Binary Search

Phone book

- Like linear search, binary search finds whether a certain item (the key) is in a vector.
- Binary search only works on ***sorted*** vectors.
  - Binary search takes advantage of the vector being sorted to make the search much faster.

key = 33

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

key = 33

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

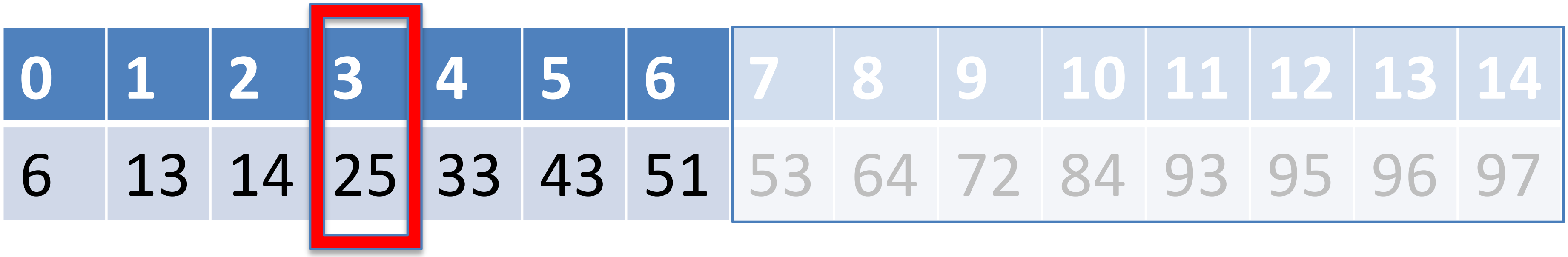
key = 33

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

key = 33

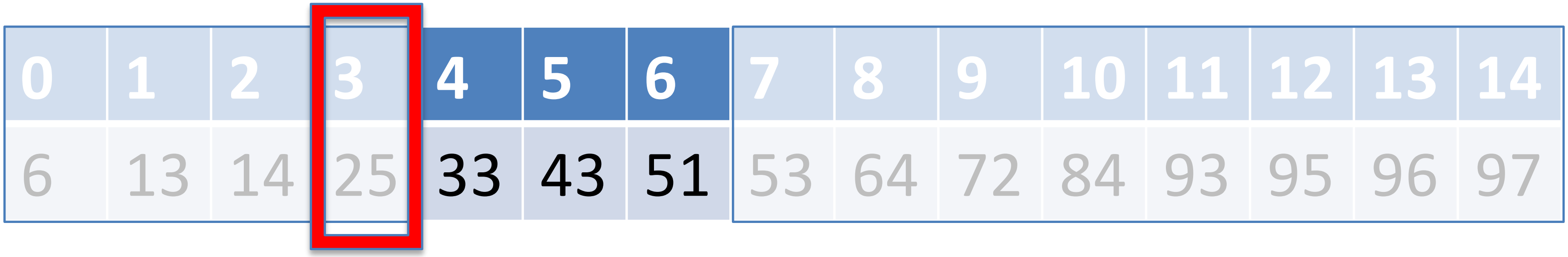
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97

key = 33



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

key = 33

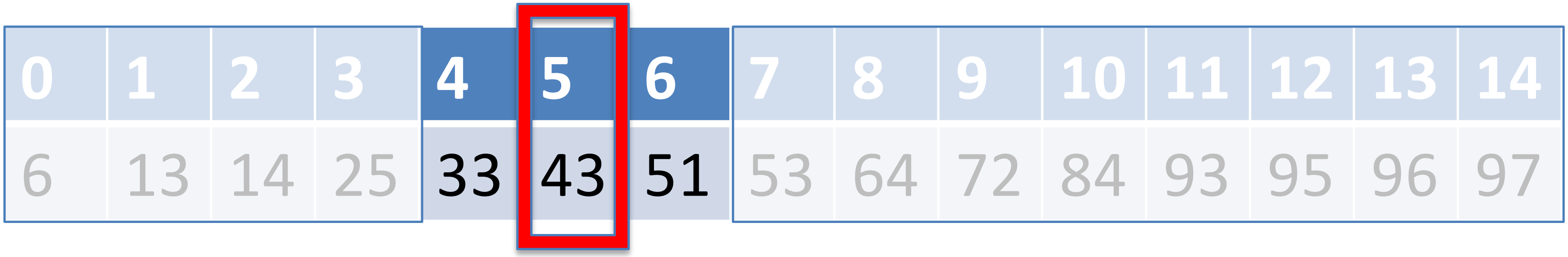


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97

key = 33

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

key = 33



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

key = 33

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

key = 33

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

key = 33

Found! (return 4)

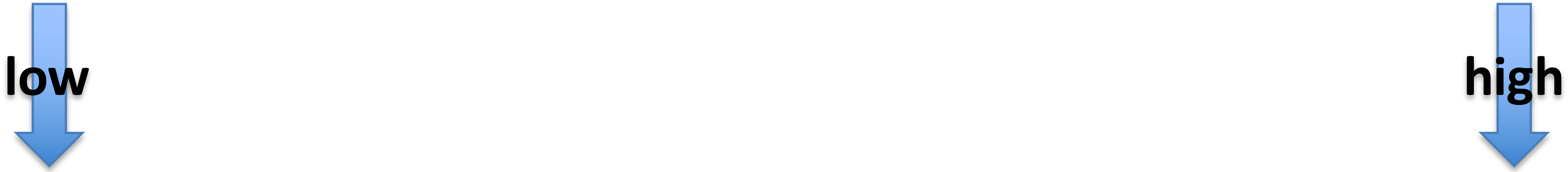
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97

- We are given
  - an array **A** of size n, indexed from 0 to n-1
  - an integer **key** to look for in the array
  - an integer **low** that is the lowest index in the array that could contain the key
  - an integer **high** that is the highest index in the array that could contain the key
- If **low** > **high**, then the item is not found (return -1)
- Compute the **middle** position in the array.
- If the item at the **middle** position is the key, return that position.
- If the item at the **middle** position is greater than the **key**, repeat from step 2, on the left sub-array.
- If the item at the **middle** position is less than the **key**, repeat from step 2 on the right sub-array.

- Three variables that do most of the work:
  - low: the smallest index that could possibly contain the key.
  - high: the largest index that could possibly contain the key.
  - mid: the midpoint of the two indices.

- If low > high, we know the item is not found (stop).
- If array[mid] == key, item is found (stop).
- If array[mid] > key, repeat algorithm with only the left half of the array.
- If array[mid] < key, repeat algorithm with only the right half of the array.

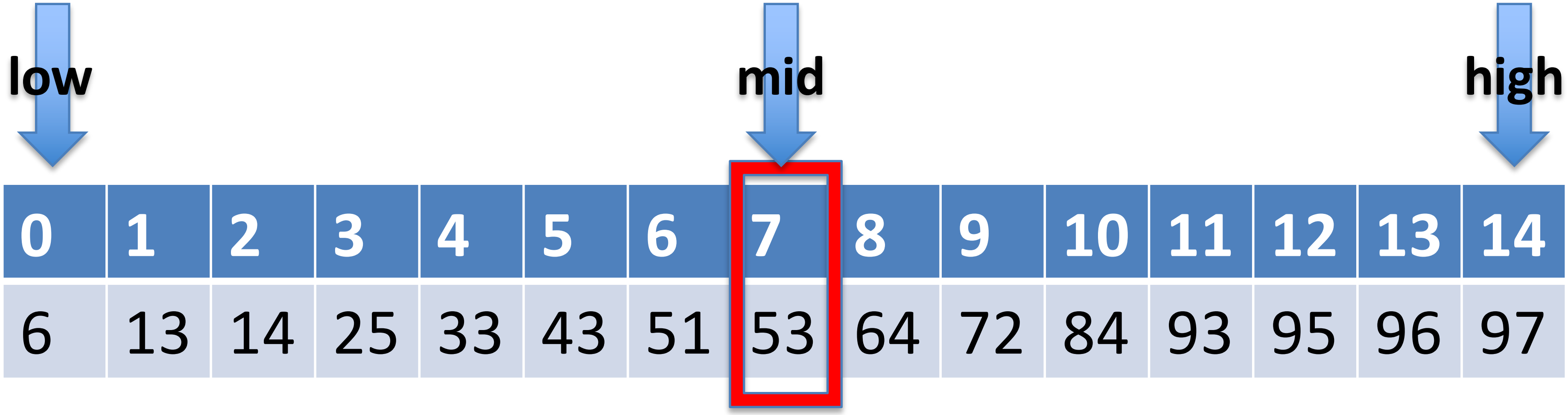
key = 33



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

low = 0
high = 14
mid =

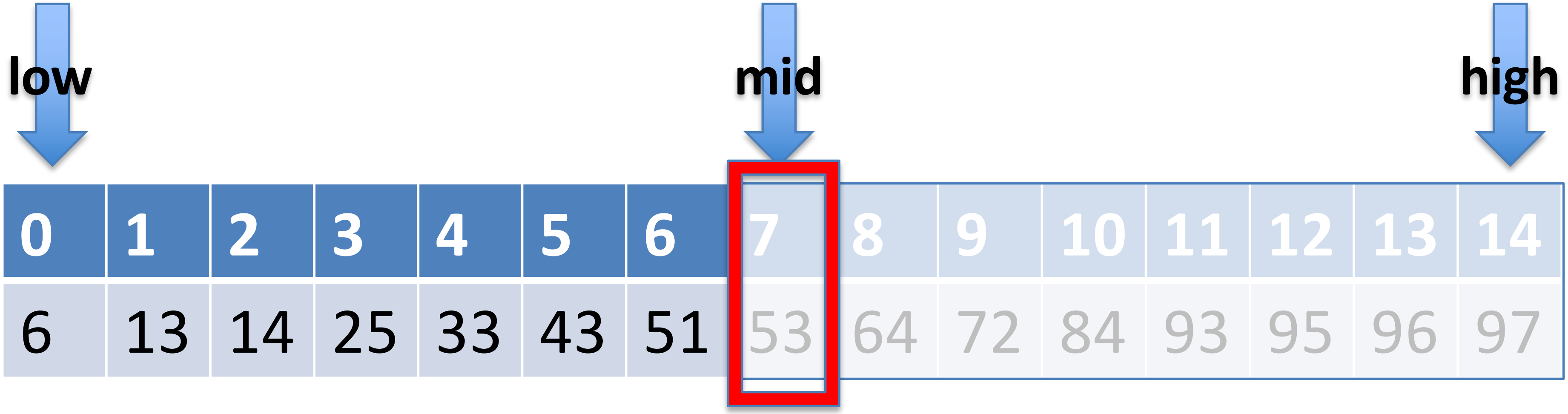
key = 33



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

low = 0
high = 14
mid = 7

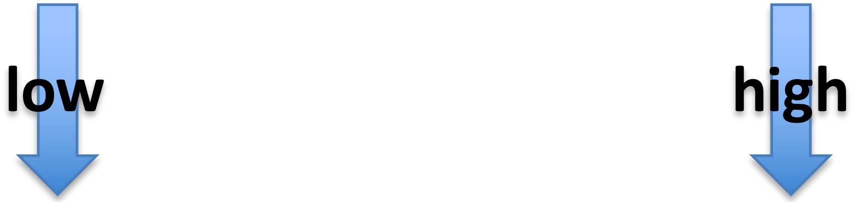
key = 33



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

low = 0
high = 14
mid = 7

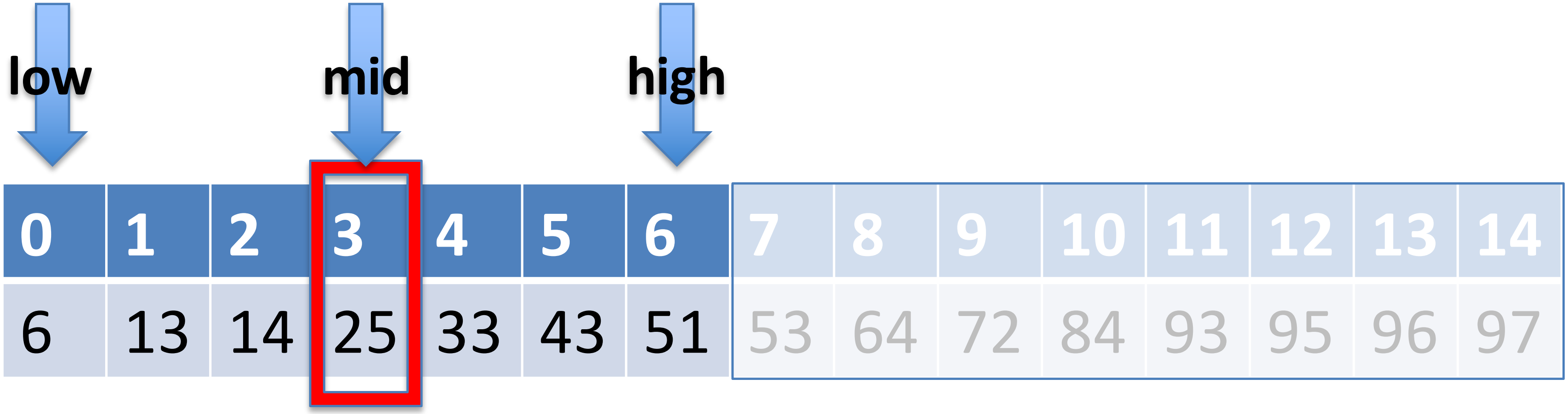
key = 33



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

low = 0
high = 6
mid =

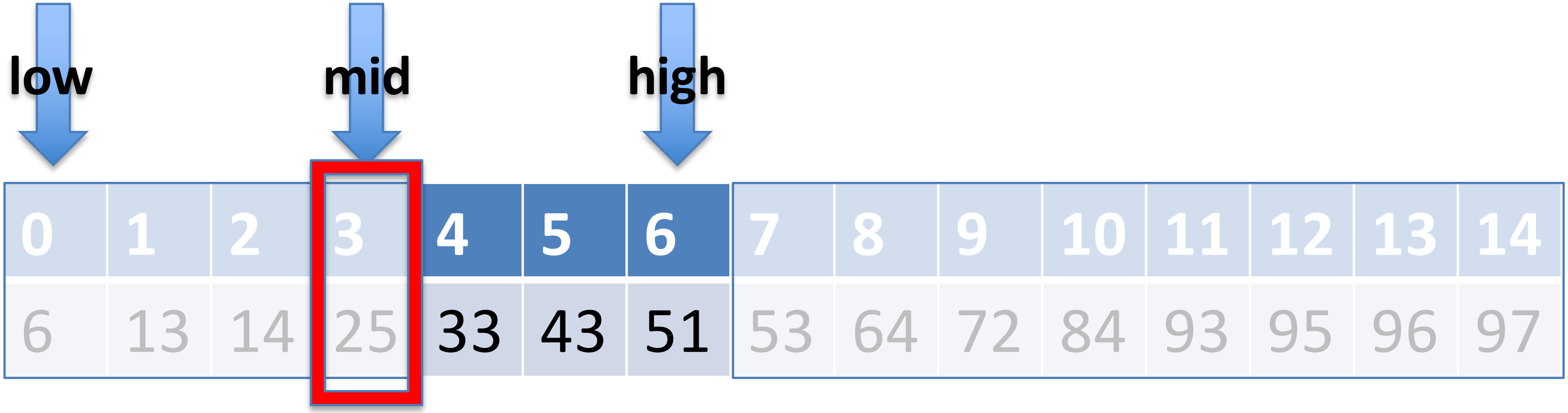
key = 33



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

low = 0
high = 6
mid = 3

key = 33



low

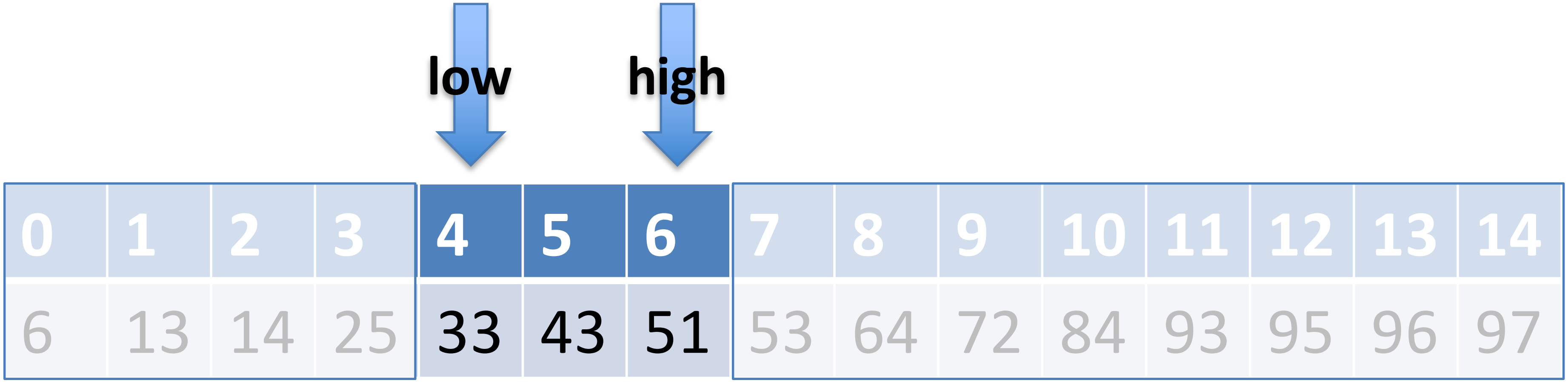
mid

high

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

low = 0
high = 6
mid = 3

key = 33



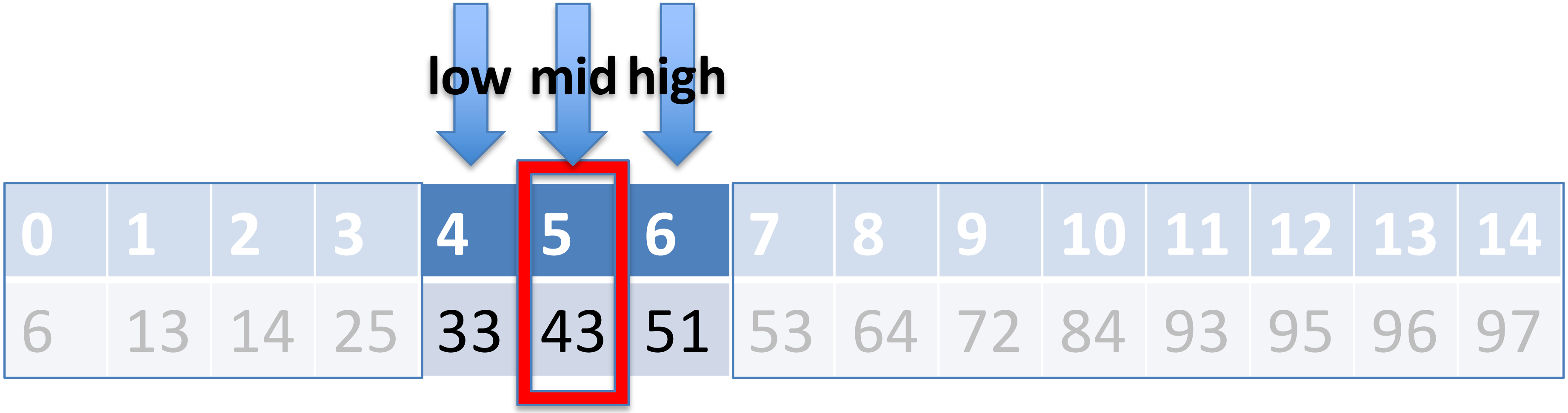
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97

low = 4

high = 6

mid =

key = 33

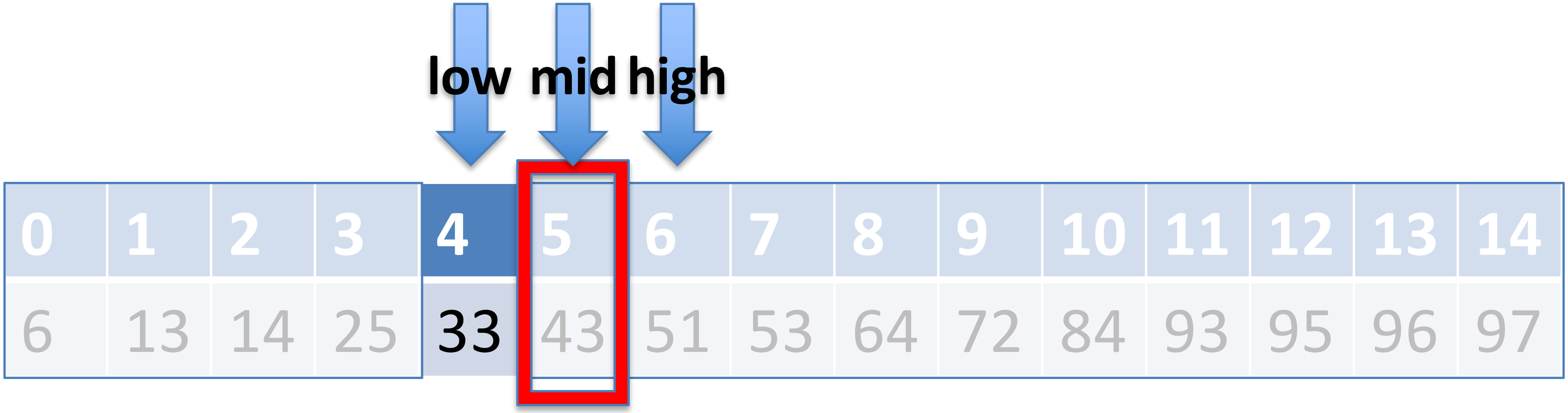


low mid high

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

low = 4
high = 6
mid = 5

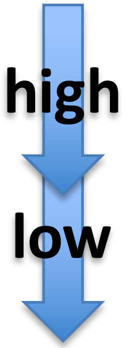
key = 33



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

low = 4
high = 6
mid = 5

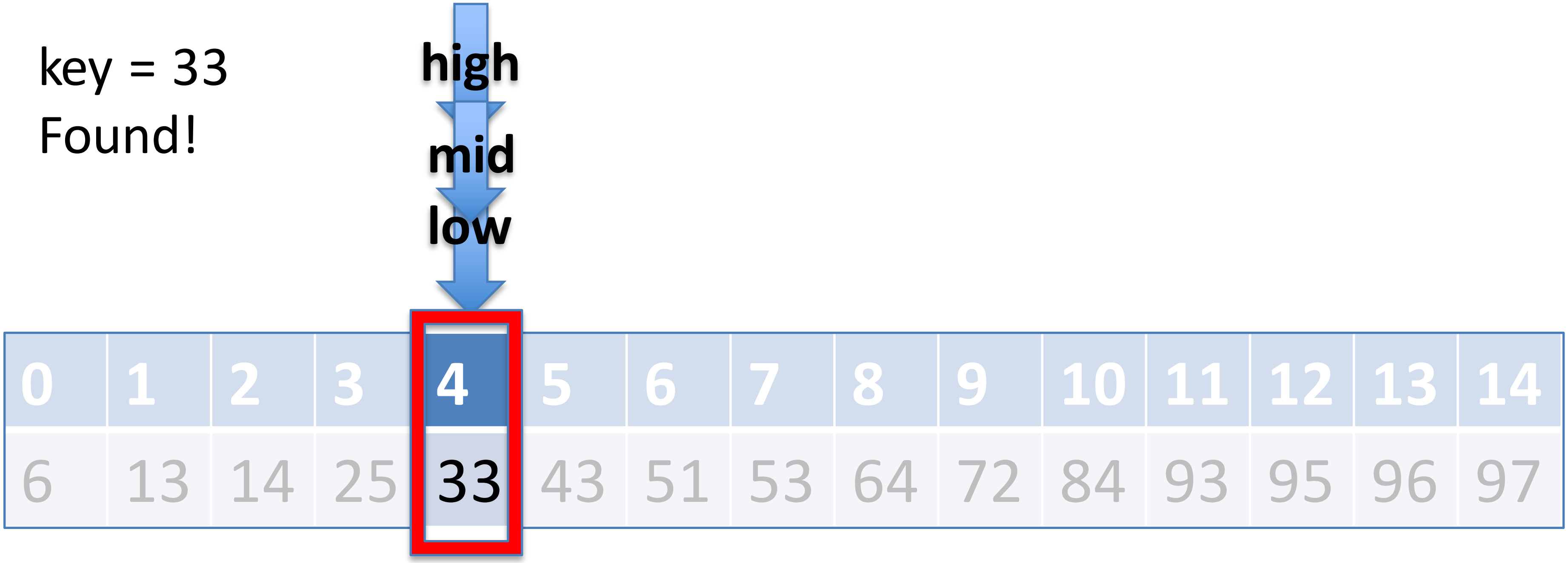
key = 33



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

low = 4
high = 4
mid =

key = 33
Found!



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

low = 4
high = 4
mid = 4

- If low > high, we know the item is not found (stop).
- If array[mid] == key, item is found (stop).
- If array[mid] > key, repeat algorithm ***with only the left half of the array***.
  - How do we change low & high?
- If array[mid] < key, repeat algorithm ***with only the right half of the array.***
  - How do we change low & high?

- If low > high, we know the item is not found (stop).
- If array[mid] == key, item is found (stop).
- If array[mid] > key, repeat algorithm ***with only the left half of the array***.
  - How do we change low & high?
  - **high = mid - 1**
- If array[mid] < key, repeat algorithm ***with only the right half of the array.***
  - How do we change low & high?
  - **low = mid + 1**

Recursive formulation

- Function: binary_search(A, key, low, high)
- Base cases:
  - If found key: Return position found.
  - If low > high: Return -1 (indicating not found).
- Recursive cases:
  - If A[mid] > key: binary_search(A, key, low, mid – 1)
  - If A[mid] < key: binary_search(A, key, mid + 1, high)