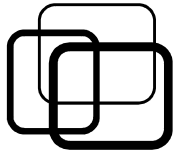
Search Algorithms

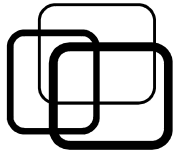
Inst. Nguyễn Minh Huy

Contents



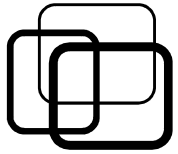
- Linear search.
- Binary search.

Contents

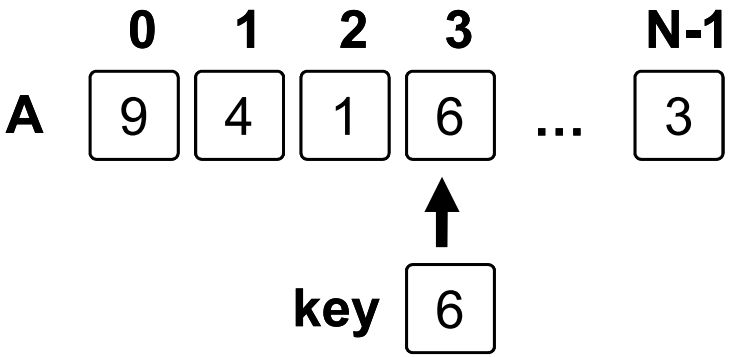


- **Linear search.**
- Binary search.

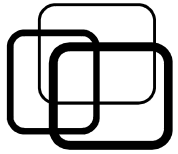
Linear search



- Searching problem:
  - Given array A of N elements.
  - Given a key.
  - Find key in A.
    - position (if found).
    - -1 (not found).



Linear search

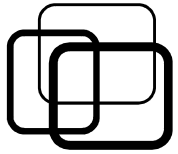


- Linear search algorithm:
  - Loop version:

```
linearSearch( array A, size N, key ) {
    for each element in A
        if  element = key
            return element position;
    return -1;
}
```

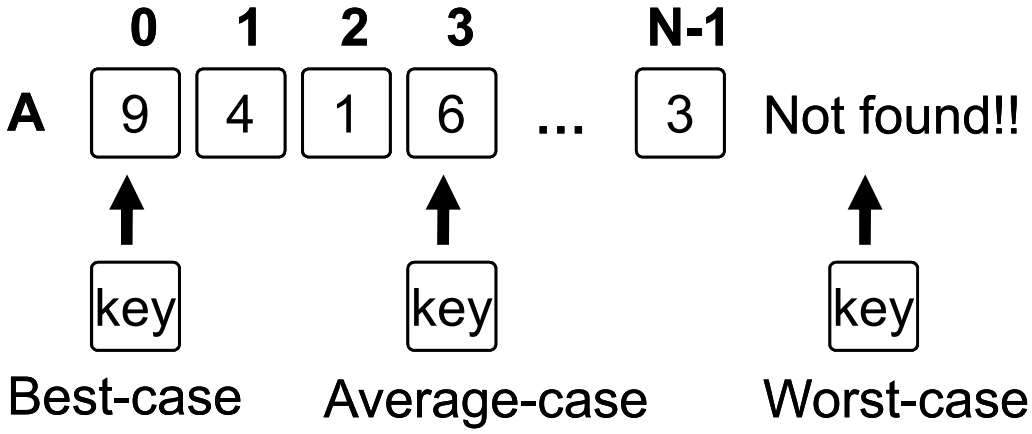
  - Recursive version:
    - Use divide-conquered technique.
      - Split at middle.
      - Split at last element.

Linear search



- Linear search analysis:

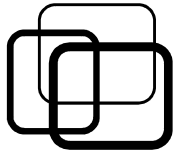
Scenario	When occur?	Complexity
Best-case	key is at the beginning	O(1)
Worst-case	key is not found	O(n)
Average-case	key is at somewhere in the middle	O(n)



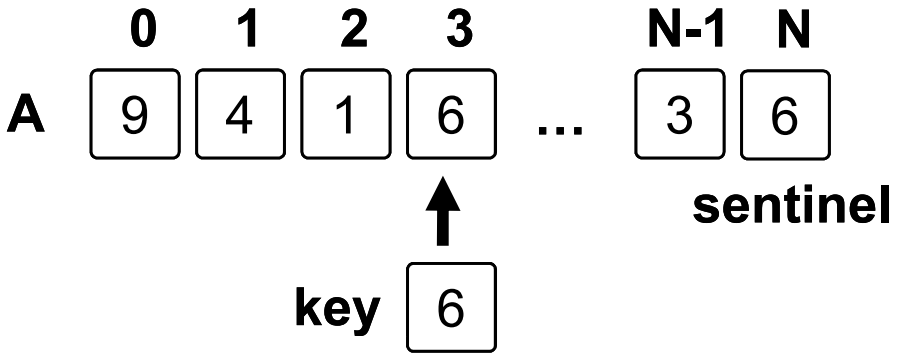
➔ Simple to implement.

➔ Fast to search **medium-sized**, **un-ordered** array.

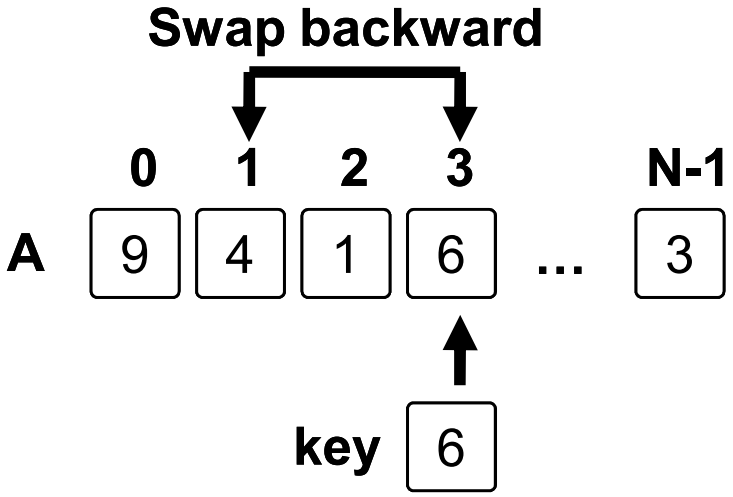
Linear search



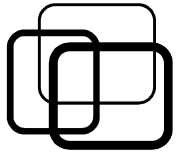
- Linear search improvement:
  - Adding sentinel:
    - Add key to array end.
    - Reduce loop condition.



  - Bubble search:
    - Some items are searched more frequently.
    - Found item should be “bubbled up”.
    - 80-20 rule: 30% faster.



Linear search

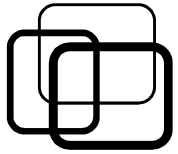


- Linear search improvement:
  - Bubble search:

```
bubbleSearch( array A, size N, key ) {
    for each element in A
        if  element = key
            swap with backward element;
            return element index;
    return -1;
}
```

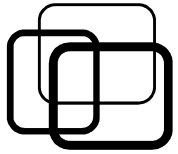
➔ How far to move backward?

Contents



- Linear search.
- **Binary search.**

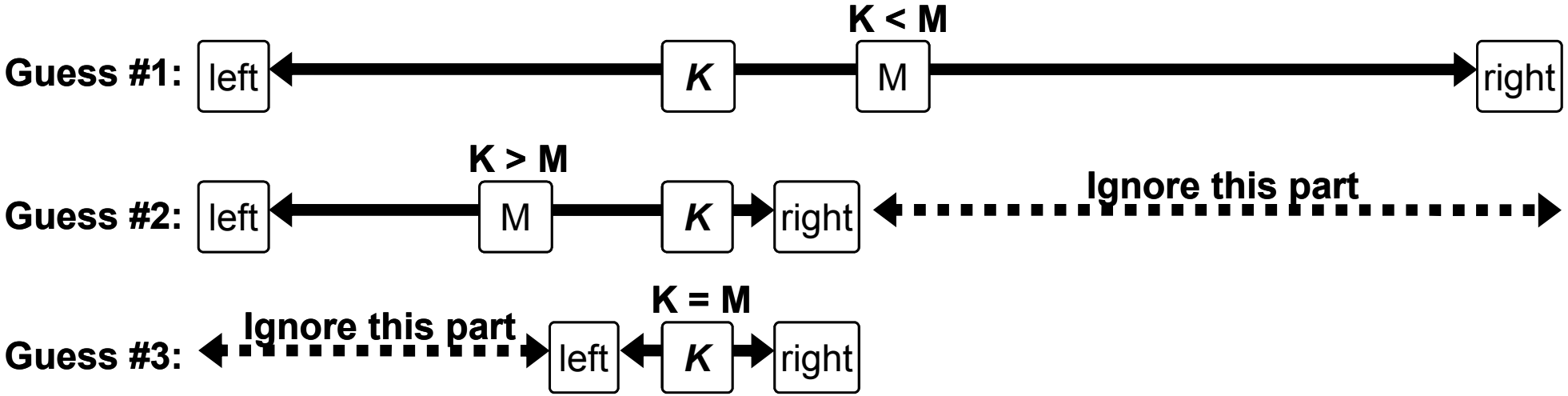
Binary search



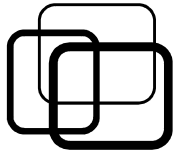
- Binary search idea:
  - Two players A, B play the “number guessing” game:
    - A: think of an integer K (1 <= K<= 100).
    - B: guess an integer M.
    - A: tell if M is less than, greater than, or equals to K.

      B continues to make guess until M equals to K.

  ➔ Find a strategy for B to make the least guesses!!



Binary search



- Binary search algorithm:
  - Recursive version:

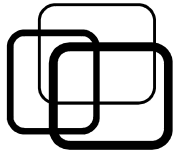
```
// A is SORTED in ascending order.
binarySearch( array A, from l, to r, key ) {
    if ( zero-sized array ) return -1;

    mid = mid-point from l to r;

    if ( key < A[ mid ] )
        return binarySearch( A, l, mid – 1, key );
    if ( key > A[ mid ] )
        return binarySearch( A, mid + 1, r, key );
    return mid;
}
```

  - Loop version??

Binary search



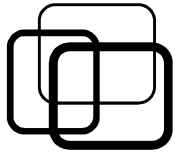
- Binary search analysis:

Scenario	When occur?	Complexity
Best-case	key is at the first mid-point	O(1)
Worst-case	key is not found	O(log(n))
Average-case	key is somewhere in the array	O(log(n))

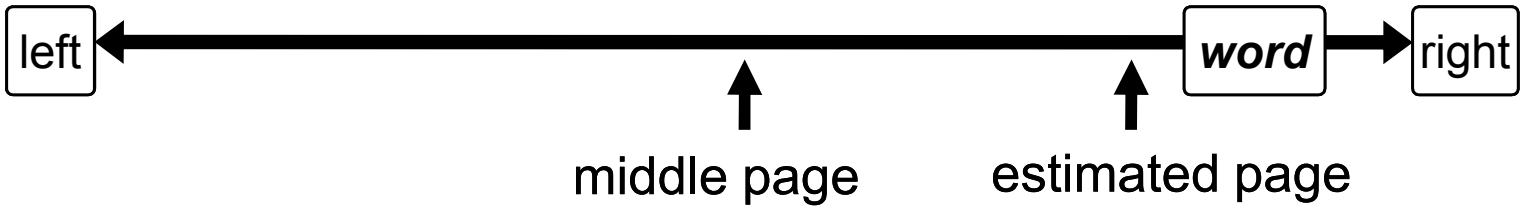
n	Linear search comparisons	Binary search comparisons
10	~ 10	~ 4
1,000	~ 1,000	~ 10
1,000,000	~ 1,000,000	~ 20
1,000,000,000	~ 1,000,000,000	~ 30

➔ Efficient way to search **LARGE, SORTED** array.

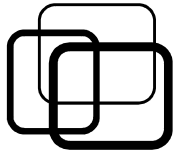
Binary search



- Binary search improvement:
  - Interpolation search:
    - How do you look up word in dictionary?
    - Look up the word ***"Submarine"***.
    - Do you first guess at the middle page?



Binary search



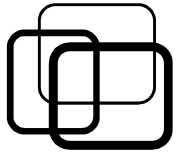
- Binary search improvement:
  - Interpolation search:

```
// A is SORTED in ascending order.
// A is UNIFORMLY DISTRIBUTED.
interpolationSearch( array A, from l, to r, key ) {
    if ( zero-sized array ) return -1;

    pos = estimated position;

    if ( key < A[ pos ] )
        return interpolationSearch( A, l, pos – 1, key );
    if ( key > A[ pos ] )
        return interpolationSearch( A, pos + 1, r, key );
    return mid;
}
```

Binary search



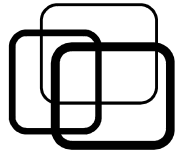
- Binary search improvement:
  - Interpolation search analysis:

Scenario	When occur?	Complexity
Best-case	key is found at the first guess	O(1)
Worst-case	key is not found	O(n)
Average-case	key is somewhere in the array	O(log(log(n)))

n	Binary search comparisons	Interpolation search comparisons
1,000,000	~ 20	~ 4
1,000,000,000	~ 30	~ 5
10^{18}	~ 60	~ 6
10^{97}	~ 330	~ 9

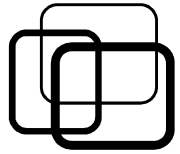
➔ Suitable for **VERY LARGE, SORTED, UNIFORM** array.

Summary



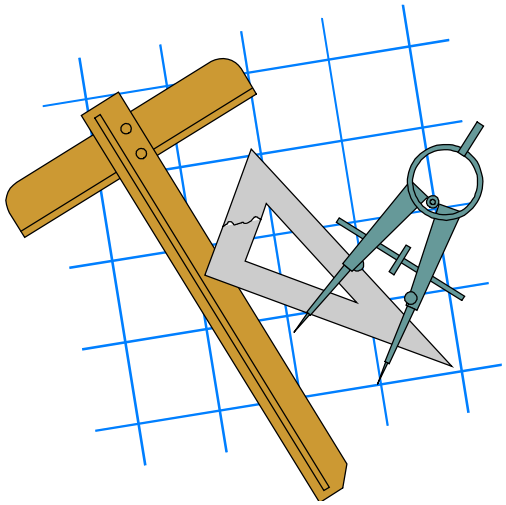
- Linear search:
  - For MEDIUM, UNSORTED array.
  - Average-case complexity: O(n).
  - Improvement: adding sentinel, bubble sort.
- Binary search:
  - For LARGE, SORTED array.
  - Average-case complexity: O(log(n)).
  - Improvement: interpolation search.

Practice

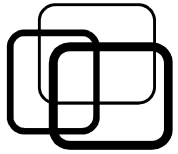


- Practice 3.1:

  Implement linear search (add sentinel) on **Singly Linked List**.



Practice

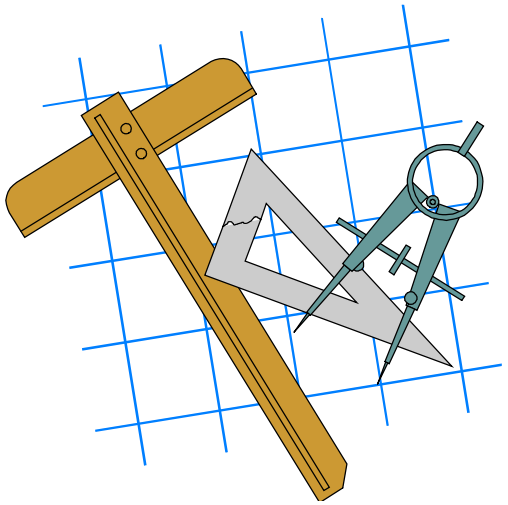


- Practice 3.2:

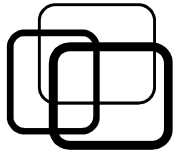
  Construct class **Dictionary** (array of words) and provide it with the following methods:

  - Initialize empty dictionary.
  - Add a word.
  - Sort all words in ascending order.
  - Count all words.

  - Linear search a word.
  - Linear search a word (add sentinel).
  - Binary search a word.
  - Interpolation search a word.



Practice

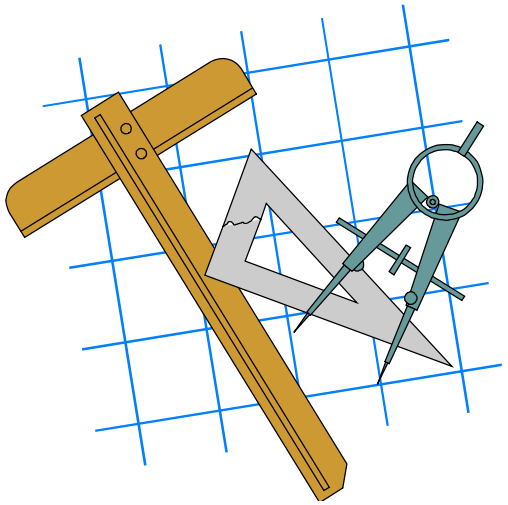


Practice 3.3:

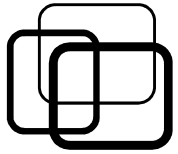
Implement the **"word guessing"** game (same as number guessing described in the lecture) for two player **User** (thinker) and **Computer** (guesser).

- *Computer: guess a word W from **Dictionary**.*
- *User : tell -1 if user word < W in **Dictionary**,*
  *or 1 if user word > W in **Dictionary**,*
  or 0 if user word = W.

loop until User tell 0.



Practice



- Practice 3.4:

  Consider class **Dictionary** in practice 3.2.

  a) Propose an efficient way to do the following series of activities:

  - Initialize a dictionary.
  - Add N words.
  - Look up K words.

  b) Find the complexity of the proposed solution.