(8-2) Arrays II
H&K Chapter 7

Instructor – Beiyu Lin

CptS 121 (May 28th, 2019)

Washington State University

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Array Searching (1)

- Lots of motivating problems
  - Find a name in the phone book
  - Amazon automated warehouse

    https://www.youtube.com/watch?v=Ox05Bks2Q3s
  - Others: https://www.youtube.com/watch?v=TvRbxGoJrUU

Graph is from: https://www.engineering.com/DesignerEdge/DesignerEdgeArticles/ArticleID/6880/Amazons-Robotic-Order-Fulfillment.aspx

C. Hundhausen, A. O'Fallon, B. Lin

Array Searching (2)

- Sequential Search Algorithm
  - Solution strategy: search list sequentially but halt if not found

```
Get the value of target and the list of names N1, N2, . . . Ni
Set the value of num to 1
Set the value of found to false
While (num <= i) and (found = false)
  if target = Nnum
    set found to true
  else
    set num to num + 1
Endwhile
```

Pseudo code was also written on the white board in the class.

C. Hundhausen, A. O'Fallon, B. Lin

Array Searching (3)

- C adaptation of sequential search

```
int sequential_search(int values[],int target, int num_values)
{
  int index, found;
  index = 0;
  found = 0; /* false */
  while (index < num_values && !found)
  {
          if (values[index] == target){found = 1;}
          else {index++;}
  }
  /* return index at which target was found */
  if (found){return (index);}
  else {return NOT_FOUND; /* #defined to -1 */}
}
```


C. Hundhausen, A. O'Fallon, B. Lin

Array Searching (4)

Binary Search

Real life example: **https://www.youtube.com/watch?v=YgVNJ2v9IPA**

- *Input*: a list of n ***sorted*** values and a target value
- *Output*: True if target value exists in list and location of target value, false otherwise
- *Method*:
  - Set left to 1 and right to n
  - Set found to false
  - While found is false and left is less than or equal to right
    - Set mid to midpoint between left and right
    - If target = item at mid then set found to true
    - If target < item then set right to mid – 1
    - If target > item then set to left to mid + 1
  - If found = true then print "Target found at location mid"
  - Else print "Sorry, target value could not be found."

Steps were also drawn in the whiteboard during the class.

C. Hundhausen, A. O'Fallon, B. Lin

Visualization of Binary Search (1)

Let's say we're searching for the value 33. Initially, we set L to 0, R to 26, and found to false:

L																												R
12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26		

Since found is false and L <= R, we set mid to (L + R) / 2:

L													M															R
12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26		

A simple example was written on the whiteboard during the class.

C. Hundhausen, A. O'Fallon, B. Lin

Visualization of Binary Search (2)

Since target (33) < array[mid] (61), we set R to mid - 1:

L R M

12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Since found is false and L <= R, we set mid to (L + R) / 2:

M R

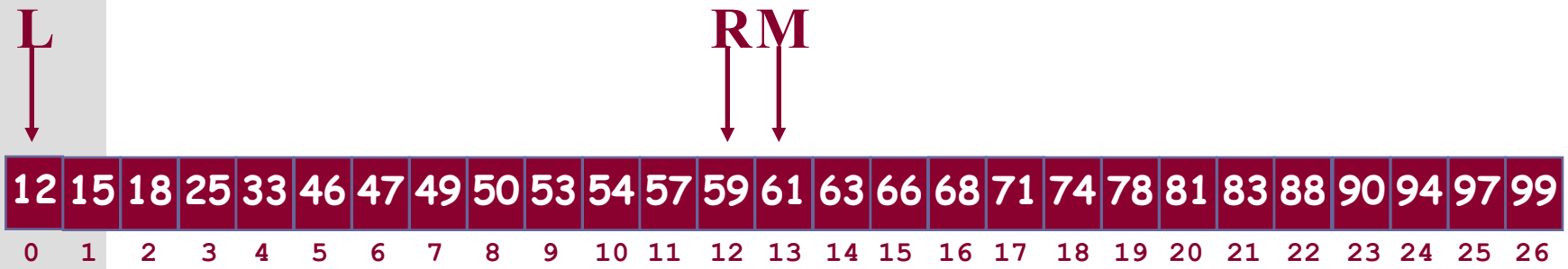
L

12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

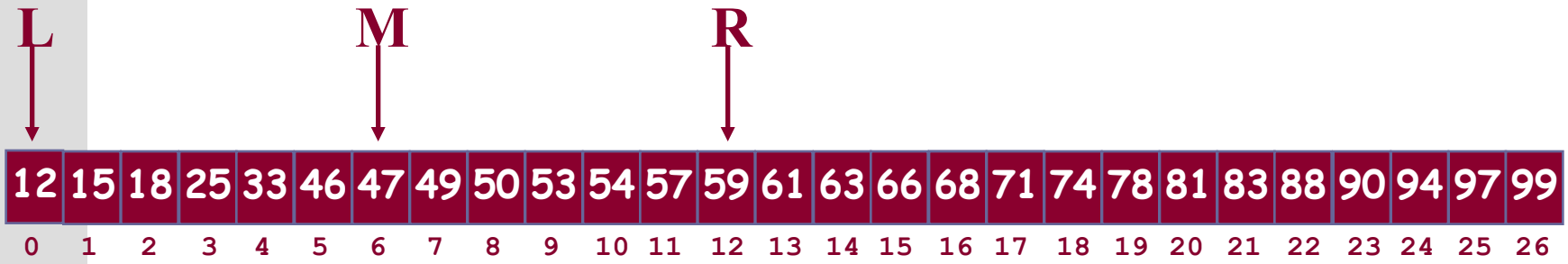
C. Hundhausen, A. O'Fallon

Visualization of Binary Search (3)

Since target (33) < array[mid] (47), we set R to mid - 1:



Since found is false and L <= R, we set mid to (L + R) / 2:



C. Hundhausen, A. O'Fallon

Visualization of Binary Search (4)

Since target (33) < array[mid] (47), we set R to mid - 1:

L					R	M																				
12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Since found is false and L <= R, we set mid to (L + R) / 2:

L		M			R																					
12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

C. Hundhausen, A. O'Fallon

Visualization of Binary Search (5)

Since target (33) > array[mid] (18), we set L to mid + 1:

M L R

12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Since found is false and L <= R, we set mid to (L + R) / 2:

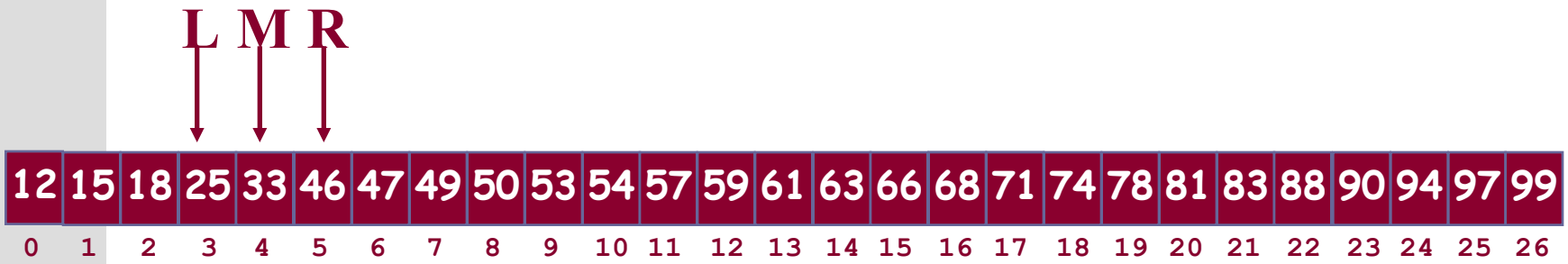
L M R

12	15	18	25	33	46	47	49	50	53	54	57	59	61	63	66	68	71	74	78	81	83	88	90	94	97	99
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

C. Hundhausen, A. O'Fallon

Visualization of Binary Search (6)

Since target (33) == array[mid] (33), we set found to true



Since found is true, we return the index of mid (4)

C. Hundhausen, A. O'Fallon

Array Searching (5)

- C adaptation of binary search (compiled and tested!)

```
int binary_search(int values[], int target, int num_values)
{
        int found, left, right, mid;
        found = 0; /* false */
        left = 0;
        right = num_values;
        while (!found && left <= right)
        {
          mid = (left + right)/2;
          if (target == values[mid])
            found = 1; /* true */
          else if (target < values[mid])
            right = mid - 1;
          else /* target > values[mid] */
            left = mid + 1;
        }
        if (found)
          return (mid);
        else
          return (NOT_FOUND); /* #defined to -1 */
}
```


C. Hundhausen, A. O'Fallon

Array Sorting (1)

- Lots of motivating problems
  - Print out a class list in alphabetical order
  - Order finish times in a race to determine who placed
  - Make look-up easier (preparation for binary search)

C. Hundhausen, A. O'Fallon

Array Sorting (2)

- Selection Sort
  - *Input*: a list of numbers
  - *Output*: a list of the same numbers in ascending order
  - *Method*:
    - Set marker that divides “sorted” and “unsorted” sections of list to the beginning of the list
    - While the unsorted section of the list is not empty
      - Call upon Find_Smallest helper function to find index of smallest value in “unsorted” section of list
      - Swap smallest value in "unsorted section of list" with first value in “unsorted” section of list
      - Move "sorted/unsorted" marker left one position

Pseudo code and graphs were drawn on the white board in the class.

C. Hundhausen, A. O’Fallon, B. Lin

Array Sorting (3)

- C adaptation of selection sort (compiled and tested!)

```
void selection_sort(int values[], int num_values)
{
  int i, index_of_smallest, temp;
  for (i = 0; i < num_values - 1; ++i)
  {
    /* Find the index of the smallest element in unsorted list... */
    index_of_smallest = find_smallest(values,i,num_values-1);
    /* Swap the smallest value in the subarray i+1 .. num_values - 1
        with the value i, thereby putting into place the ith element. */
    temp = values[i];
    values[i] = values[index_of_smallest];
    values[index_of_smallest] = temp;
  }
}

int find_smallest(int values[], int low, int high)
{
  int smallest_index, i;
  smallest_index = low;
  for (i = low + 1; i <= high; i++)
    if (values[i] < values[smallest_index])
      smallest_index = i;
  return smallest_index;
}
```


C. Hundhausen, A. O'Fallon

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016
- P.J. Deitel & H.M. Deitel, *C How to Program (7th Ed.)*, Pearson Education , Inc., 2013.

Collaborators

- Chris Hundhausen
- Andrew O'Fallon