COT 6405 Introduction to Theory of Algorithms

Dr. Yao Liu
yliu@cse.usf.edu
http://www.cse.usf.edu/~yliu/

About Instructor

- Dr. Yao Liu, Assistant Professor of Computer Science and Engineering Department
  - http://www.cse.usf.edu/~yliu/
  - yliu@cse.usf.edu
  - 813-974-1079
  - Office: ENB 336
  - Office hours:
    - MW 1:30pm – 3:00pm
    - Or by appointment

Prerequisites

- You either passed the following courses
  - COP 4530 Data Structures
  - Computer Programming (C/C++)
  - COT 3100 Intro to Discrete Structures
- Or you obtained the permission from the instructor

Text

- Required textbook
  - *Tom Cormen, Charles Leiserson, Ronald Rivest, and Cliff Stein, Introduction to Algorithms, Third Edition, MIT Press, ISBN: 978-0-262-03384-8.*

Course Website

- Course website:
  - http://www.cse.usf.edu/~yliu/Algorithm/teaching.html
  - For course materials, e.g., slides, homework files, lecture notes, etc.
  - Will be updated frequently

Grading Policy

- Homework assignments (20%)
- Midterm 1 (20%)
- Midterm 2 (20%)
- Final Exam (40%)

Please note that all tests and the final exam are closed book, closed notes, closed computer, and closed smartphones

Online session students must take in-classroom tests.

Policies on late assignments

- Homework deadlines will be hard.
- Late submission will be accepted with a 15% reduction in grade each day they are late by.
- Once a homework solution is posted or discussed in class, submissions will no longer be accepted.

Make-up Exams

- No make-up exams.
- Exceptions may be made if you are in special situations. You should provide evidences like
  – A doctor's note, which explains why you cannot attend the exam on the exam date
  – A police's report, which shows that you meet an accident on your way to the exam, or you are in some other emergency situations.

Academic Integrity

- You mush finish your assignments and tests on your own.
- An FF grade will be assigned to a student who is caught cheating for this class.
- Typical cheating behaviors include but are not limited to:
  – direct and indirect plagiarizing another student’s work or online resources.
  – modifying incorrect test and homework answers for regrading.

For Students with Disabilities:

- Reasonable accommodations will be made for students with verifiable disabilities. In order to take advantage of available accommodations, student must identify himself or herself to Students with Disabilities Services and provide documentation of a disability.
- http://www.sds.usf.edu/index.asp

COT 6405 Introduction to Theory of Algorithms

Topic 1. A Brief Overview

Course Focus

- The theoretical study of <u>design</u> and <u>analysis</u> of computer algorithms
  - Not about programming, not about math
  - <u>Design</u>: design correct algorithms which minimize cost
    - <u>Efficiency</u> is the design criterion
  - <u>Analysis</u>: predict the cost of an algorithm in terms of resource and performance

Basic Goals of Designing Algorithms

- Basic goals for an algorithm
  - always <u>correct</u>
  - always <u>terminates</u>
- More, we also care about *<u>performance</u>*
  - Tradeoffs between what is possible and what is impossible
  - We usually have a deadline
    - E.g., Computing 24-hour weather forecast within 20 hours

What is an Algorithm?

- A well defined computational procedure that
  - Takes some values as input and produces some values as an output.
- Example: input and output of a sorting algorithm

Input: A sequence of numbers $\{a_1, a_2, \ldots, a_n\}$

Output: a permutation of input sequence such that

$$a_1 < a_2 < \ldots < a_n$$

Instance Input of a problem {2, 5, 9, 6, 4}

Instance Output of a problem {2, 4, 5, 6, 9}

What is an Algorithm? (Cont’d)

- A strategy to solve a problem in a correct and efficient way.
  - E.g., how to find students that have the same birthdays in this classroom?

Why useful?

- Computers are always limited in the computational ability and memory
  - Resources are always limited
  - Efficiency is the center of algorithms
- Course Objective
  - Learn how to solve a problem in an efficient way

Why study algorithms? Tech. Com.

- Google, 1998
  - PageRank
  - MapReduce
- Symantec, 1982
  - Secure Hash Algorithm
  - Endpoint encryption
- Qualcomm, 1995
  - Viterbi Algorithm, Andrew Viterbi
- Match.com, 1993; eharmony.com, chemistry.com
  - Dimension matching

The List goes on: Why study algorithms?

Their impact is broad and far-reaching

- Internet. Web search, packet routing, distribution. file sharing
- Biology. Human genome project, protein folding.
- Computers. Circuit layout, file system, compilers.
- Computer graphics. Hollywood movies, video games, 3-D
- Security. Cell phones, e-commerce, voting machines.
- Multimedia. CD player, DVD, MP3/4, JPG, DivX, HDTV.
- Transportation. Airline crew scheduling, map routing.
- Physics. N-body simulation, particle collision simulation.
- Social networks. Recommendation algorithms
- Communications. Error correction codes

Why study algorithms? (Cont'd)

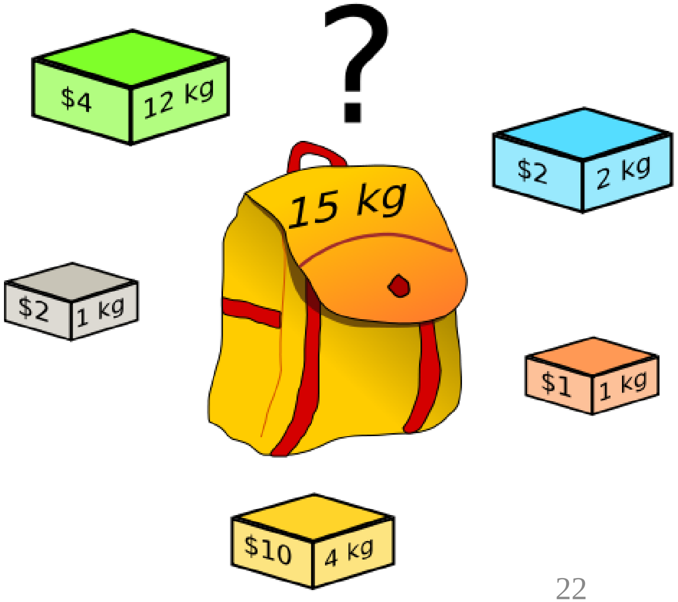
- Algorithm questions play an important role for computer science related job interviews
- Careercup
  - http://www.careercup.com/
- Leetcode
  - http://leetcode.com/

Course Goals

- Solving real-word problems in an efficient way
  - How to achieve?
    - Learn to design, using well known methods
- Implementing algorithms correctly & efficiently
  - Correctness → Arguing correctness
  - Efficiency → Analyzing time complexity

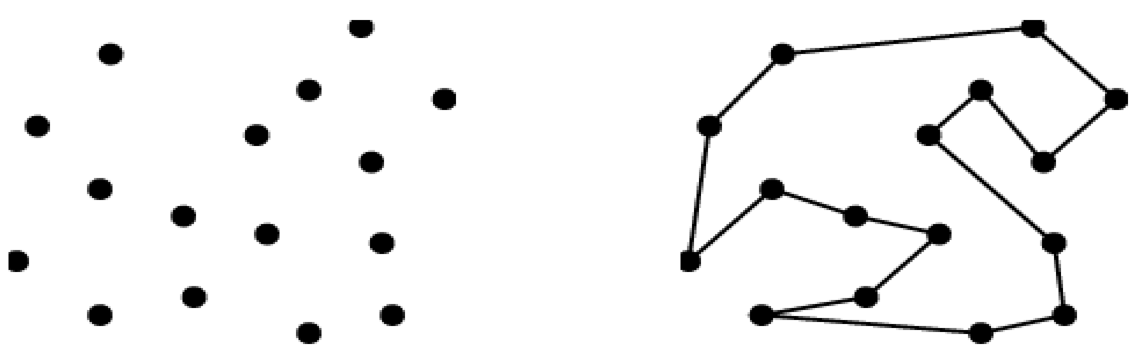
What are Commonly used algorithms

- Search (sequential, binary)
- Sort (mergesort, heapsort, quicksort, etc.)
- … →
- Traversal algorithms (breadth, depth, etc.)
- Shortest path (Floyd, Dijkstra)
- Spanning tree (Prim, Kruskal)
- Knapsack
- Traveling salesman



Hard Problems

- We focus on <u>efficient</u> algorithms in this class
- But some problems which we do NOT know any efficient solutions → NP-complete problems
  - NP: non-deterministic polynomial
- E.g., Traveling-salesman problem, Knapsack,…
  - **Input:** Distance-weighted graph *G*
  - **Problem:** Find the shortest route to visit all of the vertices exactly once



A quiz: the Coin Puzzle

- You have 8 coins which are all the same weight, except for one which is slightly heavier than the others. You can weigh two piles of coins to see which one is heavier (or if they are of equal weight). How can you find the heavier coin?
- What is your strategy?

Math preparation

- Induction
- Logarithm
- Sets
- Permutation and combination
- Limits
- Series
- Probability theory

Quick Review: Some useful formulae

- $\log_a n = \frac{\log_b n}{\log_b a}$
- $x^{\log_a y} = y^{\log_a x}$
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=0}^{t} r^i = \frac{1-r^{t+1}}{1-r} if\ r \neq 1$
- $\sum_{i=0}^{k} ia^i = \frac{a(1-a^k)}{(1-a)^2} - \frac{ka^{k+1}}{1-a}$
- $\sum_{i=0}^{\infty} ia^i = \frac{a}{(1-a)^2}$

Some useful formulae (Cont'd)

- You should be familiar with these already. Please review them and be sure you understand how each can be proved.

Quick Review: Induction

- What is induction?
- Suppose
  - Statement S(k) is true, for fixed constant k
    - Often k = 0
  - If we have S(n) S(n+1) , for all n ≥ k
  - Then, S(n) is true, for all n ≥ k

Proof By Induction

- Claim: formula $S(n)$ is true, for all $n \geq k$
- Basis:
  - Show formula $S(n)$ is true when $n = k$
- Inductive hypothesis:
  - Assume formula $S(n)$ is true, for an arbitrary $n > k$
- Step:
  - Show that formula $S(n+1)$ is true, for all $n > k$

Induction Example: Gaussian Closed Form

- Prove $1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Basis
    - If $n = 0$, then $S(0)= 0 = 0(0+1) / 2$
  - Inductive hypothesis
    - Assume $S(n)= 1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Step: show true for $(n+1)$

    $S(n+1)= 1 + 2 + \ldots + n + n+1 = (1 + 2 + \ldots + n) + (n+1)$

    $= n(n+1)/2 + n+1 = [n(n+1) + 2(n+1)]/2$

    $= (n+1)(n+2)/2$

    $= (n+1)(n+1 + 1) / 2$

Induction Example: Geometric Closed Form

- Prove $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for all $a \neq 1$
  - Basis: show that $a^0 = (a^{0+1} - 1)/(a - 1)$

    $S(0) = a^0 = 1 = (a^1 - 1)/(a - 1)$
  - Inductive hypothesis: S(n) is true
    - Assume $S(n) = a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$
  - Step (show S(n+1) is true)

    $S(n+1) = a^0 + a^1 + \ldots + a^{n+1} = (a^0 + a^1 + \ldots + a^n) + a^{n+1}$

    $= (a^{n+1} - 1)/(a - 1) + a^{n+1}$

    $= (a^{(n+1)+1} - 1)/(a - 1)$

The clarification of lg *n*

- In our textbook and U.S. education systems, lg *n* is the default of $\log_2 n$
- However, In some other countries like China and Indian, lg*n* is the default of $\log_{10} n$
- To avoid confusion, all ``lg *n*'' mentioned in our lectures, assignments, and tests mean the base 2 log, i.e., $\log_2 n$

Examples of Algorithms

- Determine whether x is one of A[1], A[2], . . . , A[n] (and retrieve other information about x).
  - Algorithm: go through each number in order and compare it to x.

```
i = 1;
while(i <= n) and (A[i] ≠ x) do
i = i +1;
if (i > n) then i = 0;
```

Examples of Algorithms(cont'd)

- Number of element comparisons.
- Worst case?
- Best case?
- Average case?

```
i = 1;
while(i <= n) and (A[i] ≠ x) do
i = i +1;
if (i > n) then i = 0;
```

Examples of Algorithms (Cont'd)

- What if the array is sorted in ascending order
  - Algorithm: binary search

```
lo = 1, hi=n;
while lo <= hi {
    mid = lo + (hi - lo)/2
    if (A[mid] == x) then return mid
    else if A[mid] < x then lo = mid +1
    else hi = mid -1
}
```

Examples of Algorithms(cont'd)

- Number of element comparisons.
- Worst case?
- Best case?
- Average case?

```
lo = 1, hi=n;
while lo <= hi {
        mid = lo + (hi - lo)/2
        if (A[mid] ==  x) then return mid
        else if A[mid] < x then lo = mid +1
        else hi = mid -1
 }
```

Examples of Algorithms (Cont'd)

- Square Matrix Multiplication.

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

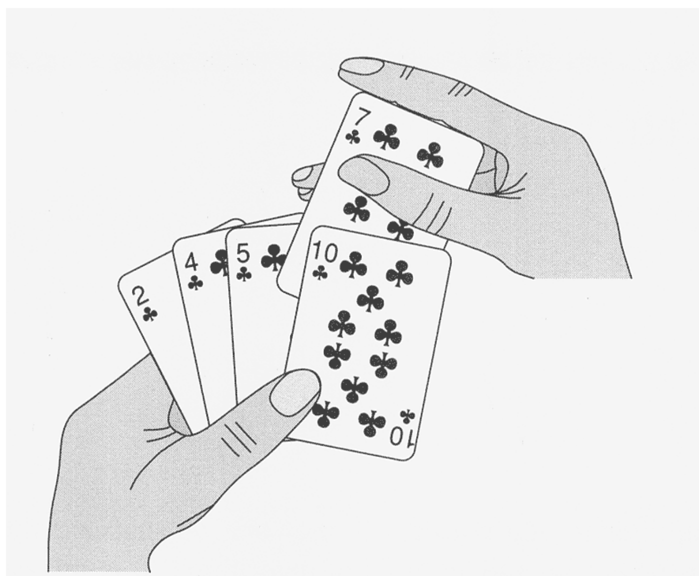
$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Examples of Algorithms (Cont'd)

```
for i = 1 to n do
    for j = 1 to n do
      {
            c[i,j] = 0
             for k = 1 to n do
             c[i,j] = c[i,j] + a[i,k]*b[k,j];
       }
```

Examples of Algorithms (Cont'd)

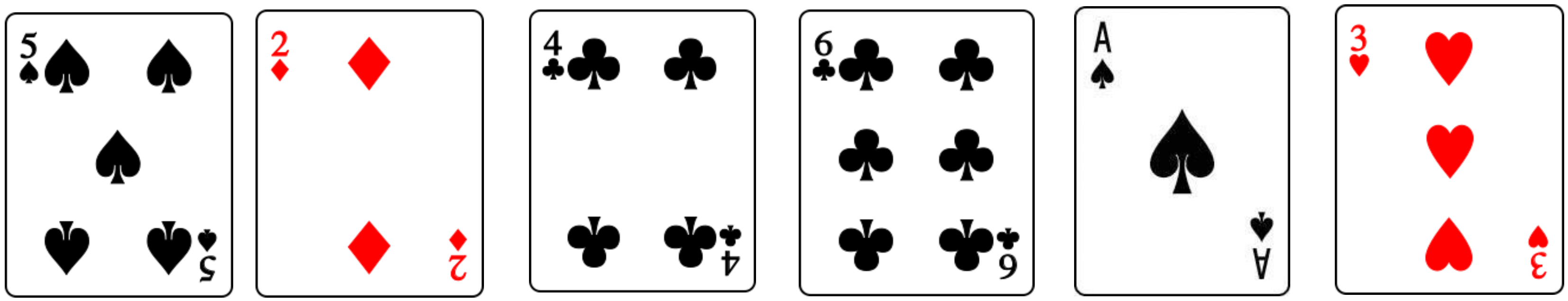
- Given *n* playing cards, sort them in ascending order
- Cards are sorted in place.



Insertion Sort

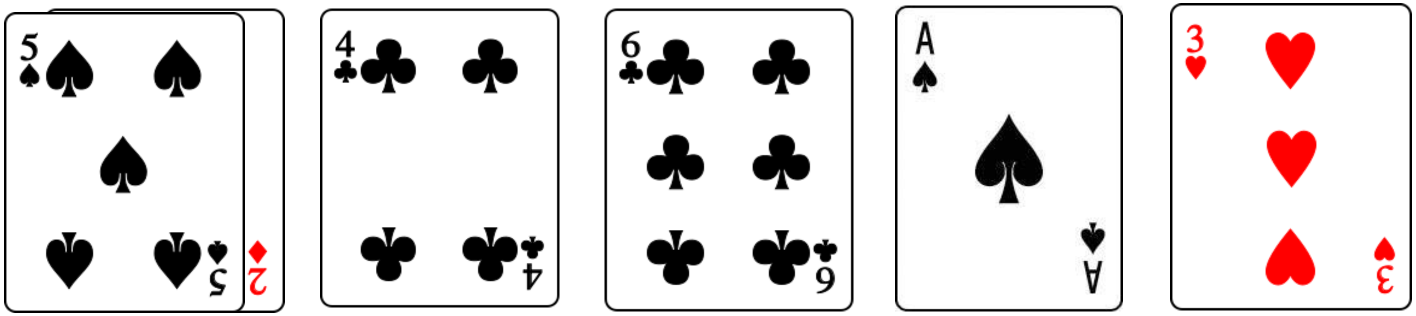
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
    }
```

Insertion Sort (cont'd)



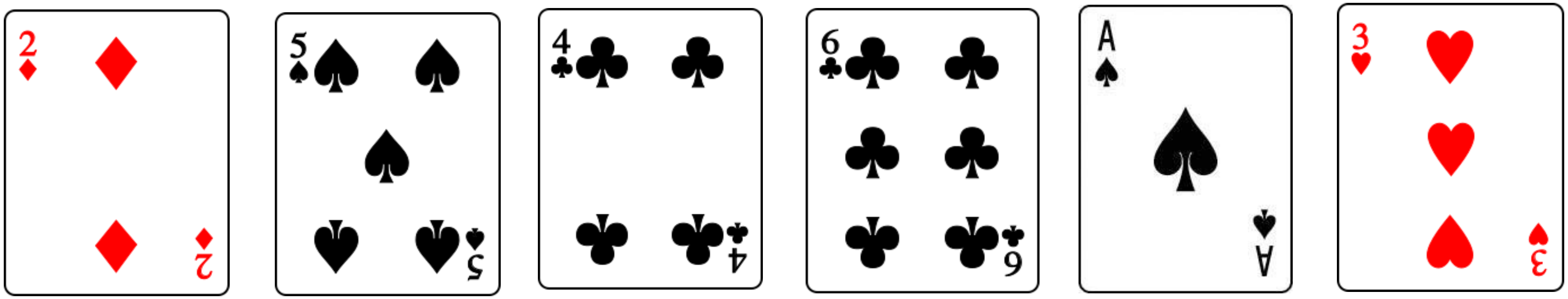
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



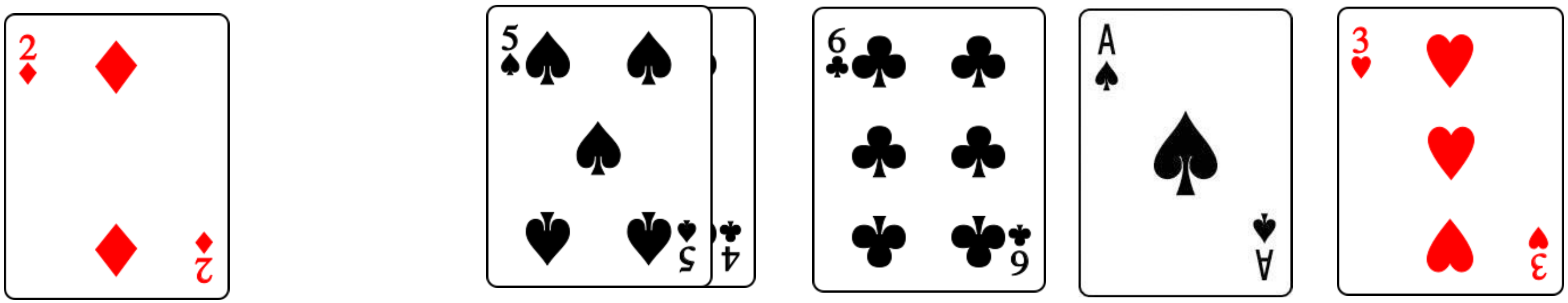
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



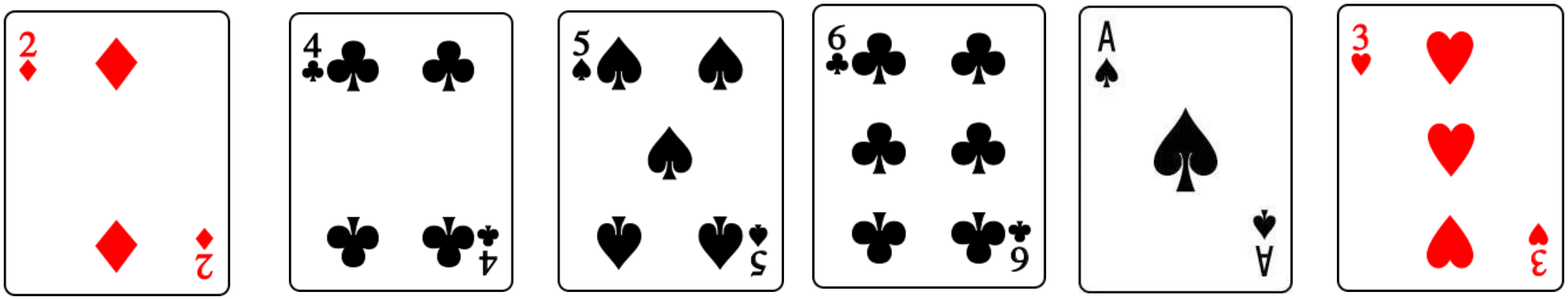
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



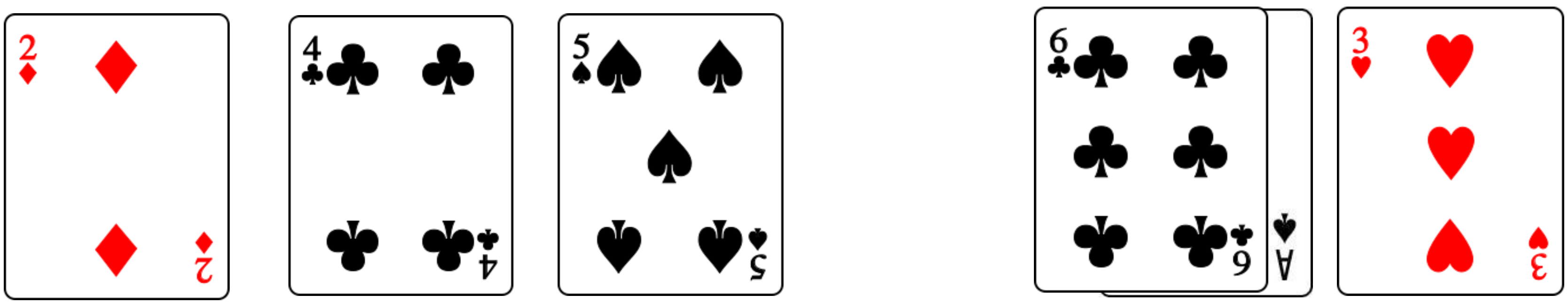
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



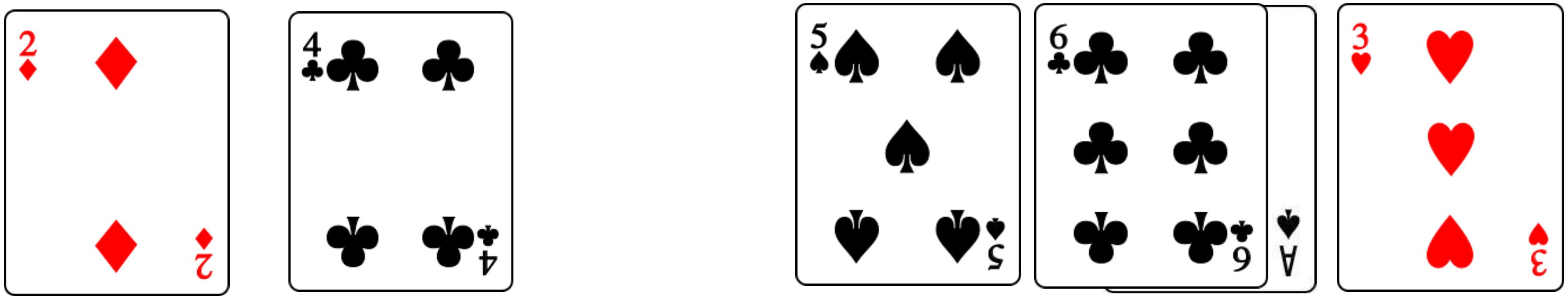

```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont’d)



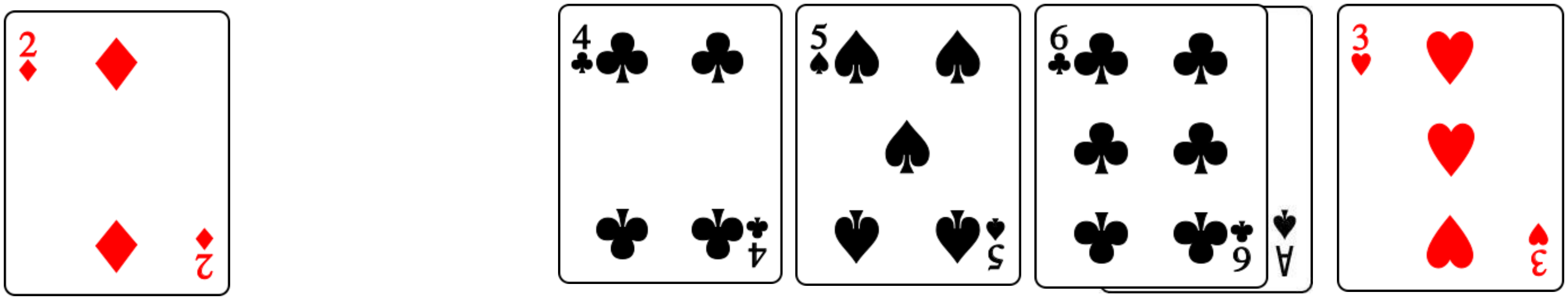

```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



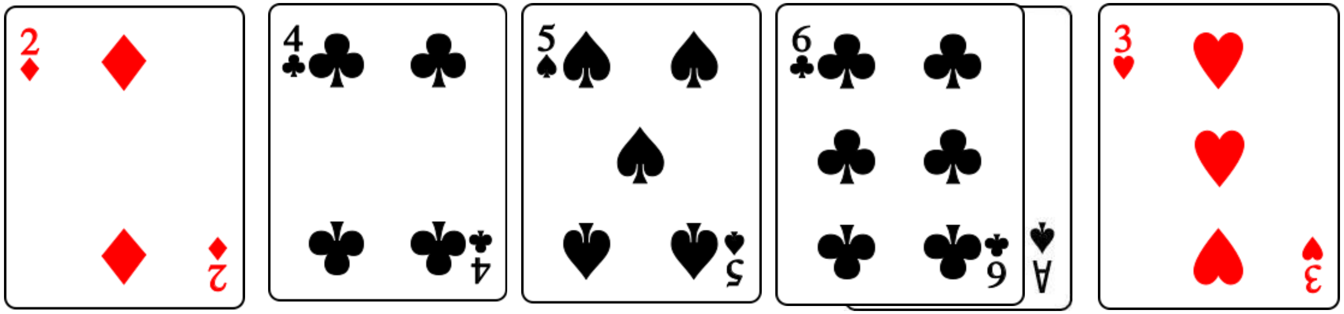
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont’d)



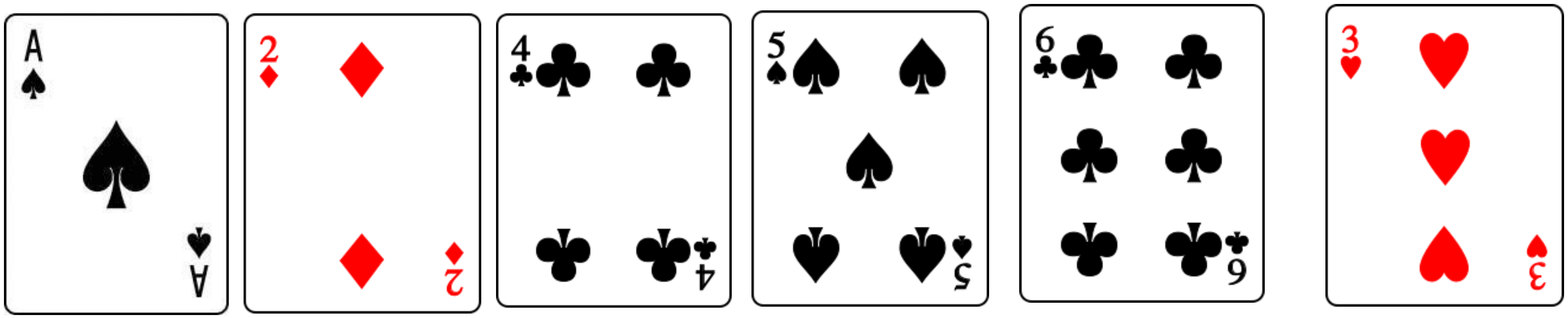
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



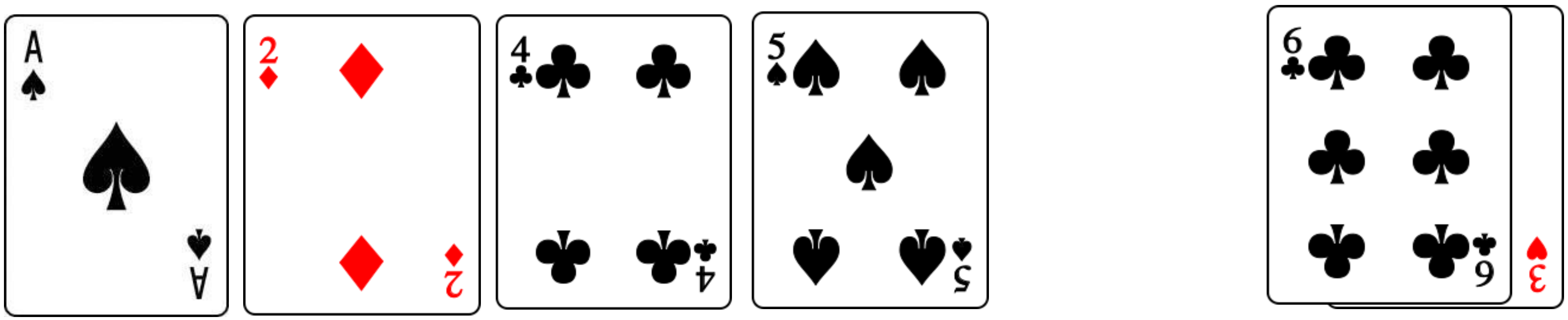
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



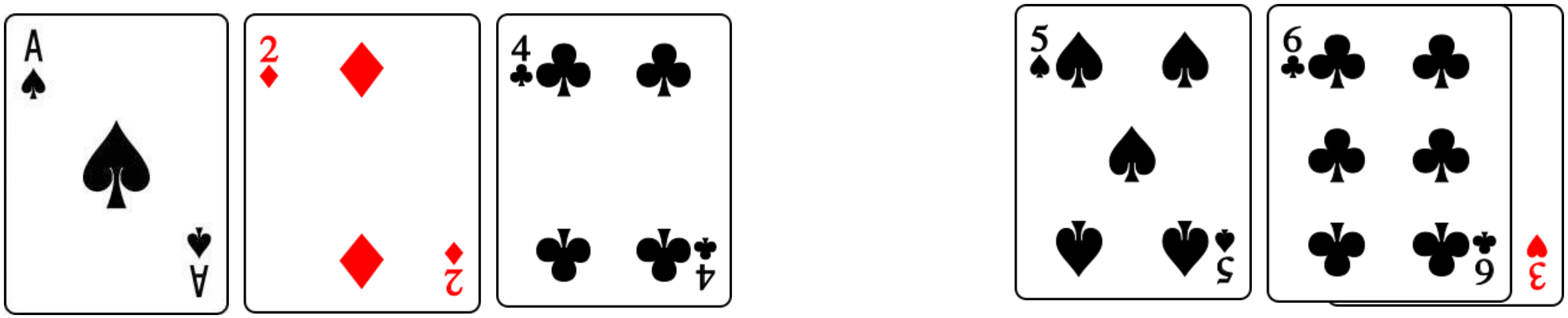

```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont’d)



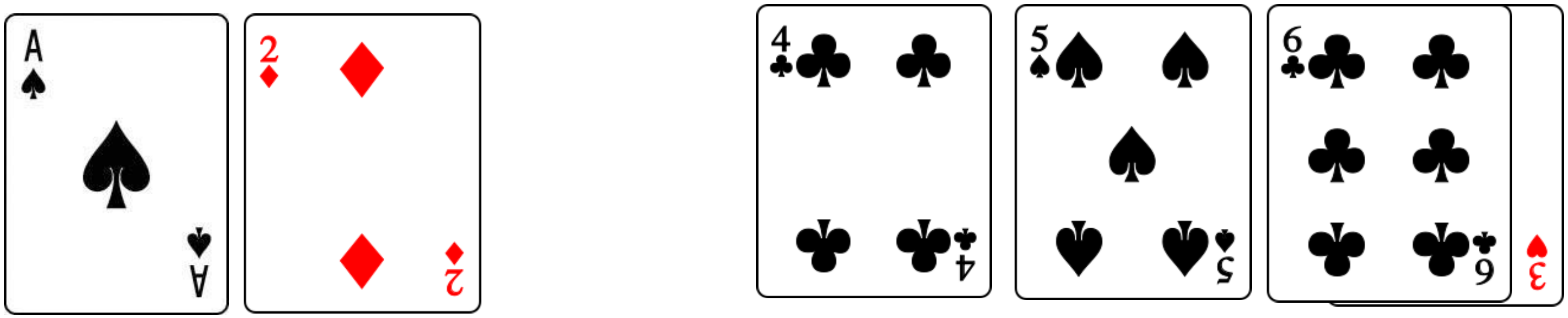
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



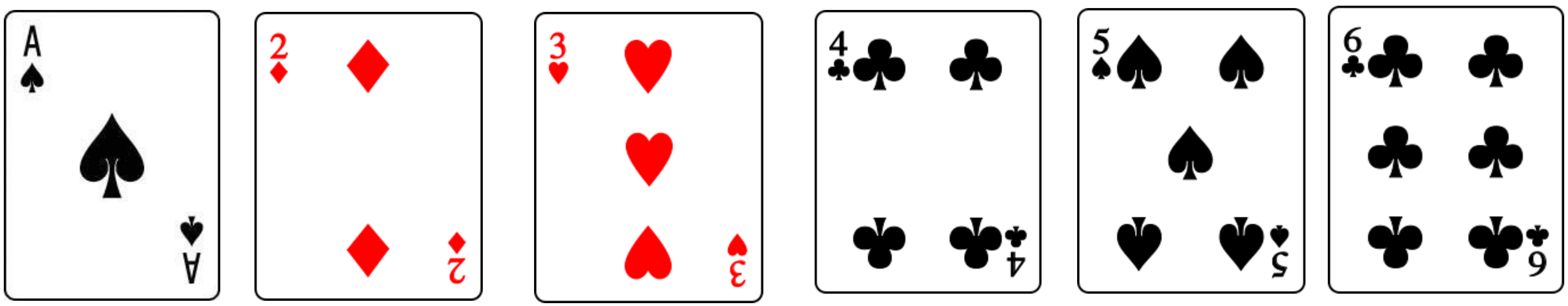
```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)



```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
  }
```

Insertion Sort (cont'd)

- Number of element comparisons.
- Worst case?
- Best case?
- Average case?

```
for j = 2 to n {
        key = A[j];
        i = j -1;
        While (i > 0) and (A[i] > key) {
            A[i+1] = A[i];
            i = i – 1;
        }
        A[i+1] = key;
 }
```