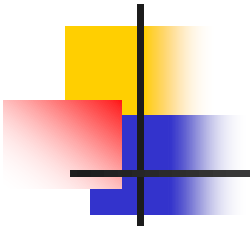
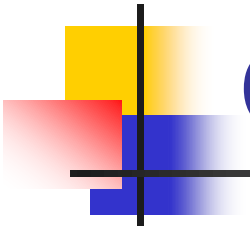
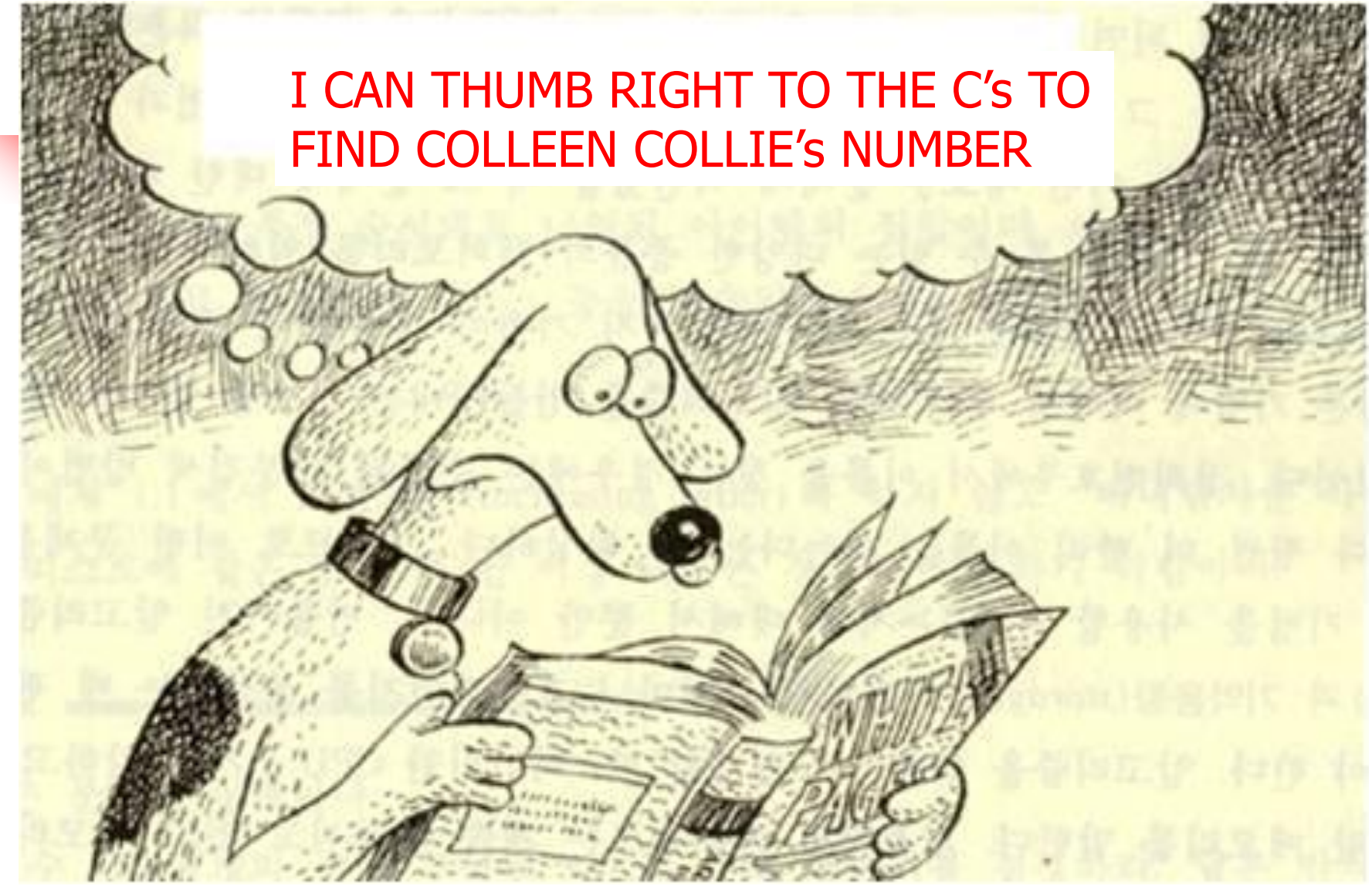


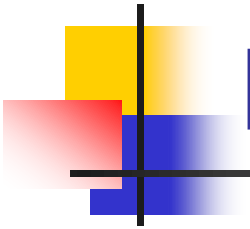
[Chapter 1]
Algorithms : Efficiency, Analysis, And Order



Goals of the Course

- Design
  - Problem solving method
  - 5 design methods: divide-and-conquer, dynamic programming, greedy approach, backtracking, branch-and-bound
- Analysis
  - Efficiency analysis through computational complexity
  - Lower bounds for sorting and searching problems
  - Intractability (NP-completeness)

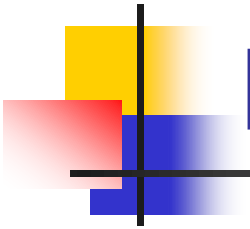

I CAN THUMB RIGHT TO THE C's TO FIND COLLEEN COLLIE's NUMBER
WHITE PAGES



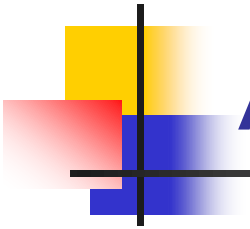
Example

- Problem
  - Determine whether the number x is in the list of S of n numbers. The answer is yes if x is in S and no if it is not.
  - Problem instance
    - S = [10, 7, 11, 5, 13, 8], n = 6, x = 8
    - Solution to this instance is, "Yes, x is in S"
- Algorithm
  - Starting with the first item in S, compare x with each item in S in sequence until x is found or S is exhausted. If x is found, answer yes; if x is not found, answer no.
- Analysis
  - Any *better* algorithm to get the same solution?

Problem Description



- Problem
  - May contain variables that are assigned specific values in the statement of the problem description.
- Parameters
  - Those variables are called parameters. For example, to describe a search problem, we need 3 variables: S, n, x.
- Instance
  - If those parameters are specified, we call it an instance.
  - S = [10, 7, 11, 5, 13, 8], n = 6, x = 8
- Solution to an instance of a problem is the answer to the question asked by the problem in that instance.
  - Solution to the above instance is, "yes, x is in S"



Algorithm Description

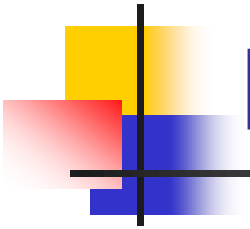
- Natural languages
- Programming languages
- Pseudo-code
  - similar, but not identical, to C++/Java.
  - Notable exceptions: unlimited array index, variable-sized array, mathematical expressions, use of arbitrary types, convenient control structure, and etc.
- In this lecture, algorithms will be represented by pseudo-code similar to C++.

Pseudo-code vs. C++ (1/2)

- Use of arrays
  - In C++, starting at 0
  - In pseudo-code, arrays indexed by other integers are ok.
- Variable-sized array size
  - (e.g) void example (int n)

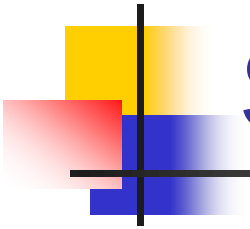
```
{  keytype S[2..n];
   ….
}
```

  - keytype S[low..high]



Pseudo-code vs. C++ (2/2)

- Mathematical expression
  - `low <= x && x <= high` ⇒ `low ≤ x ≤ high`
  - `temp = x; x = y; y = temp` ⇒ `exchange x and y`
- Use of arbitrary type
  - Index
  - Number
  - Bool
- Control structure
  - (e.g.) `repeat (n times) { … }`

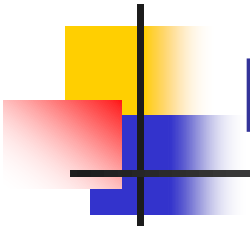


Sequential Search

- Problem
  - Is the key x in the array S of n keys?
- Inputs (parameters)
  - Positive integer n, array of keys S indexed from 1 to n.
- Outputs
  - The location of x in S. (0 if x is not in S.)
- Algorithm
  - Starting with the first item in S, compare x with each item in S in sequence until x is found or S is exhausted. If x is found, answer yes; if x is not found, answer no.

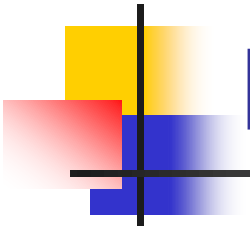
Sequential Search (Psedo-code)

```
void seqsearch(int n,                  // Input(1)
               const keytype S[],      // (2)
               keytype x,              // (3)
               index& location) {      // Output
  location = 1;
  while (location <= n && S[location] != x)
    location++;
  if (location > n)
    location = 0;
}
```



Review: sequential search

- How many comparisons for searching x in S?
  - Depends on the location of x in S
  - In worst case, we should compare ( ) times.
  - In best case,
- Any *better* algorithm to get the same solution?
  - No, we can't, unless there is an extra information in S.



Binary Search

- Problem
  - Is the key x in the array S of n keys?
  - Determine whether x is in the sorted array S of n keys.
- Inputs (parameters)
  - Positive integer n, sorted (non-decreasing order) array of keys S indexed from 1 to n, a key x.
- Outputs
  - The location of x in S. (0 if x is not in S.)

Binary Search (Psedo-code)

```
void binsearch(int n,                  // Input(1)
               const keytype S[],      // (2)
               keytype x,              // (3)
               index& location) {      // Output
  index low, high, mid;
  low = 1; high = n;
  location = 0;

while (low <= high && location == 0) {
    mid = (low + high) / 2;    // integer div
    if (x == S[mid]) location = mid;
    else if (x < S[mid]) high = mid - 1;
    else low = mid + 1;
  }
}
```

Review: binary search algorithm

- How many comparisons for searching *x* in *S*?
  - *S* is already sorted. We know it.
  - For each statement in a while-loop, the number of searching targets will be half.
  - In worst case, we should compare ( ) times.
    - (e.g.) n = 32. S[16], S[16+8], S[24+4], S[28+2], S[30+1]. S[32]
  - In best case, trivial to answer.
- Any *better* algorithm to get the same solution?
  - Hmmm. Very goooooood question.
  - But, wait to see the answer until we get to Chap. 7-8.

Table 1.1 The number of comparisons done by Sequential Search and Binary Search when x is larger than all the array items

Array Size	Number of Comparisons by Sequential Search	Number of Comparisons by Binary Search
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33