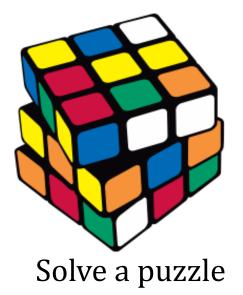
# Lecture 1 DSAA Introduction

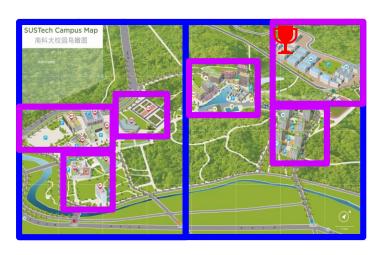
Bo Tang @ 2024, Fall

### Real World Problems



Sort cards





Find a place



Find a shortest path

## Problem Solving

- Example: a sorting problem
  - Sort a set of cards
  - Sort the student list according to scores
- How would a human solve a problem?
  - Uses brain, hands
- How would a computer solve a problem?
  - Uses CPU, memory
  - basic operations: compare two integers,
     move an integer to memory cell X, etc

# Algorithms

# Algorithms

- Algorithm: a well defined sequence of steps for solving a computational problem
  - It produces the correct output
  - It uses basic steps / defined operations
  - It finishes in *finite time*



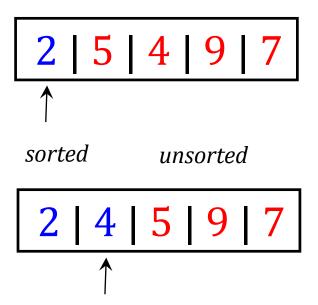
- Idea of a selection sort method
  - Start with empty hand, all cards on table
  - Pick the smallest card from table
  - Insert the card into the hand

What are the *input*, *output*, and *steps*?

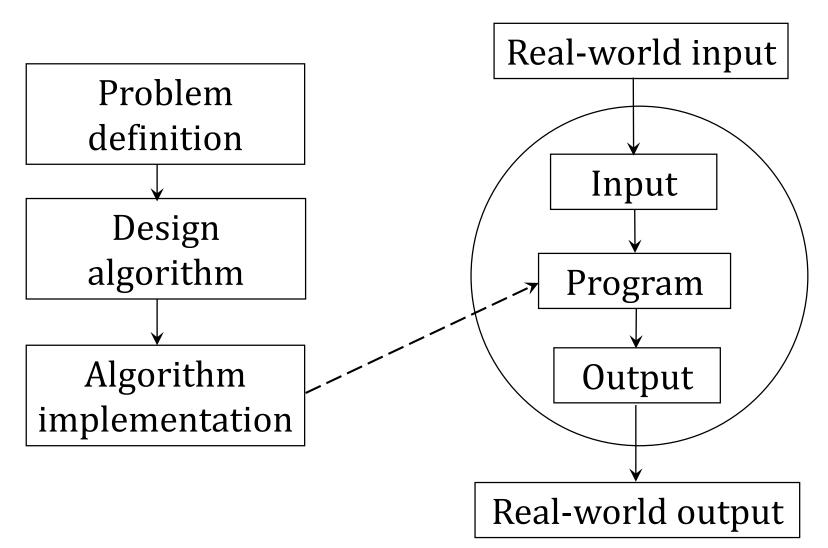
# Algorithms

- Example: selection sort algorithm
- 5 | 2 | 4 | 9 | 7

- Input: an array A of n numbers
- Output: an array A of n numbers in the ascending order
- $\diamond$  Selection-Sort (A[1..n])
  - 1. for integer  $i \leftarrow 1$  to n-1
  - 2.  $k \leftarrow i$
  - 3. for integer  $j \leftarrow i+1$  to n
  - 4. if A[k] > A[j] then
  - 5.  $k \leftarrow j$
  - 6. swap A[i] and A[k]



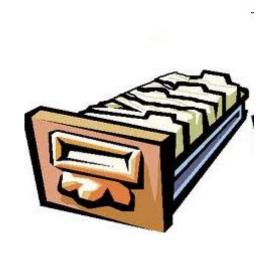
# Algorithms for Problem Solving



# Algorithms may use data structures

#### Data Structures

- What are human's data structures?
  - Used in libraries, books, clinics, companies, ......



Oriental lampshades, 105-107 Patterns, how to make, 87, 135, 137 Piping, 120 Pleating, 99-104 Pricing your work, 152 Relining lampshades, 118 Rewiring lamps, 80-82 Roses, 126 Ruffles, how to make, 122-123 Scallops, 33, 85 Shampooing lampshades, 151 Shapes of lampshades, 31-41 Silhouettes of lamps, 21-30 Slipcovers for lamps, 108 Smocking, 96, 99 Spiders, different kinds, 31-32 Sunburst pleating, 100-102

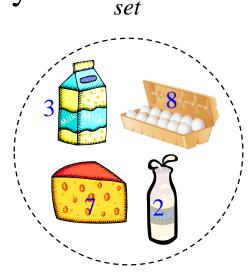
How about computer's data structures?

#### Data Structures

- Let S be a set of items, and x be a search key
  - A key is a number, e.g., product id
- Useful operations on a set S
  - $\diamond$  Search(S, x): search whether x appears in S
  - $\bullet$  Insert(S, x): insert item x into S
  - $\triangleright$  Delete(S, x): remove item x from S



- A way of organizing data objects for efficient usage
- Building blocks for designing algorithms



search key



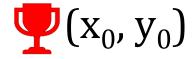
### Data Structures

You will learn them in this course ...

- Why so many data structures?
  - They support different operations, and with different time complexities
- Which data structure is better?
  - Depends on the frequency of operations used in your algorithm
  - E.g., it is fast for the most frequent operation in your algorithm

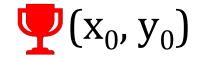
| Array       |
|-------------|
| Linked List |
| Stack       |
| Queue       |
| Hash table  |
| Неар        |
| Tree        |
|             |

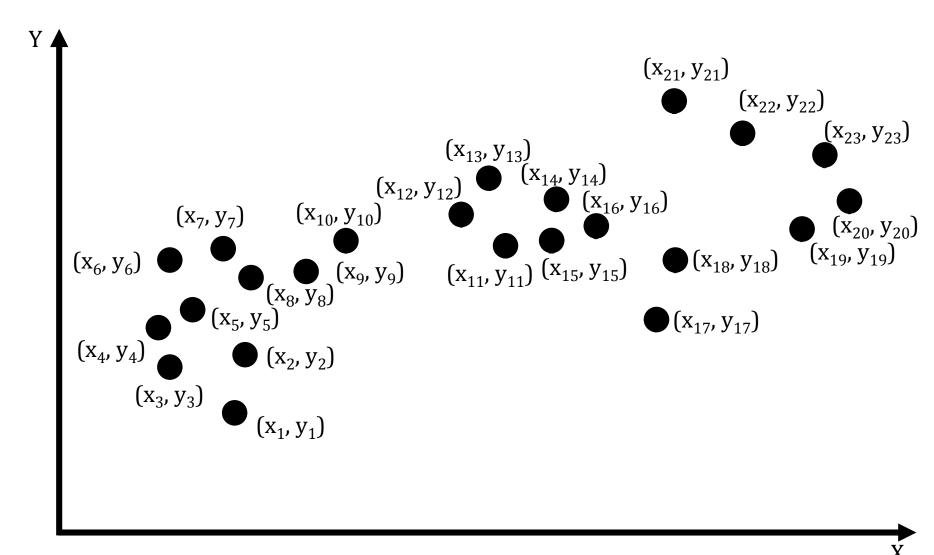
# DSAA demo: find LY102

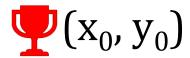




Any ideas ?

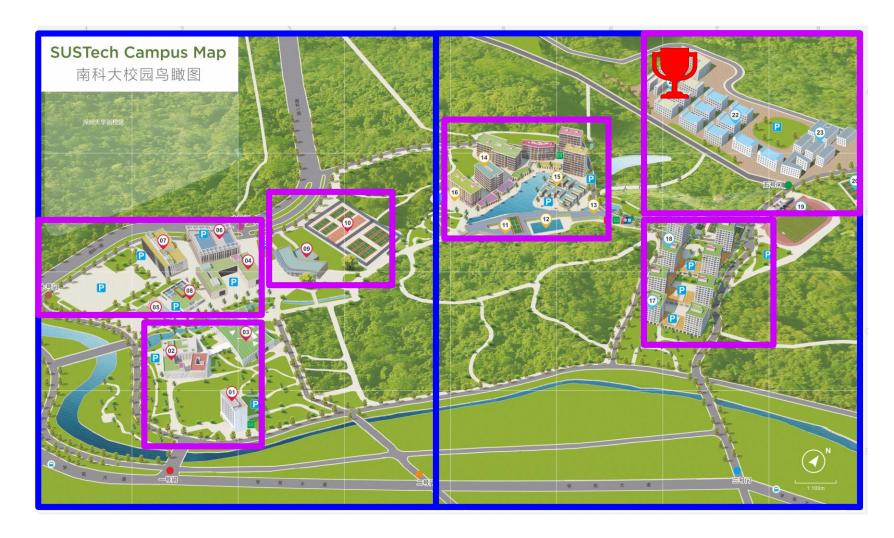




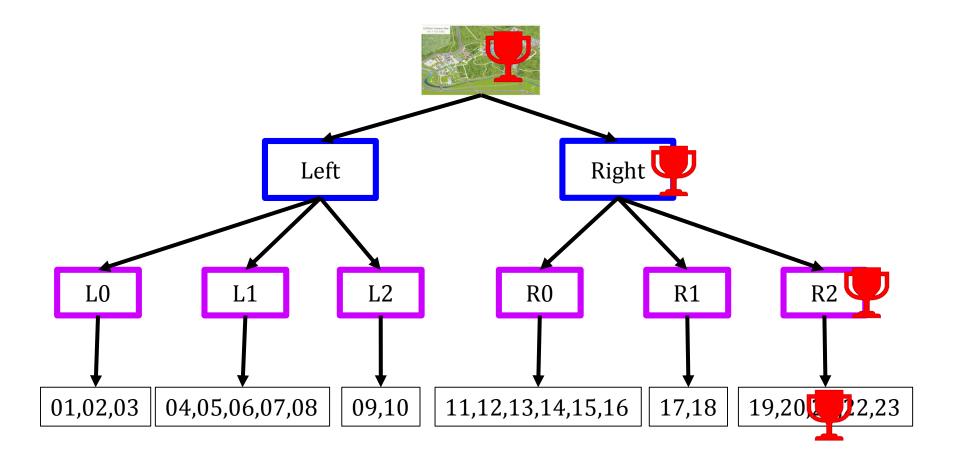


if(xi==x0 && yi==y0)  $x_{21} == x_0 \&\& y_{21} == y_0$ "It is our classroom" else "It is not our classroom"



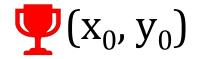






### Find LY102 Classroom

- ♦ Let S be R-tree node, and x be our classroom
- $\bullet$  Useful operations on a set S
  - $\diamond$  Cover(S, x): verify whether S covers x
  - ⋄ Children(S): Find the children of S
  - $\diamond$  Search(S, x): search whether x appears in S
- *R*-tree structure:
  - A way of organizing data objects for efficient usage
    - Prune a subset of candidates by one checking function
  - Building blocks for designing algorithms





# Algorithms Design Techniques

# Algorithmic Design Techniques

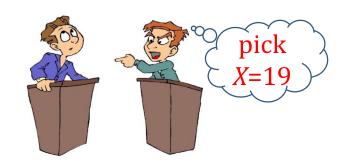
1..i-1

- Incremental technique
  - <u>Build</u> a solution into a <u>larger solution</u>
  - ⋄ E.g., we have a sorted subarray A[1..i-1], then append an item to obtain a sorted subarray A[1..i]
- Recursive technique (or divide-and-conquer)
  - Reduce the problem into smaller subproblems
  - $\bullet$  E.g., find the smallest item in subarray A[i..n], then sort the subarray A[i+1..n]



### Guess the Number Game

- Rules
  - We Host: pick a secret integer X from 1 to 20
  - Guest: guess V as the answer
  - → Host: "V is too low" / "V is too high" / "V is correct!"
- Simple strategy: test each integer in ascending order
  - $\diamond$  Guess 1  $\rightarrow$  too low
  - $\bullet$  Guess 2  $\rightarrow$  too low
  - **>** .....
  - $\bullet$  Guess 19  $\rightarrow$  correct!



Can you suggest a more efficient strategy?

# Divide-and-conquer Strategy

pick X=19

- Guess the number game
  - $\diamond$  Guess 10  $\rightarrow$  too low
    - [Think] Is X between 1 and 9? NO
    - [Think] Is X between 11 and 20? YES

- $\diamond$  Guess 15  $\rightarrow$  too low
  - [Think] Is X between 11 and 14? NO
  - [*Think*] Is *X* between 16 and 20? YES

- $\bullet$  Guess 18 $\rightarrow$  too low
- $\diamond$  Guess 19 $\rightarrow$  correct!

# Recursive Technique



#### Iteration

 When we encounter a problem that requires repetition, we often use iteration – i.e., some type of loop

- Sample problem: printing the series of integers from n1 to n2, where n1 <= n2.</li>
  - printSeries(1,8) should print the following
    1, 2, 3, 4, 5, 6, 7, 8

Iterative solution:

```
public static void printSeries(int n1, int n2){
    for(int i=n1; i<n2; i++)
        System.out.print(i + ", ");
    System.out.println(n2);
}</pre>
```

### Recursion

- An alternative approach to problems that require repetition is to solve them using recursion
- A recursive method is a method that calls itself
- Applying this approach to the printSeries problem:
  public static void printSeries(int n1, int n2){

```
if(n1 == n2){
          System.out.println(n2);
} else {
          System.out.print(n1 + ", ");
          printSeries(n1 + 1, n2);
}
```

# Tracing a Recursive Method

return

# Recursive Problem-Solving

- When we use recursion, we solve a problem by reducing it to a simpler problem of the same kind
- We keep doing this until we reach a problem that is simple enough to be solved directly.
- The simplest problem is known as the base case

The base case stops the recursion, because it does not make another call to the method

### Recursive Problem-Solving

If the base case hasn't been reached, we execute the recursive case

- The recursive case:
  - Reduces the overall problem to one or more simpler problems of the same kind
  - Makes recursive calls to solve the simpler problems.

## Template of a Recursive Method

```
recursiveMethod(parameters){
   if(stopping condition){
          // handle the base case
   } else {
          // recursive case
          // possibly do something here
          recursiveMethod(Modified parameters);
          // possibly do something here
```

- There can be multiple base cases and recursive cases
- When we make the recursive call, we typically use parameters that bring us closer to a base case

# Printing a File to the Console

Here is a method that prints a file using iteration
public static void printFile (Scanner input){
 while(input.hasNextLine()){
 System.out.println(input.nextLine());
 }
}

# Printing a File in Reverse Order

- What if we want to print the lines of a file in reverse order?
  - It's not easy to do this using iteration. Why?
  - It's easy to do it using recursion!

How could we modify our pervious method to make it print the lines in reverse order?

```
public static void printFileRecursive (Scanner input){
   if(!input.hasNextLine()){ // base case
        return;
   } else { // recursive case
        String line = input.nextLine();
        printFileRecursive(input); // print the rest
        System.out.println(line);
   }
```

# Thinking Recursively

- When solving a problem using recursion, ask yourself these questions:
  - Mow can I break this problem down into one or more smaller subproblems?
    - Make recursive method calls to solve the subproblems
  - What are the base cases?
    - i.e., which subproblems are small enough to solve directly?
  - Do I need to combine the solutions to the subproblems? If so, how should I do so?

# Thinking Recursively

```
void I_Know_Recursion()
void 要理解递归()
                               if(I do not know recursion)
  if (不理解) {
    要理解递归();
                                      I_Know_Recursion();
int main()
                        int main()
  要理解递归();
                               I_Know_Recursion();
                               return 0;
  return 0;
```

Is it infinite loop?

## Take Home Message

- Algorithms
  - How to sort cards?
- Data structures
  - How to find our classroom?
- Divide and conquer strategy
  - Now to guess the number game?
- Iteration
  - How to print a series of numbers
- Recursion
  - Now to print a file in reverse order?
  - Why iteration is not easy to print a file in revers order?

### Thank You!