Binary Search

Shusen Wang

Stevens Institute of Technology

Problem of Search

arr =

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

Problem of Search

`arr =`

3 $_0$	5 $_1$	12 $_2$	16 $_3$	17 $_4$	26 $_5$	32 $_6$	51 $_7$	53 $_8$	64 $_9$

- **Inputs:** (i) an array whose elements are in the ascending order and (ii) a key.
- **Goal:** Search the key in the array. If found, return its index; if not found, return -1.

Problem of Search

`arr =`

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

Example 1:

- Search for the element 53.
- Return 8.

Problem of Search

`arr =`

3_{0}	5_{1}	12_{2}	16_{3}	17_{4}	26_{5}	32_{6}	51_{7}	53_{8}	64_{9}

Example 2:

- Search for the element 9.
- Return -1.

Example: key=26

arr =

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

`left=0`

`right=9`

Example: key=26

arr =

3_0	5_1	12_2	16_3	17_4	26_5	32_6	51_7	53_8	64_9

left=0

right=9

- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right ← mid-1.
- If arr[mid]<key ==> left ← mid+1.

Example: key=26

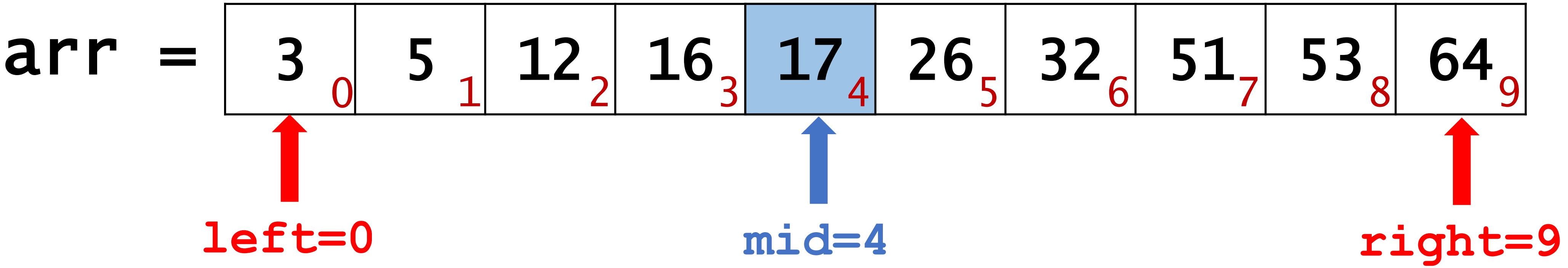
arr =

0	1	2	3	4	5	6	7	8	9
3	5	12	16	17	26	32	51	53	64

left=0 mid=4 right=9

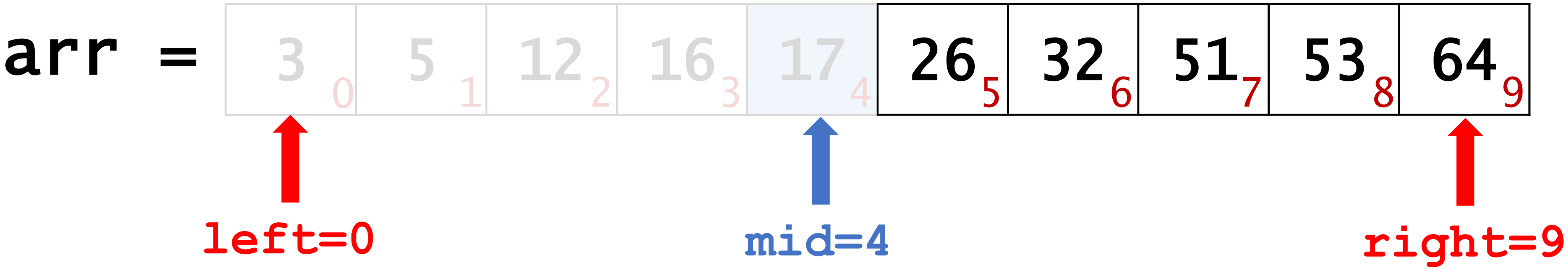
- mid $\leftarrow \lfloor(\text{left} + \text{right})/2\rfloor$.
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right ← mid-1.
- If arr[mid]<key ==> left ← mid+1.

Example: key=26



- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key  ==> return mid.`
- `If arr[mid]>key   ==> right ← mid-1.`
- **`If arr[mid]<key   ==> left ← mid+1.`**

Example: key=26



- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key    ==> return mid.`
- `If arr[mid]>key     ==> right ← mid-1.`
- `If arr[mid]<key     ==> left ← mid+1.`

Example: key=26

`arr =`

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

`left=5` `right=9`

- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key   ==> return mid.`
- `If arr[mid]>key    ==> right ← mid-1.`
- `If arr[mid]<key    ==> left ← mid+1.`

Example: key=26

`arr =`

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

`left=5` `right=9`

- mid $\leftarrow \lfloor(\text{left} + \text{right})/2\rfloor$.
- `If arr[mid]==key  ==> return mid.`
- `If arr[mid]>key   ==> right ← mid-1.`
- `If arr[mid]<key   ==> left ← mid+1.`

Example: key=26

arr =

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

`left=5` `mid=7` `right=9`

- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key  ==> return mid.`
- `If arr[mid]>key   ==> right ← mid-1.`
- `If arr[mid]<key   ==> left ← mid+1.`

Example: key=26

arr =

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

left=5 mid=7 right=9

- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key ==> return mid.`
- `If arr[mid]>key ==> right← mid-1.`
- `If arr[mid]<key ==> left ← mid+1.`

Example: key=26

arr =

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

left=5 mid=7 right=9

- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key ==> return mid.`
- `If arr[mid]>key ==> right← mid-1.`
- `If arr[mid]<key ==> left ← mid+1.`

Example: key=26

arr =

value	3	5	12	16	17	26	32	51	53	64
index	0	1	2	3	4	5	6	7	8	9

left=5

right=6

- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key` `==>` `return mid.`
- `If arr[mid]>key` `==>` `right` $\leftarrow$ `mid-1.`
- `If arr[mid]<key` `==>` `left` $\leftarrow$ `mid+1.`

Example: key=26

arr =

3	5	12	16	17	26	32	51	53	64
0	1	2	3	4	5	6	7	8	9

left=5

right=6

- mid $\leftarrow \lfloor(\text{left} + \text{right})/2\rfloor$.
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right← mid-1.
- If arr[mid]<key ==> left← mid+1.

Example: key=26

- mid $\leftarrow \lfloor(\text{left} + \text{right})/2\rfloor$.
- If `arr[mid]==key` ==> `return mid.`
- If `arr[mid]>key` ==> `right← mid-1.`
- If `arr[mid]<key` ==> `left← mid+1.`

Example: key=26

- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key   ==> return mid.`
- `If arr[mid]>key    ==> right← mid-1.`
- `If arr[mid]<key    ==> left← mid+1.`

Time Complexity

- Each iteration reduces the size of array by half.
- Let n be the size of the array.
- The total number of iterations is at most $\log_2 n$.
- Time complexity: $O(\log n)$.

```
int search(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] == key)
            return mid; // key is found
        if (arr[mid] < key)
            left = mid + 1; // search in the right half
        else
            right = mid - 1; // search in the left half
    }
    return -1; // key is not found
}
```

```
int search(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] == key)
            return mid; // key is found
        if (arr[mid] < key)
            left = mid + 1; // search in the right half
        else
            right = mid - 1; // search in the left half
    }
    return -1; // key is not found
}
```

```
int search(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] == key)
            return mid; // key is found
        if (arr[mid] < key)
            left = mid + 1; // search in the right half
        else
            right = mid - 1; // search in the left half
    }
    return -1; // key is not found
}
```

```
int search(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (key == arr[mid])
            return mid; // key is found
        if (key > arr[mid])
            left = mid + 1; // search in the right half
        else
            right = mid - 1; // search in the left half
    }
    return -1; // key is not found
}
```

```
int search(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (key == arr[mid])
            return mid; // key is found
        if (key > arr[mid])
            left = mid + 1; // search in the right half
        else
            right = mid - 1; // search in the left half
    }
    return -1; // key is not found
}
```

```
int search(int arr[], int left, int right, int key) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] == key)
            return mid; // key is found
        if (arr[mid] < key)
            left = mid + 1; // search in the right half
        else
            right = mid - 1; // search in the left half
    }
    return -1; // key is not found
}
```

How to support both search and insertion?

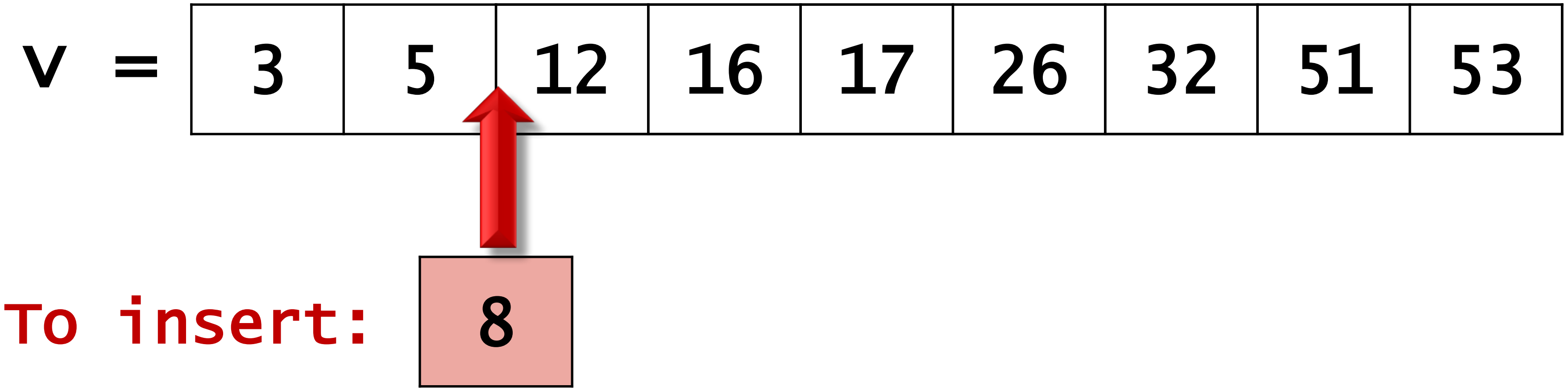
Vector

v =

3	5	12	16	17	26	32	51	53

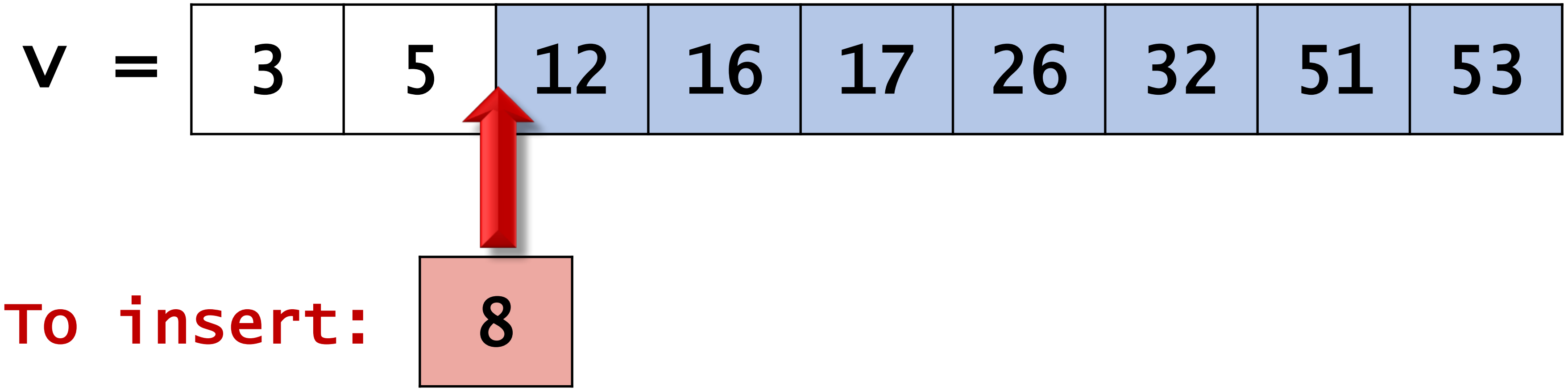
- The ascending order must be kept; otherwise, search would take $O(n)$ time.
- Inserting an item into the middle has $O(n)$ time complexity (on average).

Vector



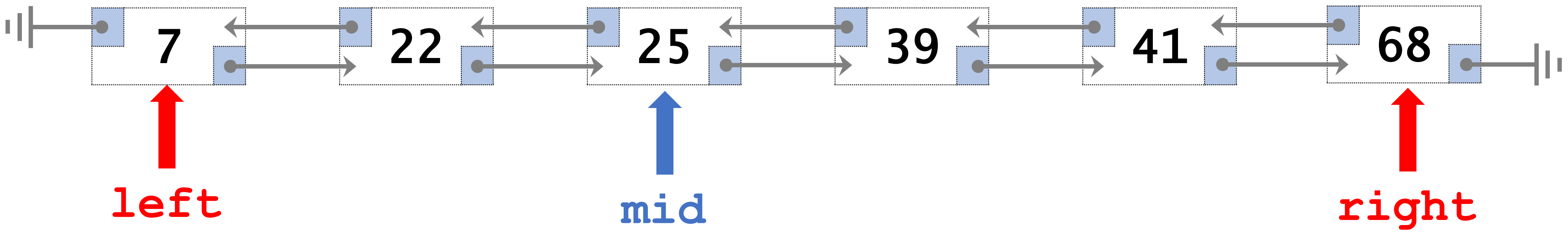
- Inserting an item into the middle has $O(n)$ time complexity (on average).

Vector



- Inserting an item into the middle has $O(n)$ time complexity (on average).

List



- Can we perform binary search in the list?
- No. Given left and right, we cannot get mid efficiently.

Comparisons

	Search	Insertion
Vector	$O(\log n)$	$O(n)$
List	$O(n)$	$O(1)$

Comparisons

	Search	Insertion
Vector	$O(\log n)$	$O(n)$
List	$O(n)$	$O(1)$
Skip List	$O(\log n)$	$O(\log n)$

Thank You!