Data Structures and Algorithms

SEARCHING ALGORITHMS

Nguyễn Ngọc Thảo
nnthao@fit.hcmus.edu.vn

Ho Chi Minh City, 05/2022

Outline

- The searching problem
- Linear search
    - Linear search without a sentinel
    - Linear search with a sentinel
- Binary search

The searching problem

- Searching for a keyword, a value, or a specific piece of data (information) is the basis of many computing applications.

Find your car in a parking lot

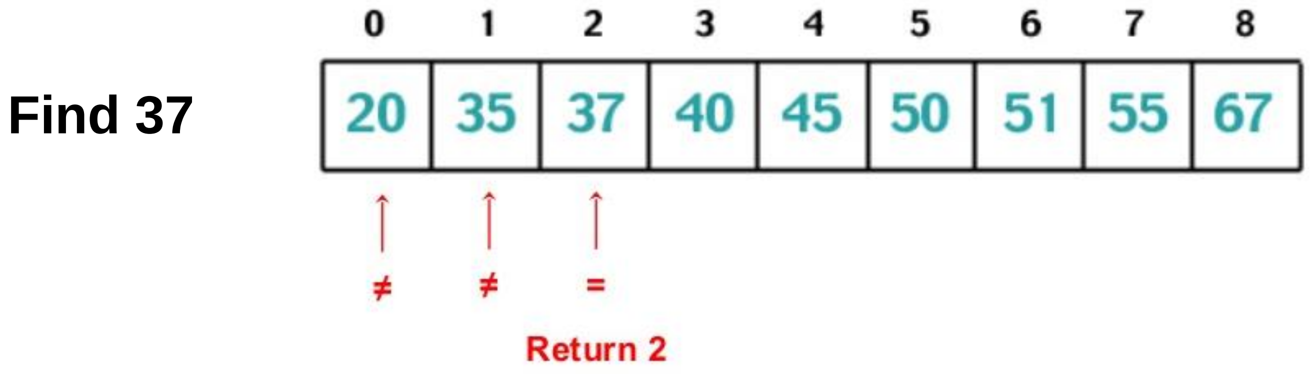
Search for a specific book in the library

Look for information on the Internet

Retrieve a document in a series of documents

The searching algorithms

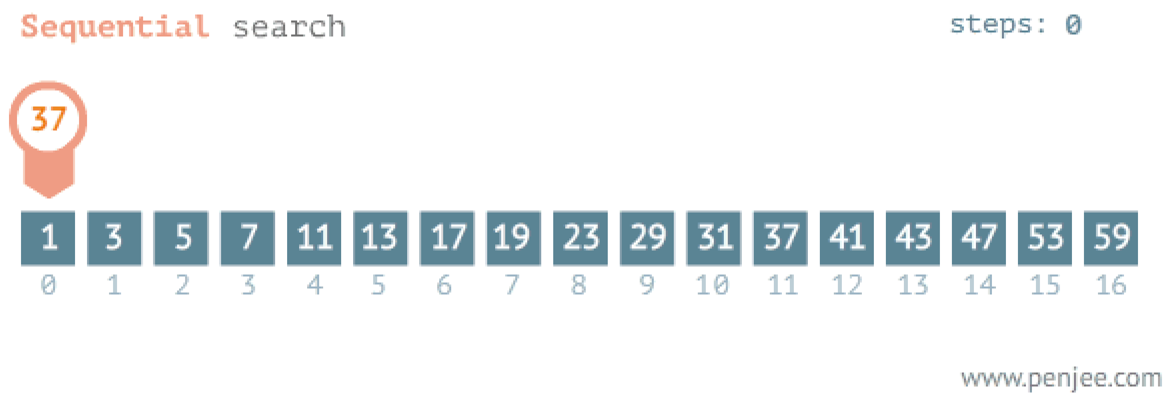
- The searching algorithms are designed to check for an element or retrieve an element from any data structure.
  - For example, finding a value on a given array is to locate the index of the first element that is equal to the value considered.



- These algorithms are categorized into: Sequential search and Interval search.

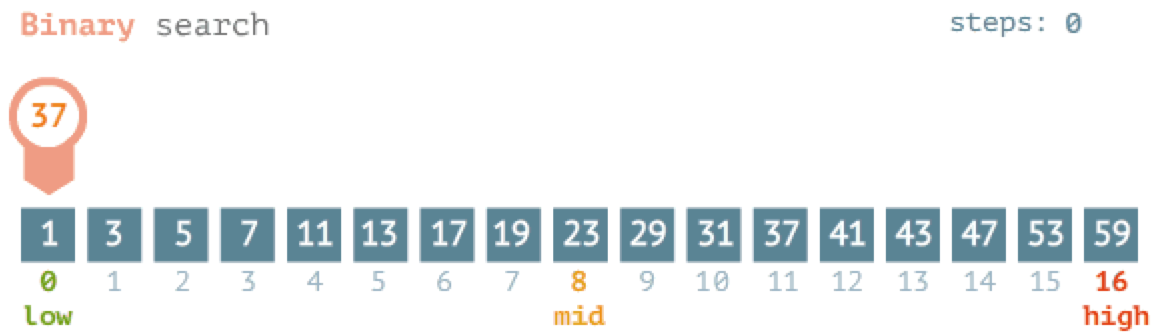
Sequential search

- Key idea: The list of elements is traversed sequentially, and every element is checked.
- Representative: Linear search $O(n)$



Interval search

- Key idea: The center(s) of the search structure are targeted, and the search space is split into parts after each iteration.
  - Specifically designed for searching in sorted data-structures
  - Much more efficient than Sequential search
- Representative: Binary search $O(\log_2 n)$



Linear search

- *Linear search without a sentinel*
- *Linear search with a sentinel*

Linear search without a sentinel

- Given an array of n elements and an element x to search for.
- Start from the first element of the array and one by one compare x with each element of the array
    - If x matches with an element, return the index.
    - If x does not match with any of elements, return -1.

```
Input: find x = 5 in {1, 25, 5, 2, 37, 40}

Output: 2 (x is present at index 2)
```

```
Input: find x = 6 in {1, 25, 5, 2, 37, 40}

Output: -1 (x is not present in the array)
```

Linear search without a sentinel

- Step 1. Set the increment variable $i = 1$
- Step 2. Compare the element $a[i]$ with x
  - If $a[i] = x$ then **return** $\boldsymbol{i}$ (x is present at index i)
  - Otherwise, increase i by 1 and go to **Step 3**
- Step 3. Check whether the end of the array is reached by comparing i with n
  - If $i \leq n$ then go to **Step 2** (i.e., still within the array)
  - Otherwise, **return -1** (x is not in the array)

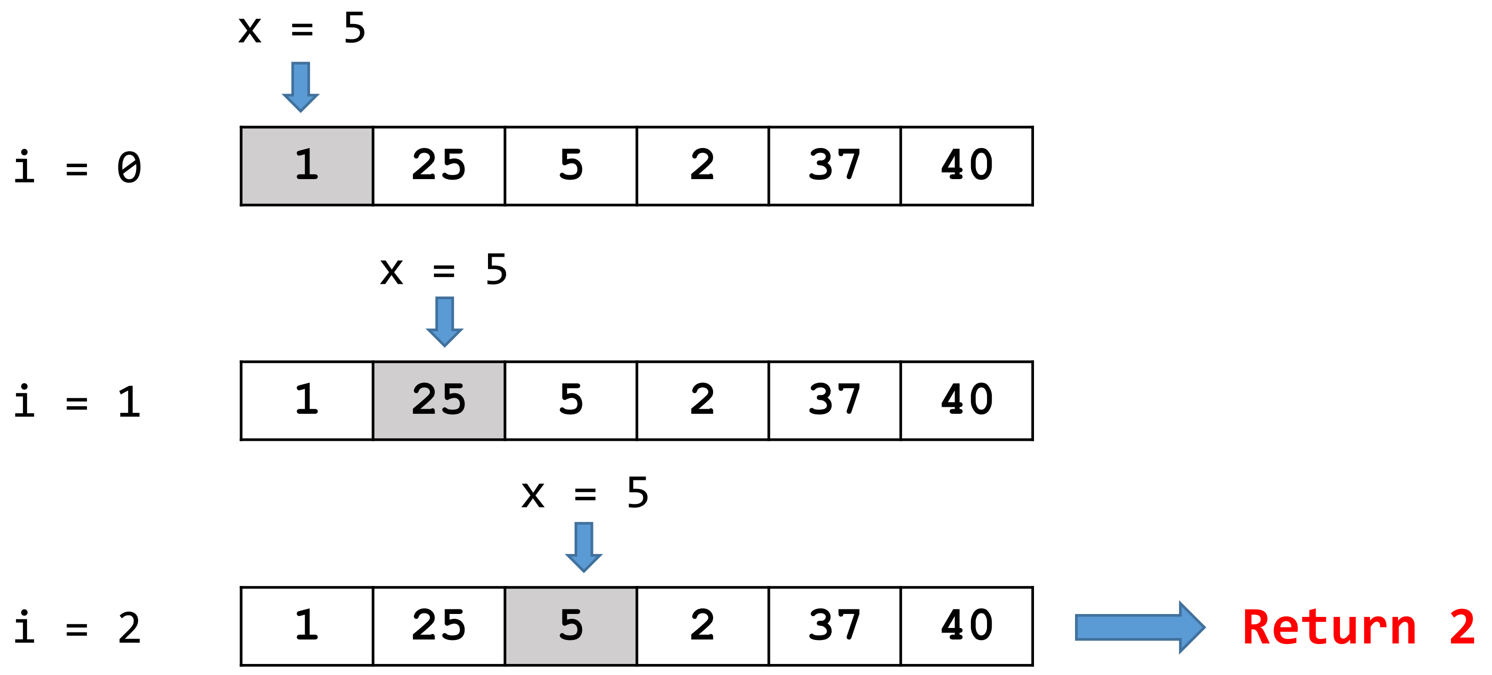
Linear search without a sentinel

- The function **`linearSearch`** sequentially (and exhaustively) searches an integer x on an array of unordered integers.

```
int linearSearch(int arr[], int n, int x){
   for(int index = 0; index < n; ++index){
      if (arr[index] == x)
         return index;
   }
   return -1;
}
```

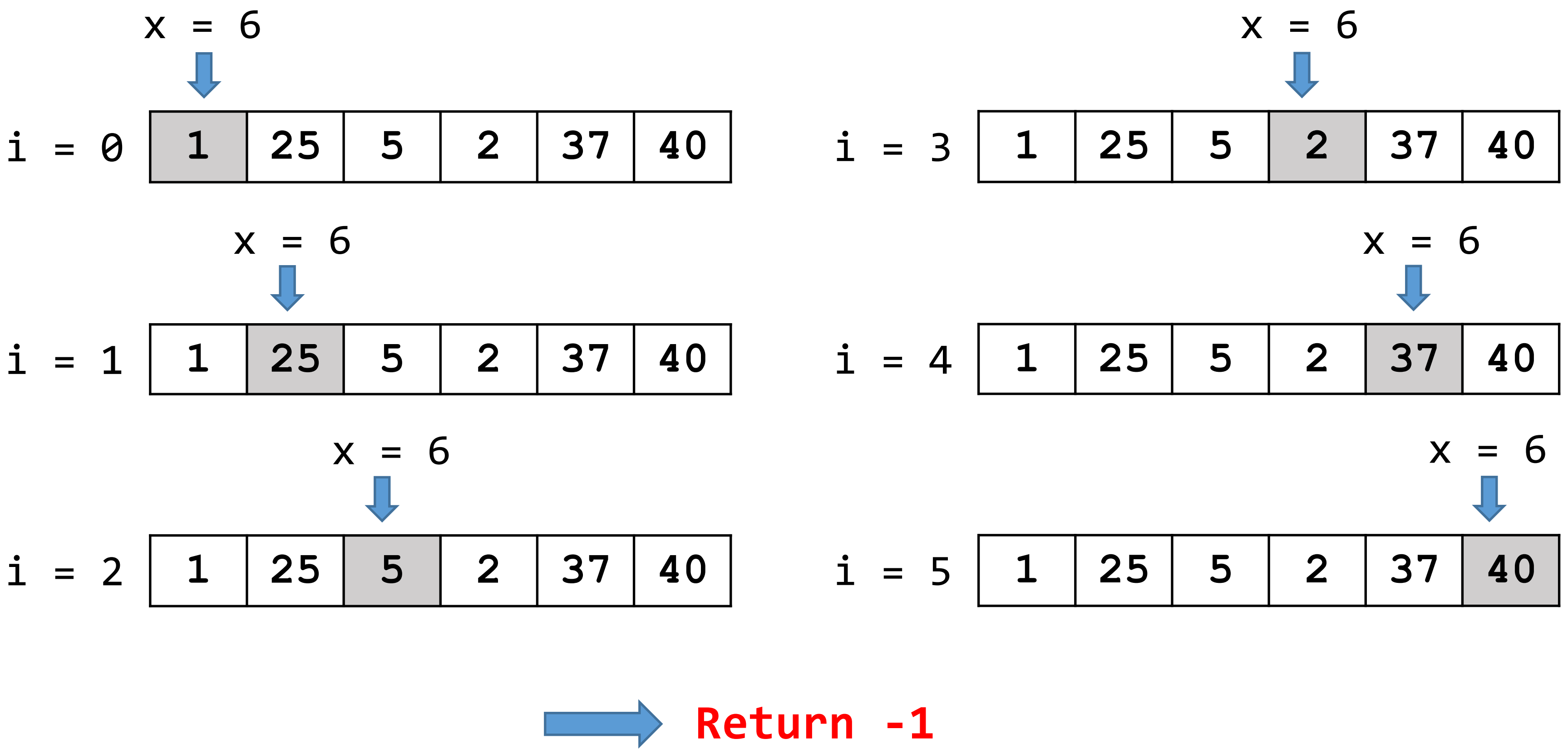
Linear search without a sentinel

- Example 1: find $x = 5$ in {1, 25, 5, 2, 37, 40}



Linear search without a sentinel

- Example 2: find $x = 6$ in {1, 25, 5, 2, 37, 40}



Linear search without a sentinel

Best case	Worst case	Average case
$O(1)$	$O(n)$	$O(n)$

- Comparison is the basic operation in search algorithms.
  - The number of operations is proportional to the number of elements.
- Analysis of the average case
  - Assume that the data has uniform distribution, i.e., the probabilities of observing x at any location are all the same.
  - There are 2 comparisons for each iteration. Thus,

$$2 \times (1 + 2 + .. + n)/n = n + 1$$

Linear search with a sentinel

- Replace the last element of the array with the search element x (sentinel) and run a while loop until x is found

- Step 1. Set $last = a[n]$ and $a[n] = x$
- Step 2. Set the increment variable $i = 1$
- Step 3. Compare the element $a[i]$ with x
  - If $a[i] = x$ then go to **Step 4**
  - Otherwise, increase i by 1 then repeat **Step 3**
- Step 4. Set $a[n] = last$.
  - If $i < n$ or $a[n] = x$ then **return i** (x is present at index i)
  - Otherwise, **return -1** (x is not in the array)

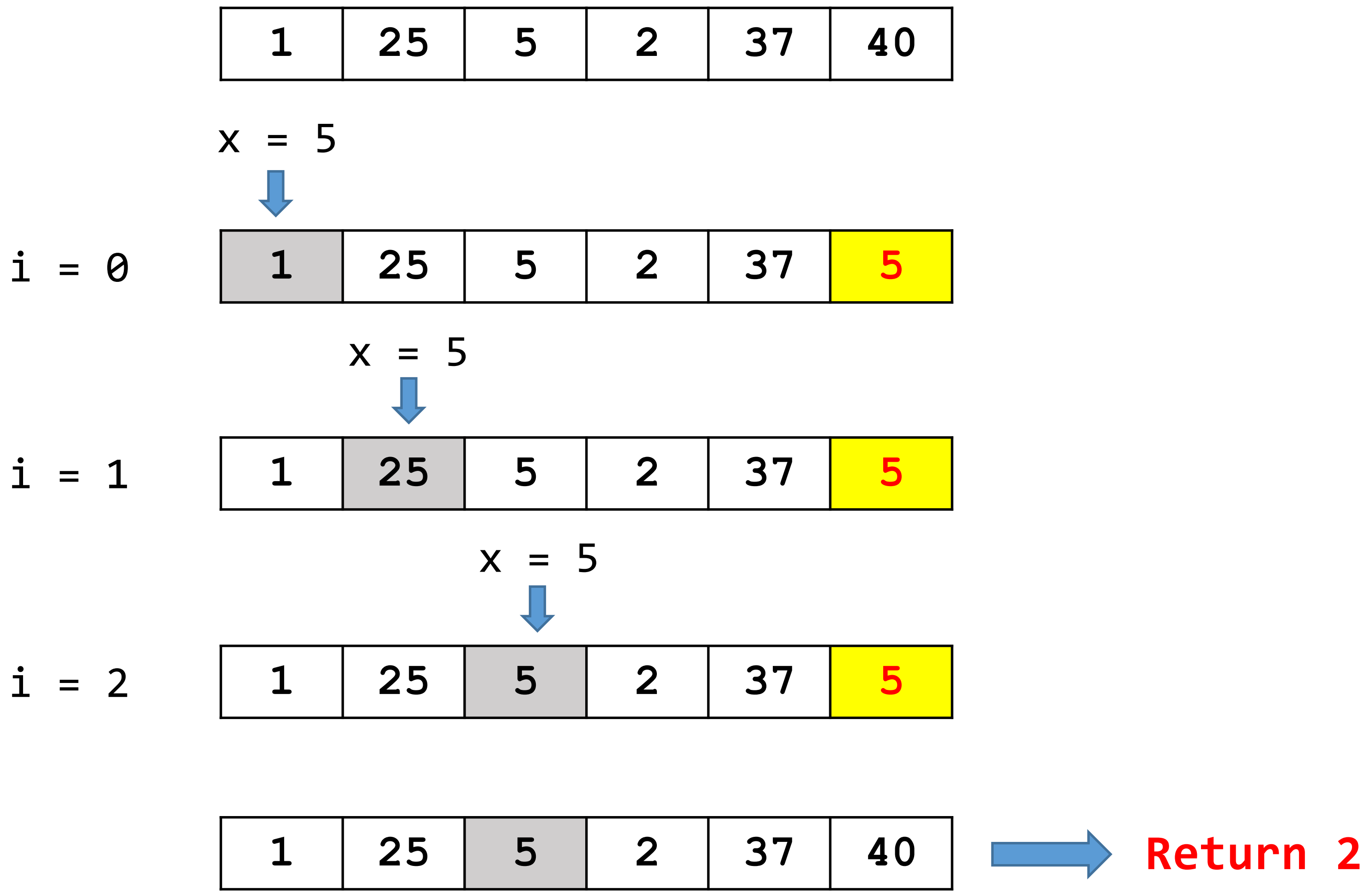
Linear search with a sentinel

- The function **linearSearchSentinel** searches the array without checking whether the current index is within the array.

```
int linearSearchSentinel(int arr[], int n, int x){
   int last = arr[n - 1];      // Last element of the array
   arr[n - 1] = x;          // The search element is the end
   int i = 0;
   while (arr[i] != x)
      i++;
   arr[n - 1] = last;          // Put the last element back
   if ((i < n - 1) || (x == arr[n - 1]))
      return i;
   return -1;
}
```

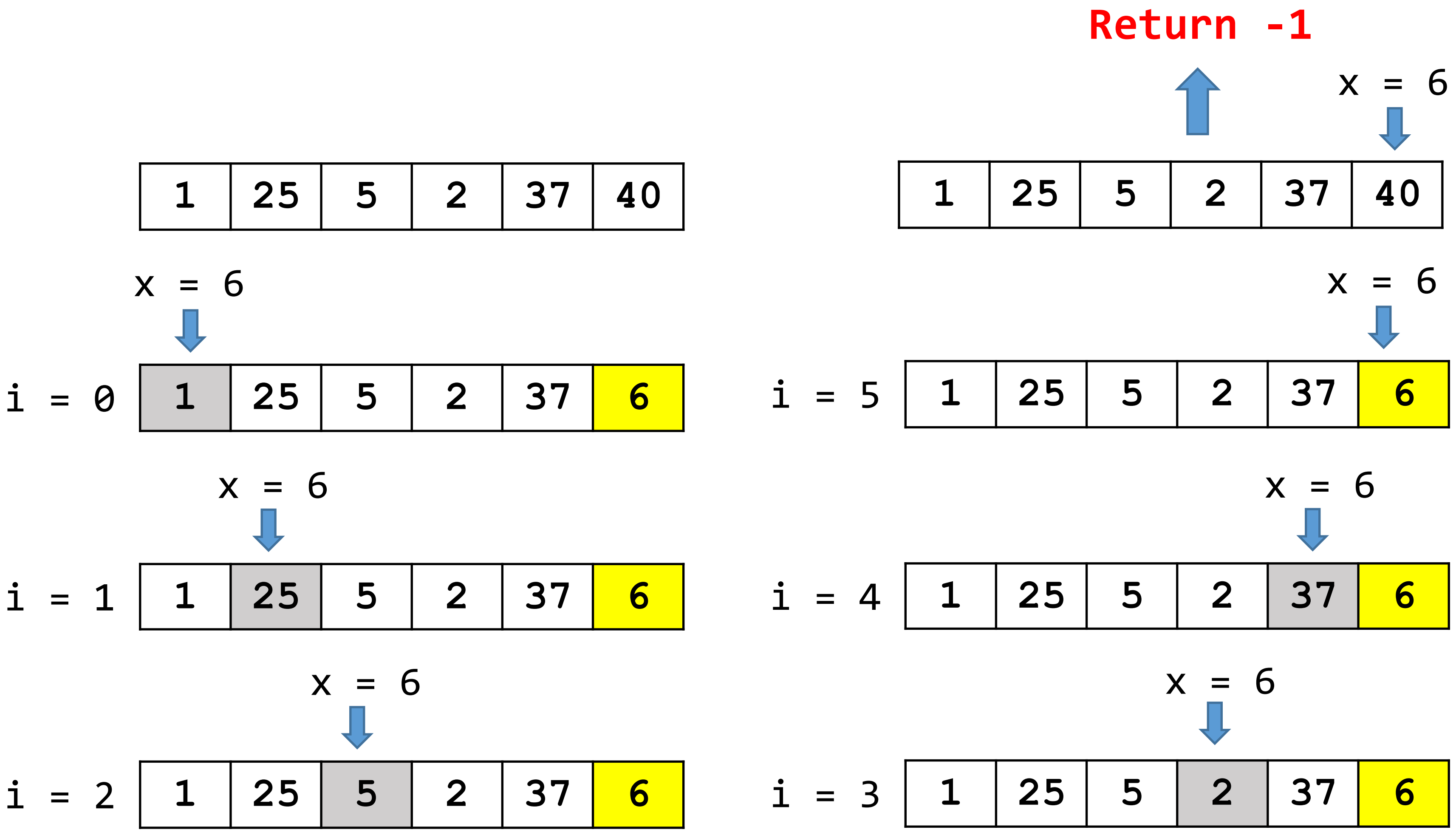
Linear search with a sentinel

- Example 1: find $x = 5$ in {1, 25, 5, 2, 37, 40}



Linear search with a sentinel

- Example 2: find $x = 6$ in {1, 25, 5, 2, 37, 40}



An analysis of linear search

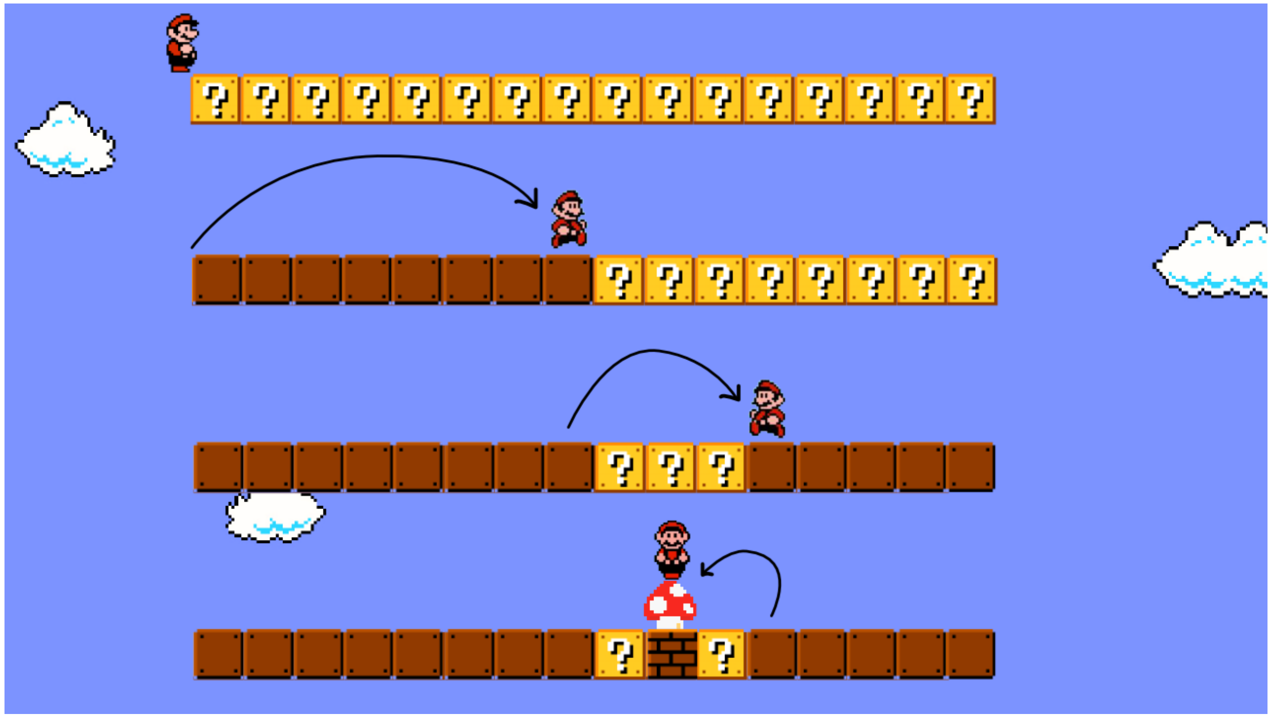
- Linear search is rarely used in practice.
    - Binary search and hash tables allow significantly faster searching.
- The time complexity is not affected by the data ordering.
- For large n, the use of sentinel offers lower running time.
    - 20% faster with $n = 15{,}000$

Checkpoint 01: Linear search with sorted data

Consider a sequential search of n data items.

If the data items are sorted into ascending order, how can you determine that your desired item is not in the data collection without always making n comparisons?

Binary search



Binary search algorithm

- Given an array of n elements and an element x to search for.
- Divide and conquer: Search a sorted array for a particular item by successively dividing the array in half
  - Begin with an interval covering the whole array
  - If x is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half.
  - Repeatedly check until x is found, or the interval is empty.

Binary search algorithm

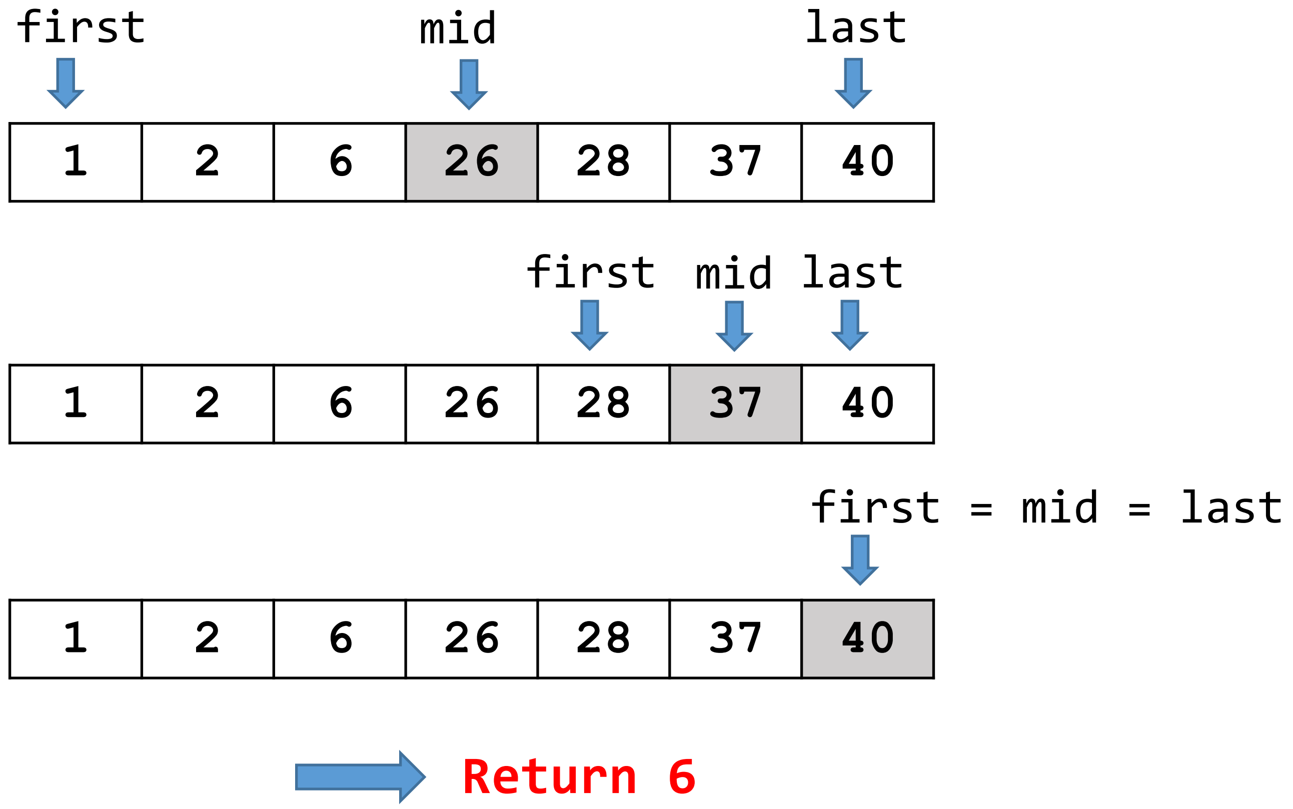
- Step 1. Set $first = 1$ and $last = n$
- Step 2. Check whether the interval is empty by comparing $first$ and $last$
  - If $first \leq last$ then go to **Step 3**
  - Otherwise, **return -1** (x is not in the array)
- Step 3. Set $mid = \lfloor (first + last)/2 \rfloor$. Compare the element $a[mid]$ with x
  - If $x = a[mid]$ then **return $\boldsymbol{i}$** (x is present at index i)
  - If $x < a[mid]$ then set $last = mid - 1$
  - Otherwise, set $first = mid + 1$
  - Go to **Step 2**

Binary search implementation

```
int binarySearch(int arr[], int n, int x){
   int first, middle, last;
   first = 0; last = n - 1;
   while (first <= last) {
      middle = (first + last) / 2;
      if (arr[middle] == x)
         return middle;
      if (arr[middle] > x)
         last = middle - 1;
      else
         first = middle + 1;
   }
   return -1;
}
```

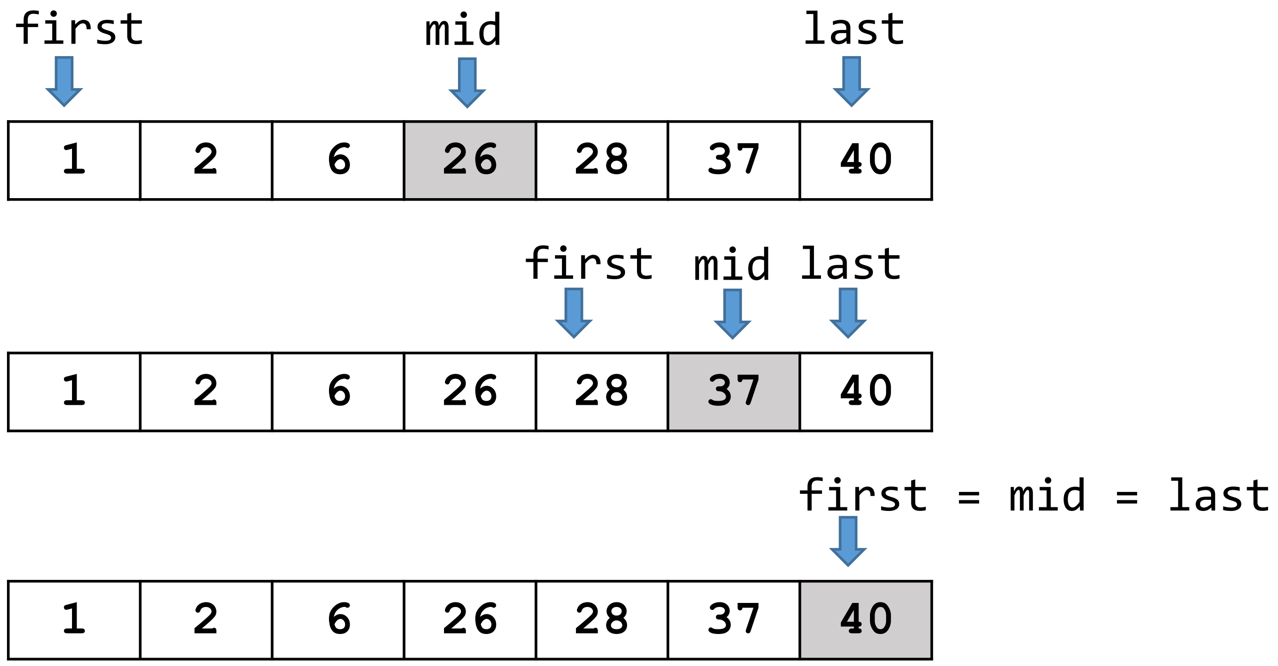
Example 1: x is found

- Find $x = 40$ in $\{1, 2, 6, 26, 28, 37, 40\}$



Example 2: x is not found

- Find $x = 47$ in $\{1, 2, 6, 26, 28, 37, 40\}$



`Next step: the while loop stops because first = n > last = n-1`

Return -1

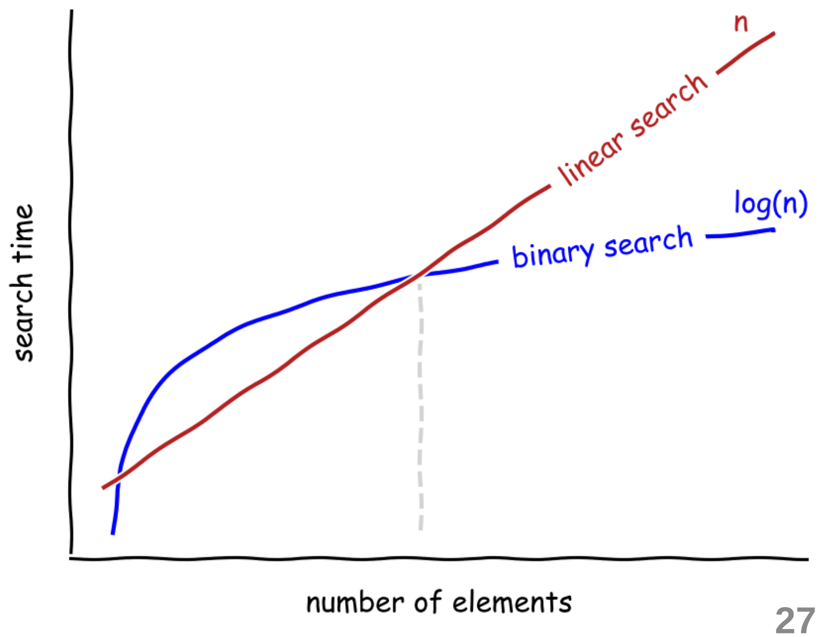
An analysis of binary search

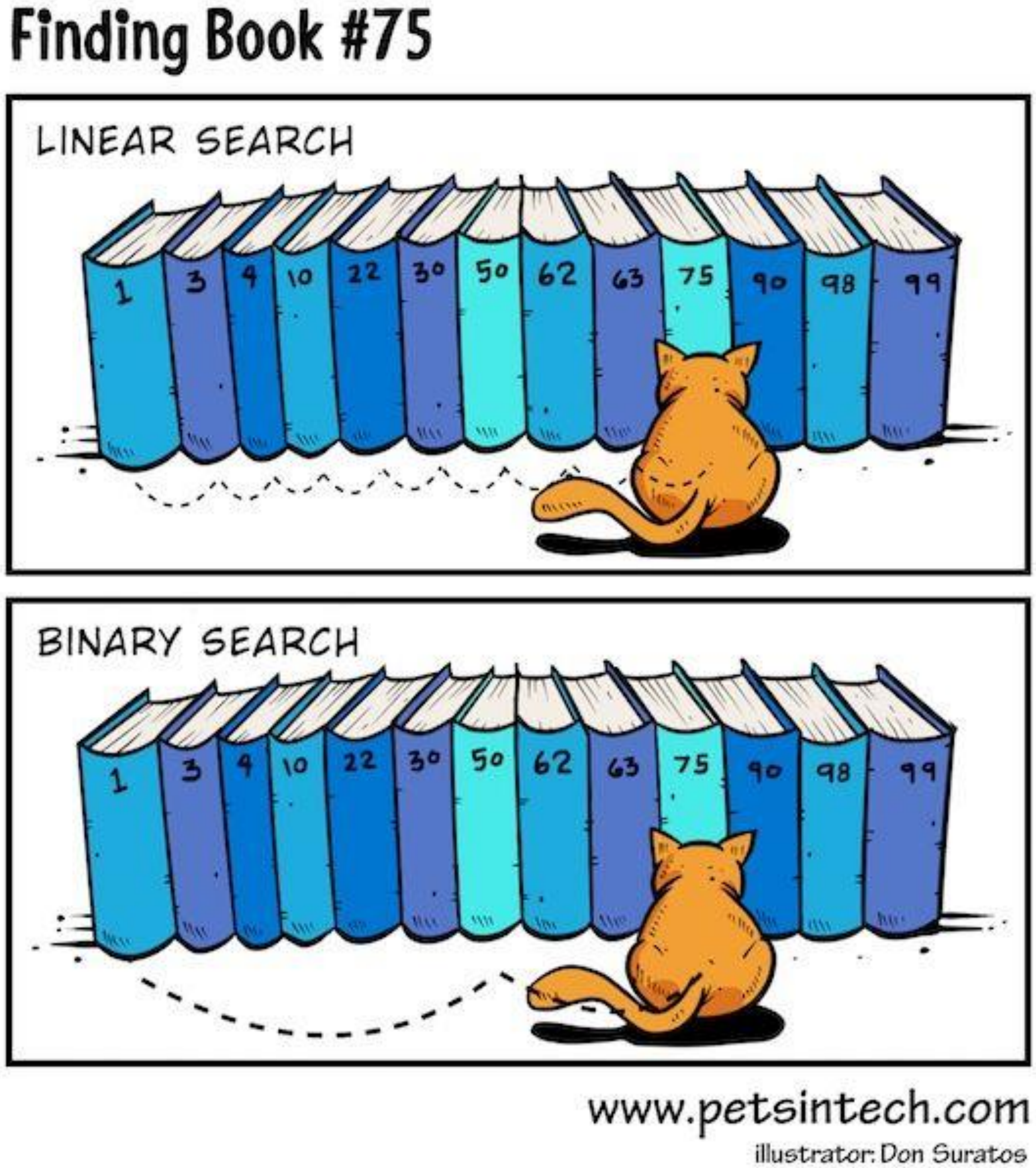
Best case	Worst case	Average case
$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$

- The algorithm still requires at most k divisions to obtain a subarray with one item.
- For $n = 2^k$: It is obvious that $k = \log_2 n$
- For $n \neq 2^k$: Find the smallest k such that $2^{k-1} < n < 2^k$. That is, $k = 1 + \lfloor \log_2 n \rfloor$.

An analysis of binary search

- For large arrays, the binary search has an enormous advantage over a sequential search.
    - E.g., $n = 1{,}000{,}000$, a binary search needs at most 20 comparisons, but a linear search of the same items requires 1 million comparisons.
- *Maintaining the array in sorted order requires an overhead cost, which can be substantial.*




Finding Book #75
LINEAR SEARCH
1 3 4 10 22 30 50 62 63 75 90 98 99
BINARY SEARCH
1 3 4 10 22 30 50 62 63 75 90 98 99
www.petsintech.com
illustrator: Don Suratos

Checkpoint 02: Run binary search on a sorted array

You are using binary search to look for a number x in the array

{5, 11, 25, 28, 45, 78, 100, 120, 125}

When the algorithm stops, what are the values of the variables first and last, respectively? (Indices are from 0)

x can be one of the following cases:

a) 45
b) 100
c) 30
d) 72

Acknowledgements

- This part of the lecture is adapted from

  [1] Pr. Nguyen Thanh Phuong (2020) "*Lecture notes of CS163 – Data structures*" University of Science - Vietnam National University HCMC.

  [2] Pr. Van Chi Nam (2019) "*Lecture notes of CSC14004 – Data structures and algorithms*" University of Science - Vietnam National University HCMC.

  [3] Frank M. Carrano, Robert Veroff, Paul Helman (2014) "*Data Abstraction and Problem Solving with C++: Walls and Mirrors*" Sixth Edition, Addion-Wesley. **Chapter 10.**

Exercises

01. Binary search algorithm

a) Find $x = 8$ in {3, 5, 6, 8, 11, 12, 14, 15, 17, 18}. Which of the following is the sequence of elements that binary search compares with x?

  i. 12, 6, 11, 8

  ii. 11, 5, 6, 8

  iii. 3, 5, 6, 8

  iv. 18, 12, 6, 8

b) Find $x = 16$ in {3, 5, 6, 8, 11, 12, 14, 15, 17, 18}. What is the sequence of elements that binary search compares with x?

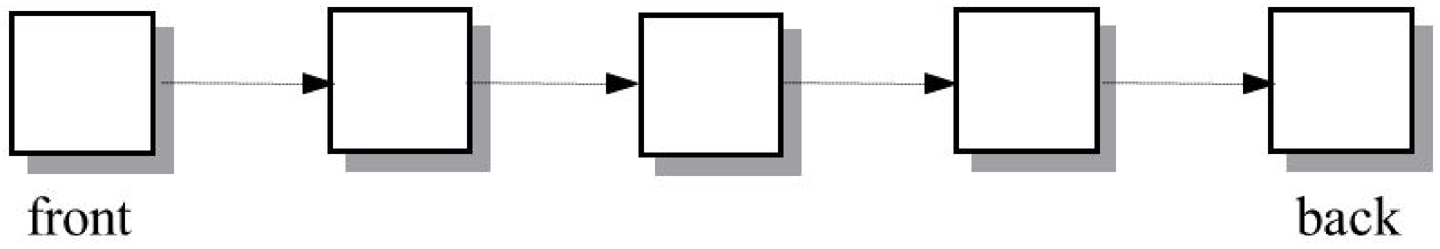
02. Binary search case studies

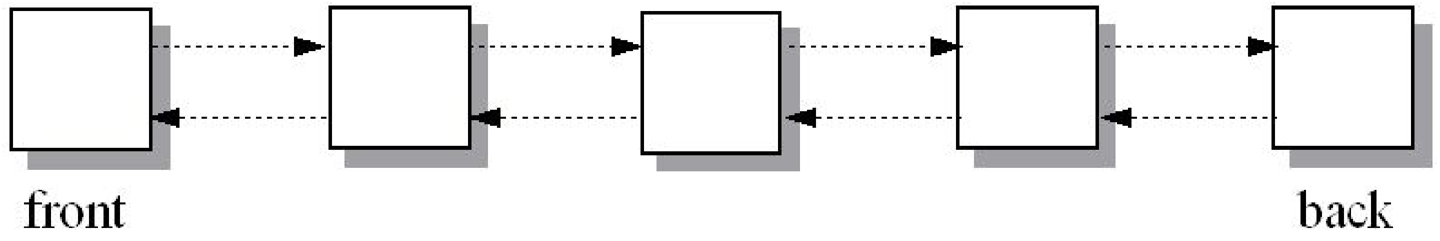
a) 32 teams qualified for the 2018 World Cup. If the names of the teams were sorted alphabetically in an array, at most how many items the binary search would have to examine to find the location of a particular team in the array?

b) In 2013, there were 193 member states in the United Nations. If the names of these states were sorted alphabetically in an array, at most how many items the binary search would have to examine to locate a particular name in the array?

03. Binary search on linked lists

- Identify the feasibility of binary search on singly linked lists, doubly linked lists and circular (singly/doubly) linked lists.





04. “Guess-the-number”

- Design an algorithm for the program “guess-the-number”, which is described as follows
  - The user thinks of a positive number k in the range of [1, 10000].
  - The program guesses a value i and the user tells whether i is larger or smaller than k.
  - The process continues until k is correctly guessed, i.e., $k = i$.