for 41.

`[lo, hi]` is the range of elements we are searching in.



because `41 > 32`, we can ignore the elements `<= 32`



A Better Solution: Binary Search

Example. Search for 41.

[lo, hi] is the range of elements we are searching in.

lo							mid								hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

because 41 > 32, we can ignore the elements <= 32

								lo			mid				hi
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
								8	9	10	11	12	13	14	15

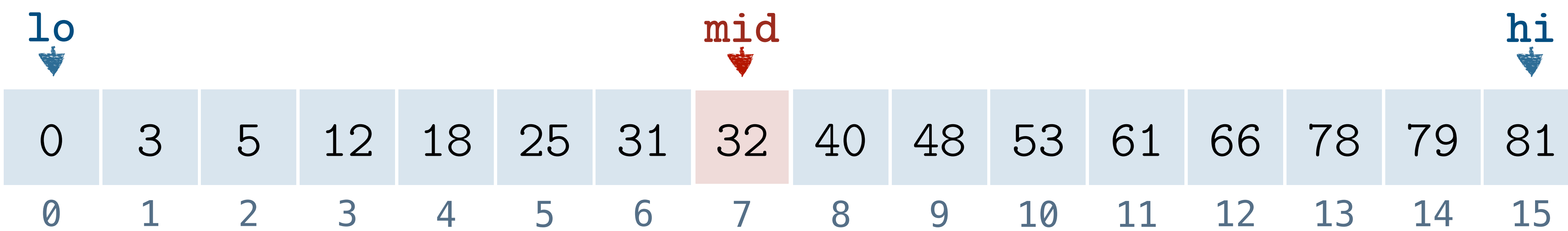
because 41 < 61, we can ignore the elements >= 61

								lo		hi					
0	3	5	12	18	25	31	32	40	48	53	61	66	78	79	81
								8	9	10					

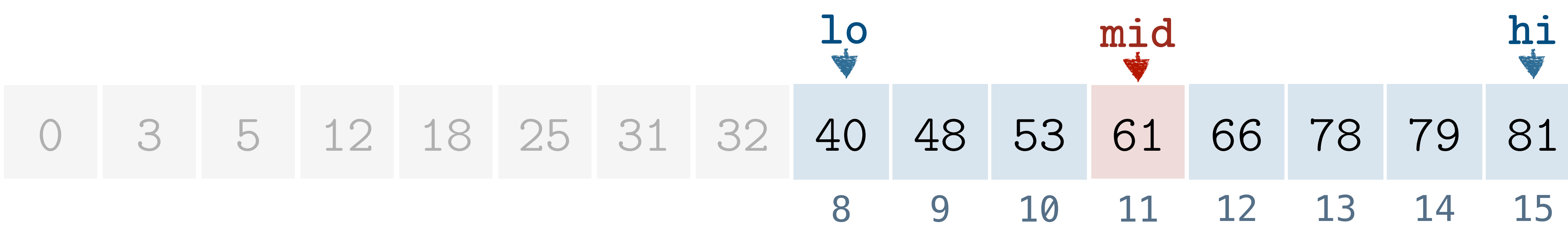
A Better Solution: Binary Search

Example. Search for 41.

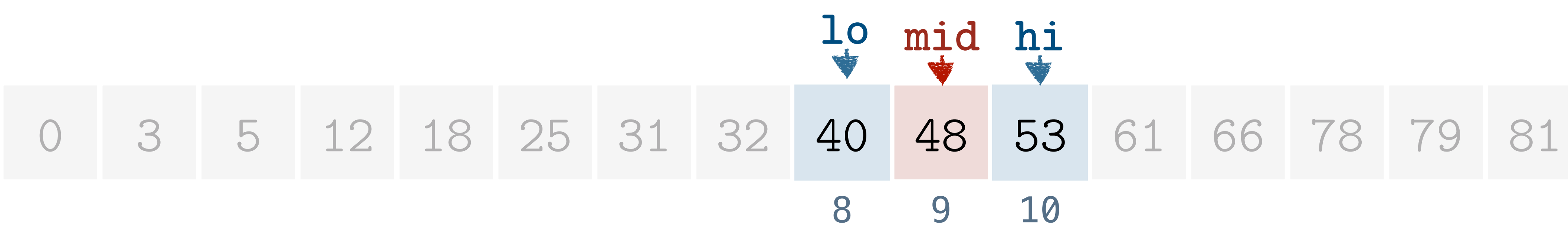
`[lo, hi]` is the range of elements we are searching in.



because `41 > 32`, we can ignore the elements `<= 32`



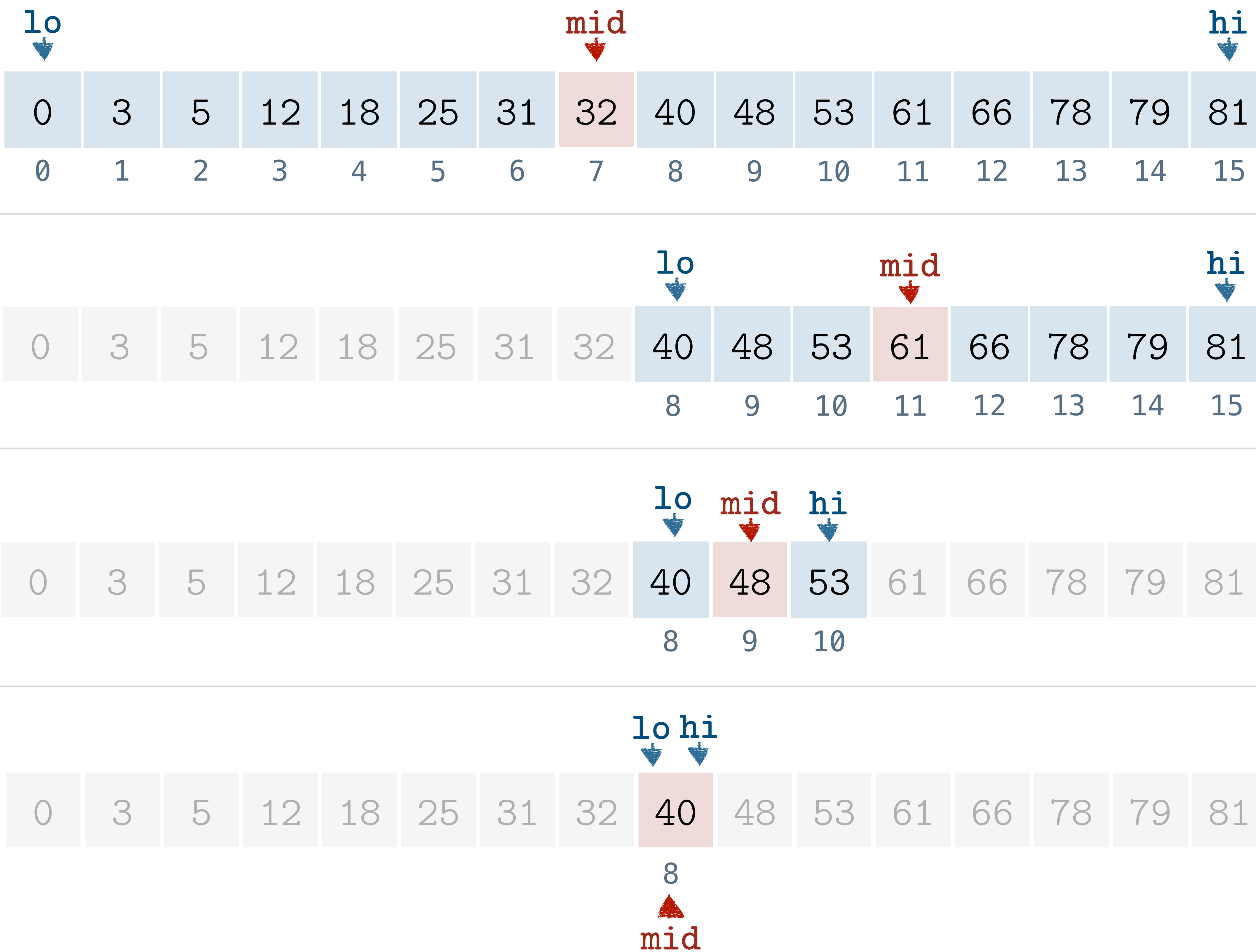
because `41 < 61`, we can ignore the elements `>= 61`



A Better Solution: Binary Search

Example. Search for 41.

`[lo, hi]` is the range of elements we are searching in.



because `41 > 32`, we can ignore the elements `<= 32`

because `41 < 61`, we can ignore the elements `>= 61`

because `41 < 48`, we can ignore the elements `<= 48`

not found!

proceeding in the algorithm makes `lo > hi`

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!

   mid =




}
```

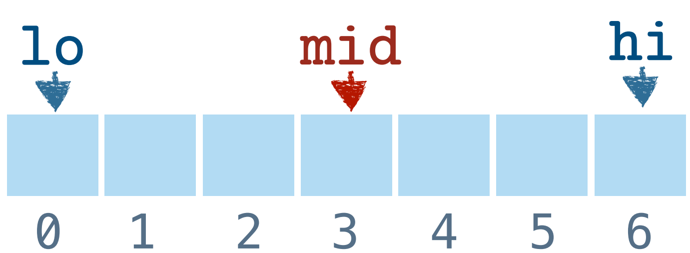
Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!
   mid = lo + (hi-lo) / 2




}
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!
   mid = lo + (hi-lo) / 2



}
```

Examples for computing `mid`:

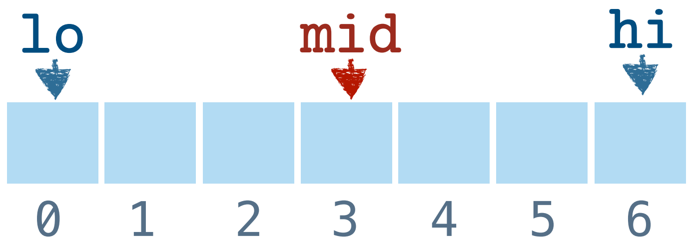



```
mid = 0 + (6-0)/2
    = 0 + 3 = 3
```

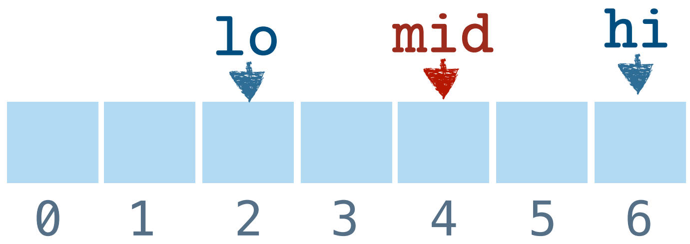
Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!
   mid = lo + (hi–lo) / 2



}
```

Examples for computing `mid`:




```
mid = 0 + (6-0)/2
    = 0 + 3 = 3
```



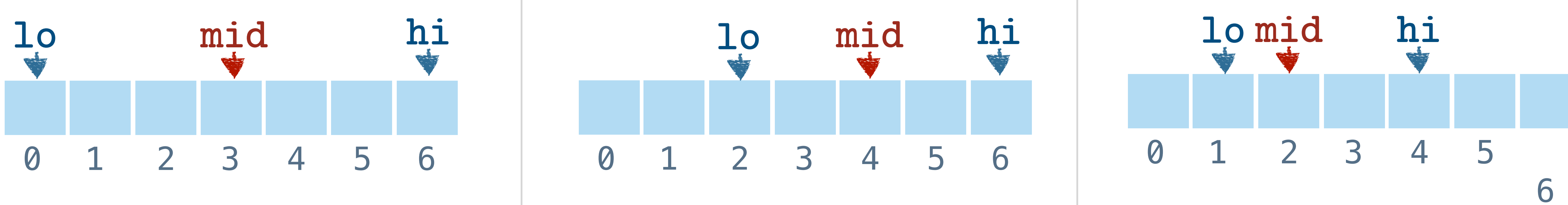

```
mid = 2 + (6-2)/2
    = 2 + 2 = 4
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!
   mid = lo + (hi–lo) / 2

}
```

Think:
What is wrong with
`mid = (lo + hi) / 2` ?

Examples for computing `mid`:




```
mid = 0 + (6-0)/2
    = 0 + 3 = 3
```

```
mid = 2 + (6-2)/2
    = 2 + 2 = 4
```

```
mid = 1 + (4-1)/2
    = 1 + 1 = 2
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!

   mid = lo + (hi-lo) / 2

   if      (k  > a[mid]) search right of mid
   else if (k  < a[mid]) search left of mid
   else                  found!
}
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!

   mid = lo + (hi-lo) / 2

   if      (k  > a[mid]) search right of mid
   else if (k  < a[mid]) search left of mid
   else                  found!
}
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!

   mid = lo + (hi-lo) / 2

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) search left of mid
   else                  found!
}
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) not found!

   mid = lo + (hi−lo) / 2

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid−1)
   else                  found!
}
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                  return mid;
}
```

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                  return mid;
}
```

Recursive

```
int search(int a[], int k, int n) {









}
```

Iterative

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                  return mid;
}
```

Recursive

```
int search(int a[], int k, int n) {

   int lo = 0, hi = n-1;

   while (        ) {





   }

   return -1;
}
```

Iterative

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                  return mid;
}
```

Recursive

```
int search(int a[], int k, int n) {

   int lo = 0, hi = n-1;

   while (lo <= hi) {




   }

   return -1;
}
```

Iterative

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                  return mid;
}
```

Recursive

```
int search(int a[], int k, int n) {

   int lo = 0, hi = n-1;

   while (lo <= hi) {

      int mid = lo + (hi-lo) / 2;



   }

   return -1;
}
```

Iterative

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if       (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                   return mid;
}
```

Recursive

```
int search(int a[], int k, int n) {

   int lo = 0, hi = n-1;

   while (lo <= hi) {

       int mid = lo + (hi-lo) / 2;

       if       (k  > a[mid]) lo = mid+1;


   }

   return -1;
}
```

Iterative

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                  return mid;
}
```

Recursive

```
int search(int a[], int k, int n) {

   int lo = 0, hi = n-1;

   while (lo <= hi) {

      int mid = lo + (hi-lo) / 2;

      if      (k  > a[mid]) lo = mid+1;
      else if (k  < a[mid]) hi = mid-1;

   }

   return -1;
}
```

Iterative

Binary Search: Implementation

```
int search(int a[], int k, int lo, int hi) {
   if (lo > hi) return -1;

   int mid = lo + (hi-lo) / 2;

   if      (k  > a[mid]) return search(a, k, mid+1, hi)
   else if (k  < a[mid]) return search(a, k, lo, mid-1)
   else                  return mid;
}
```

Recursive

```
int search(int a[], int k, int n) {

   int lo = 0, hi = n-1;

   while (lo <= hi) {

      int mid = lo + (hi-lo) / 2;

      if      (k  > a[mid]) lo = mid+1;
      else if (k  < a[mid]) hi = mid-1;
      else                  return mid;
   }

   return -1;
}
```

Iterative

Binary Search: Analysis

```
int search(int a[], int k, int n) {
   int lo = 0, hi = n-1;
   while (lo <= hi) {
       int mid = lo + (hi-lo) / 2;
       if      (k  > a[mid]) lo = mid+1;
       else if (k  < a[mid]) hi = mid-1;
       else if (k == a[mid]) return mid;
   }
   return -1;
}
```

case	key comparisons	explanation
best		
worst		

Binary Search: Analysis

```
int search(int a[], int k, int n) {
   int lo = 0, hi = n-1;
   while (lo <= hi) {
       int mid = lo + (hi-lo) / 2;
       if      (k  > a[mid]) lo = mid+1;
       else if (k  < a[mid]) hi = mid-1;
       else if (k == a[mid]) return mid;
   }
   return -1;
}
```

case	key comparisons	explanation
best	2	if `k  = a[mid]`
worst		

Binary Search: Analysis

```
int search(int a[], int k, int n) {
   int lo = 0, hi = n-1;
   while (lo <= hi) {
      int mid = lo + (hi-lo) / 2;
      if      (k  > a[mid]) lo = mid+1;
      else if (k  < a[mid]) hi = mid-1;
      else                  return mid;
   }
   return -1;
}
```

case	key comparisons	explanation
best	2	if `k = a[mid]`
worst		

Each iteration reduces the current search range by half.
How many steps are there from n to 1 by halving?

Binary Search: Analysis

```
int search(int a[], int k, int n) {
   int lo = 0, hi = n-1;
   while (lo <= hi) {
       int mid = lo + (hi-lo) / 2;
       if      (k  > a[mid]) lo = mid+1;
       else if (k  < a[mid]) hi = mid-1;
       else                  return mid;
   }
   return -1;
}
```

case	key comparisons	explanation
best	2	if `k = a[mid]`
worst	$2(\lfloor \log_2(n) \rfloor + 1)$	if `k < a[0]` 2 comparisons per iteration and $\lfloor \log_2(n) \rfloor + 1$ iterations

Each iteration reduces the current search range by half.
How many steps are there from n to 1 by halving?

Binary Search runs in $O(\log n)$ time

works only if the array is sorted

Binary Search: Analysis

```
int search(int a[], int k, int n) {
   int lo = 0, hi = n-1;
   while (lo <= hi) {
       int mid = lo + (hi-lo) / 2;
       if         (k  > a[mid]) lo = mid+1;
       else if (k  < a[mid]) hi = mid-1;
       else                        return mid;
   }
   return -1;
}
```

Food for thought:
How can we do 1 comparison per iteration instead of 2 in the worst case?

case	key comparisons	explanation
best	2	if `k  = a[mid]`.
worst	$2(\lfloor \log_2(n) \rfloor + 1)$	if `k < a[0]` 2 comparisons per iteration and $\lfloor \log_2(n) \rfloor + 1$ iterations

Each iteration reduces the current search range by half.
How many steps are there from n to 1 by halving?

Binary Search runs in $O(\log n)$ time

works only if the array is sorted

How good is Binary Search?

Worst case performance. Binary Search vs Linear Search

	binary search*		linear search	
array size	# of compares	time	# of compares	time
1,000	10		1,000	
1,000,000	20		1,000,000	
10,000,000	24		10,000,000	
100,000,000	27		100,000,000	
200,000,000	28		200,000,000	
400,000,000	29		400,000,000	
800,000,000	30		800,000,000	
1,600,000,000	31		1,600,000,000	

* assuming an optimized implementation of binary search that does 1 comparison per iteration, not 2.

How good is Binary Search?

Worst case performance. Binary Search vs Linear Search

	binary search		linear search	
array size	# of compares	time	# of compares	time
1,000	10		1,000	
1,000,000	20		1,000,000	
10,000,000	24		10,000,000	
100,000,000	27		100,000,000	
200,000,000	28		200,000,000	
400,000,000	29		400,000,000	
800,000,000	30		800,000,000	
1,600,000,000	31		1,600,000,000	

comparisons increase by 1 when the array size doubles!

* assuming an optimized implementation of binary search that does 1 comparison per iteration, not 2.

Logarithmic growth

How good is Binary Search?

Worst case performance. Binary Search vs Linear Search

	binary search		linear search	
array size	# of compares	time	# of compares	time
1,000	10	instant	1,000	instant
1,000,000	20	instant	1,000,000	0.0024 sec
10,000,000	24	instant	10,000,000	0.023 sec
100,000,000	27	instant	100,000,000	0.22 sec
200,000,000	28	instant	200,000,000	0.43 sec
400,000,000	29	instant	400,000,000	0.84 sec
800,000,000	30	instant	800,000,000	1.65 sec
1,600,000,000	31	instant	1,600,000,000	3.6 sec

Tests done on a 2.6 GHz 6-Core Intel Core i7 MacBook Pro with 16 GB DDR4 RAM

instant = a few microseconds or less

How good is Binary Search?

Worst case performance. Binary Search vs Linear Search

	binary search		linear search	
array size	# of compares	time	# of compares	time
1,000	10	instant	1,000	instant
1,000,000	20	instant	1,000,000	0.0024 sec
10,000,000	24	instant	10,000,000	0.023 sec
100,000,000	27	instant	100,000,000	0.22 sec
200,000,000	28	instant	200,000,000	0.43 sec
400,000,000	29	instant	400,000,000	0.84 sec
800,000,000	30	instant	800,000,000	1.65 sec
1,600,000,000	31	instant	1,600,000,000	3.6 sec

Tests done on a 2.6 GHz 6-Core Intel Core i7 MacBook Pro with 16 GB DDR4 RAM

Fadi: Who cares? I won't have more than 10,000,000 elements in my array!

Shadi: Are you sure it won't matter?

Fadi's Application

A web filter. Receives requests to check if a website is blacklisted.

- chegggg.com
- ghashashoon.jo
- ghashishni.com
- ghashashto.com
- ghoshexpert.jo
- learnghosh.com
- junk.food.com
- mkoren.com
- dont-study.com

...
...

Blacklist

Fadi's Blacklist app has ~10,000,000 websites and expects to support millions of daily requests.

Fadi's Application

A web filter. Receives requests to check if a website is blacklisted.

chegggg.com
ghashashoon.jo
ghashishni.com
ghashashto.com
ghoshexpert.jo
learnghosh.com
junk.food.com
mkoren.com
dont-study.com
⋮

Blacklist

Fadi's Blacklist app has ~10,000,000 websites and expects to support millions of daily requests.

Linear search can support

$$\approx \frac{1}{0.023} \approx 43 \text{ requests / sec}$$

Binary search, can support

$$\approx \frac{1}{2 \times 10^{-6}} \approx \frac{1}{2} \text{ million requests / sec}$$

Fadi's Application

A web filter. Receives requests to check if a website is blacklisted.

cheggggg.com
ghashashoon.jo
ghashishni.com
ghashashto.com
ghoshexpert.jo
learnghosh.com
junk.food.com
mkoren.com
dont-study.com
...
...

Blacklist

Fadi's Blacklist app has ~10,000,000 websites and expects to support millions of daily requests.

Linear search can support

$$\approx \frac{1}{0.023} \approx 43 \text{ requests / sec}$$

Binary search, can support

$$\approx \frac{1}{2 \times 10^{-6}} \approx \frac{1}{2} \text{ million requests / sec}$$

! Linear search does not scale well; binary search does!

Binary Search: A Simple Algorithm?

1946: First published version of binary search.
1960: First published version of binary search that worked for arbitrary array sizes! [ref]

Binary Search: A Simple Algorithm?

1946: First published version of binary search.
1960: First published version of binary search that worked for arbitrary array sizes! [ref]

1988: A survey found 15/20 major CS1/CS2 textbooks to have errors in their binary search implementations! [ref]

Binary Search: A Simple Algorithm?

1946: First published version of binary search.
1960: First published version of binary search that worked for arbitrary array sizes! [ref]

1988: A survey found 15/20 major CS1/CS2 textbooks to have errors in their binary search implementations! [ref]

2006: A bug in Java's implementation of binary search was found after ~20 years of use!

Google AI Blog

The latest news from Google AI

Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken

Friday, June 2, 2006

Posted by Joshua Bloch, Software Engineer

I remember vividly Jon Bentley's first Algorithms lecture at CMU, where he asked all of us incoming Ph.D. students to write a binary search, and then dissected one of our implementations in front of the class. Of course it was broken, as were most of our implementations. This made a real impression on me, as did the treatment of this material in his wonderful *Programming Pearls* (Addison-Wesley, 1986; Second Edition, 2000). The key lesson was to carefully consider the invariants in your programs.

https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

Things you need to know for the exam

You need to be able to:

- implement (and trace) linear search and binary search iteratively and recursively.
- implement and trace minor modified versions of both algorithms (like the one slide 5).
- analyze the worst and best case running times of linear search, binary search and simple modifications of these algorithms (like the one on slide 5).

You don't need to know:

- What Fadi's app does.
- How many seconds linear search takes on Dr. Ibrahim's machine for different input sizes.
- How many textbooks implemented binary search incorrectly or for how many years Java's binary search implementation had an overflow bug.