

Lecture 1

DSAA Introduction

Bo Tang @ 2022, Fall

Several pages are based on the notes by Dr. Ken Yiu (PolyU) and Dr. David Sullivan (BU)

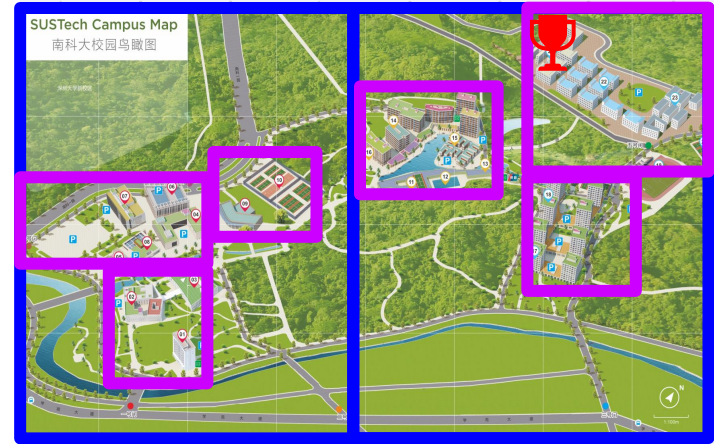
Real World Problems



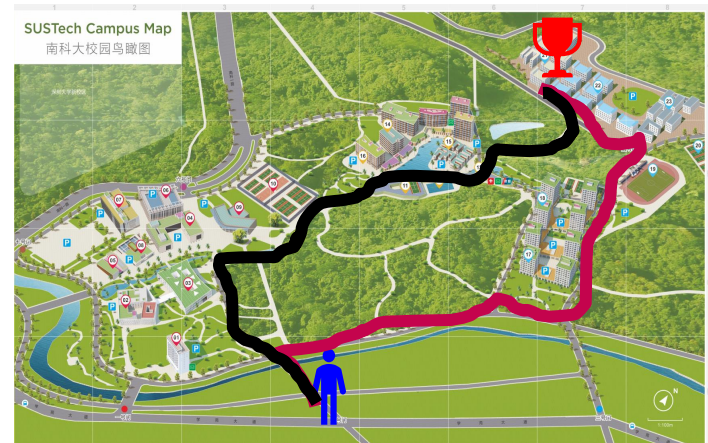
Sort cards



Solve a puzzle



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

- ◆ Input: an **array** A of n numbers
- ◆ Output : an **array** A of n numbers in the ascending order
- ◆ 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 $A[k]$

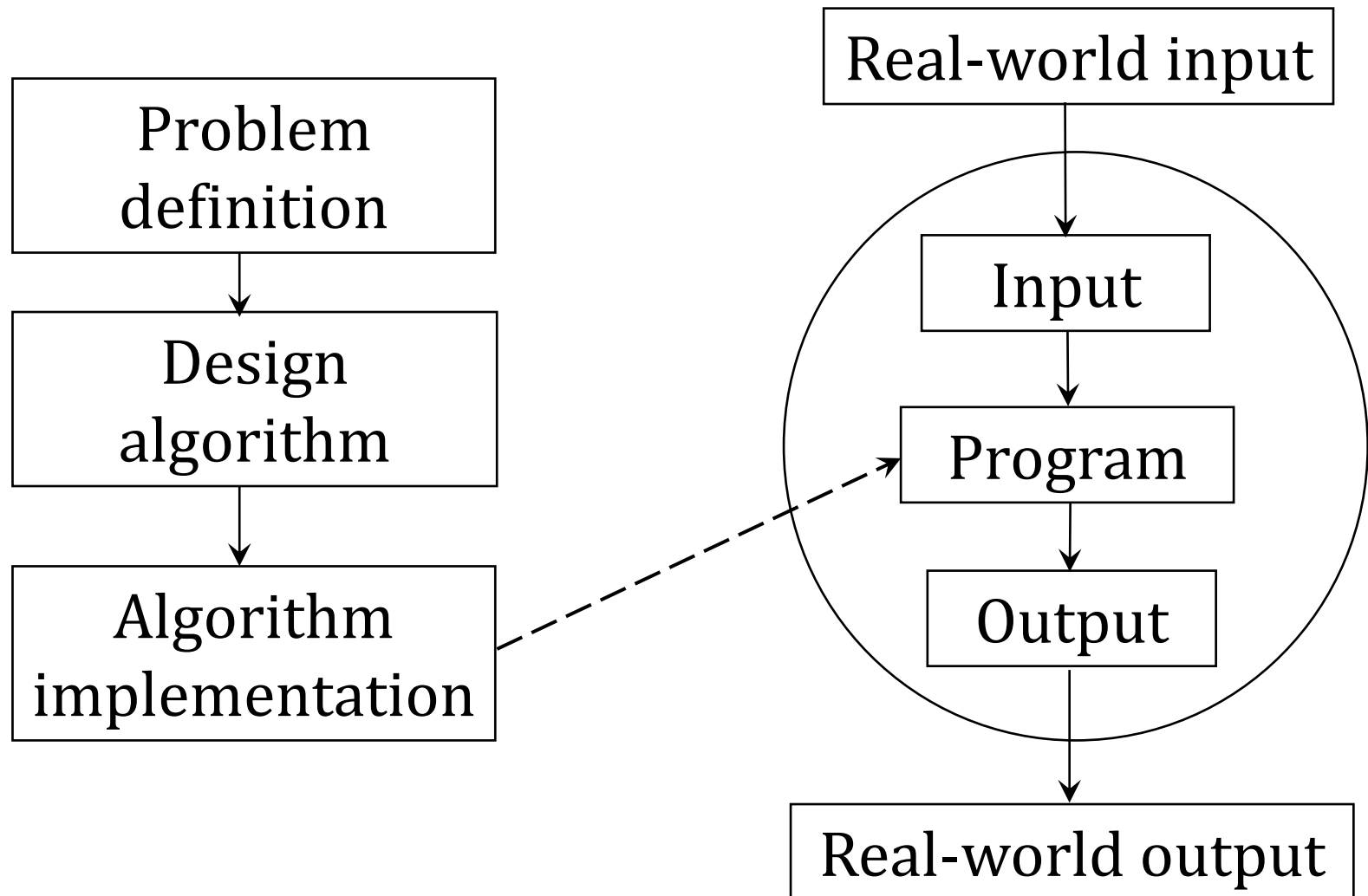


sorted

unsorted



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

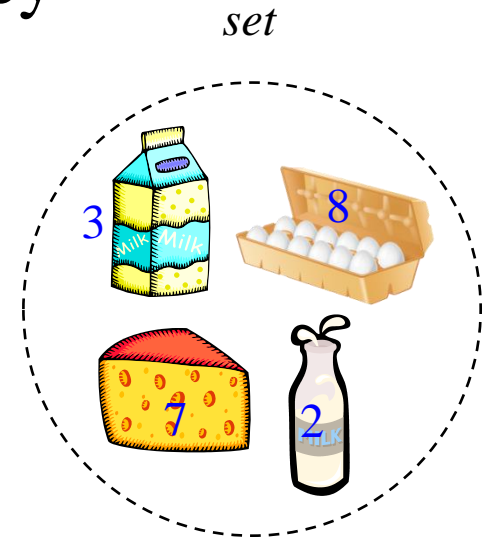
- A key is a number, e.g., product id

- Useful operations on a set S

- Search(S, x): search whether x appears in S
- Insert(S, x): insert item x into S
- Delete(S, x): remove item x from S

- Data structure:*

- 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
Heap
Tree
.....

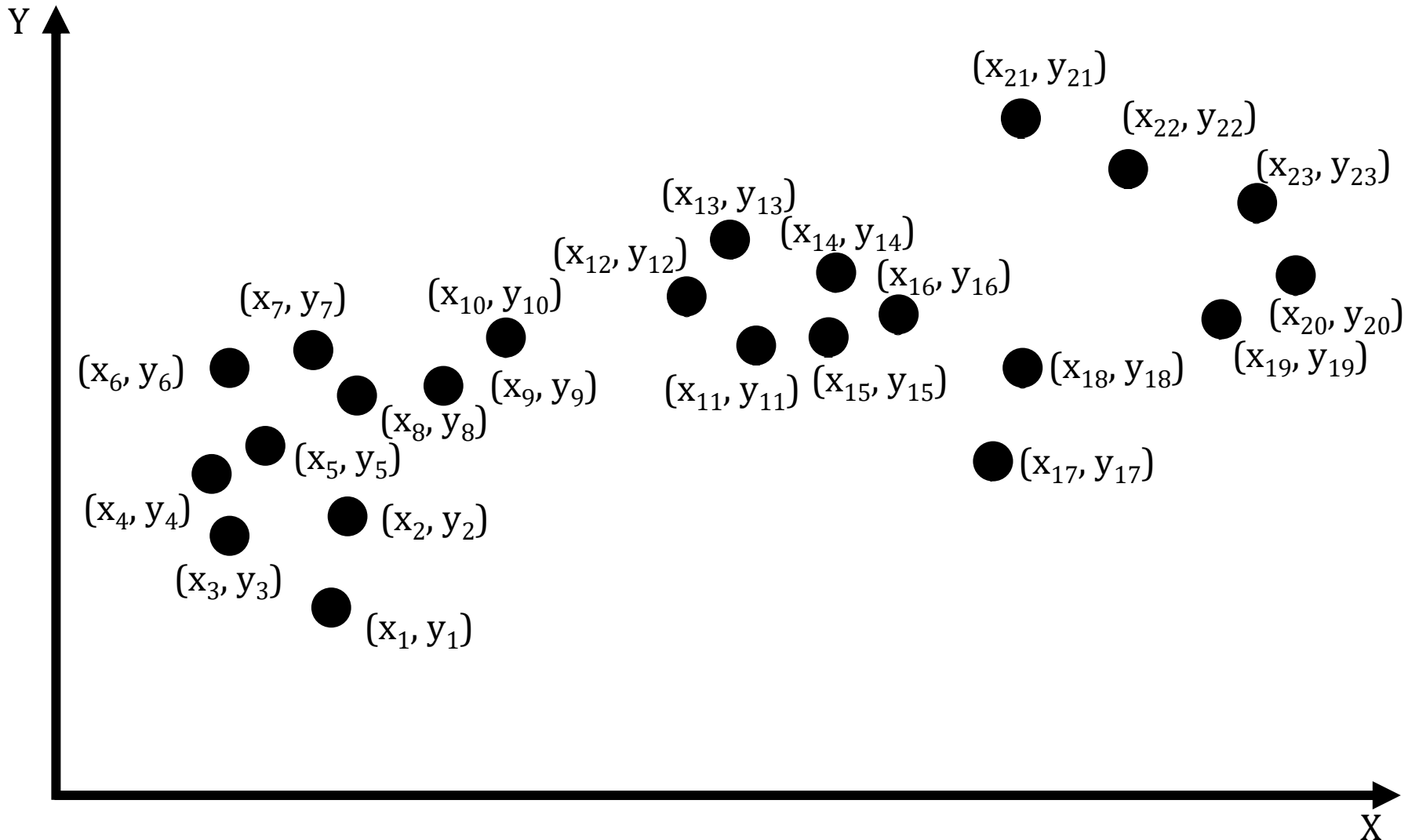
DSAA demo: find LY102

Find LY102 Classroom (x_0, y_0)



◆ Any ideas ?

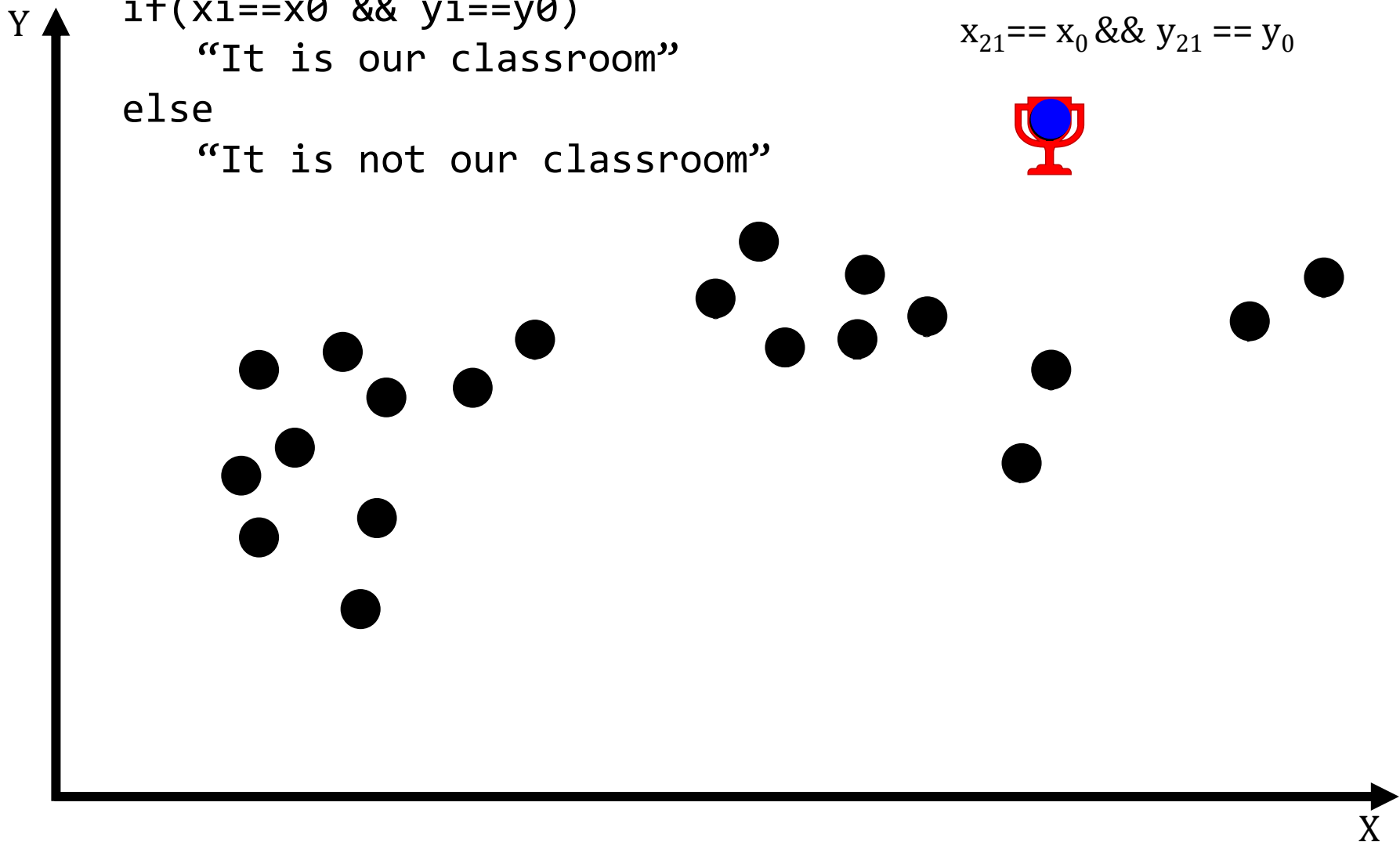
Find LY102 Classroom (x_0, y_0)



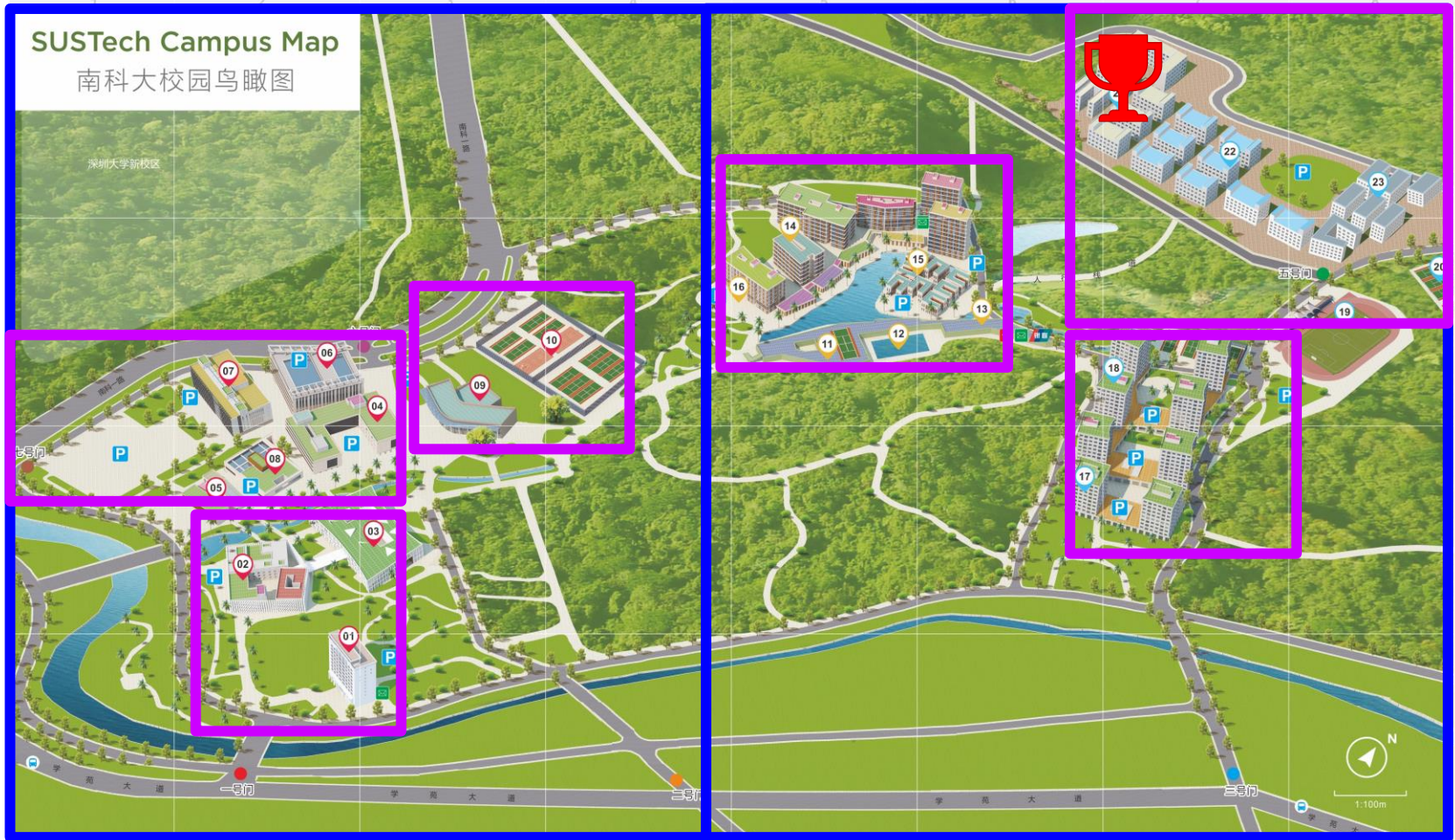
Find LY102 Classroom 🏆 (x_0, y_0)

```
if(xi==x0 && yi==y0)
    "It is our classroom"
else
    "It is not our classroom"
```

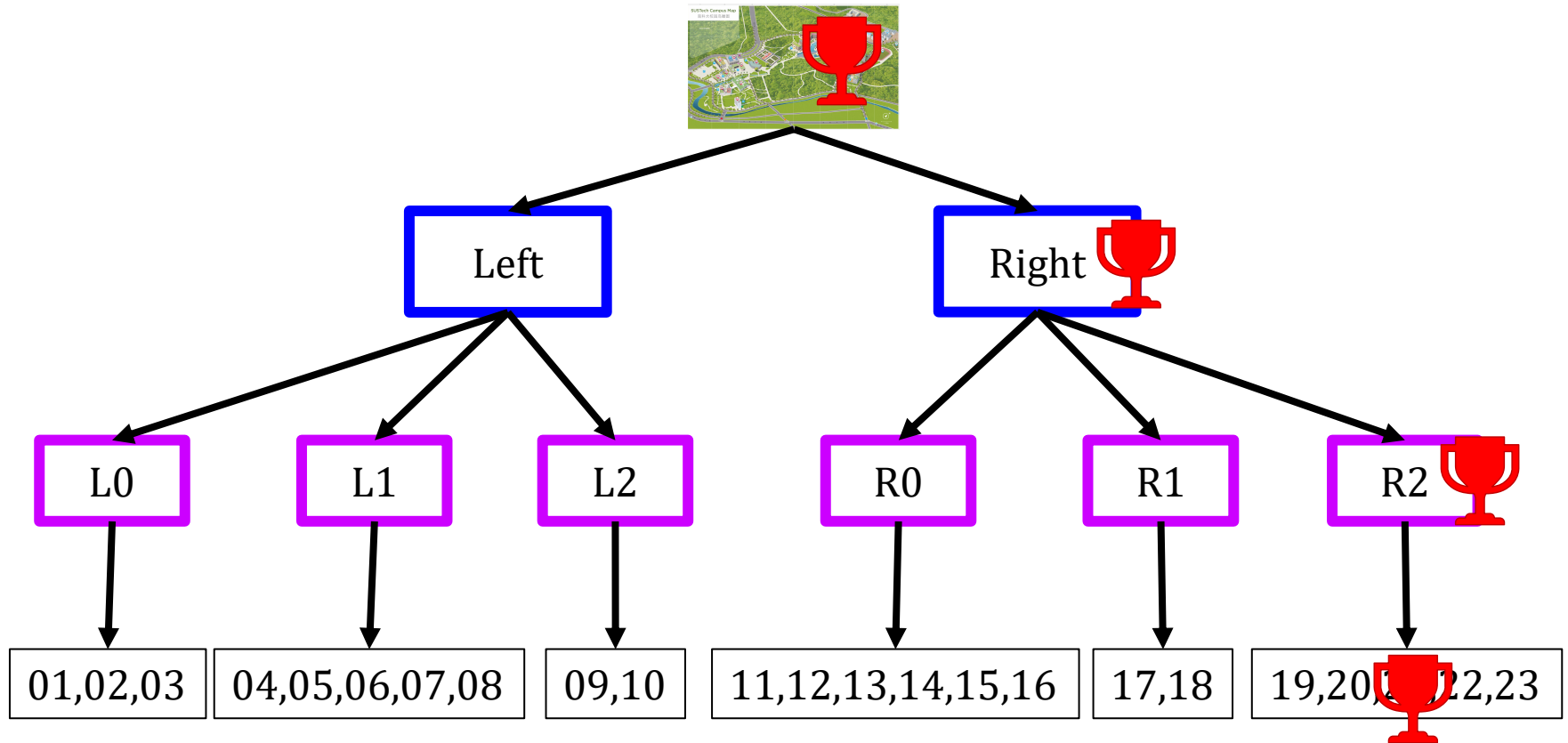
$x_{21} == x_0 \ \&\& \ y_{21} == y_0$



Find LY102 Classroom



Find LY102 Classroom



Find LY102 Classroom

- ◆ Let S be R-tree node, and x be our classroom
- ◆ Useful operations on a set S
 - ◆ $\text{Cover}(S, x)$: verify whether S covers x
 - ◆ $\text{Children}(S)$: Find the children of S
 - ◆ $\text{Search}(S, x)$: 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

Find LY102 Classroom 🏆 (x_0, y_0)

$$(x_0, y_0)$$


Southern University of Science and Technology

◆ Any ideas ?

Algorithms Design Techniques

Algorithmic Design Techniques

◆ *Incremental technique*

1.. $i-1$ | i

- ◆ Build a solution into a larger solution
- ◆ 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)

递归，分而治之

- ◆ Reduce the problem into smaller subproblems
- ◆ E.g., find the smallest item in subarray $A[i..n]$, then sort the subarray $A[i+1..n]$

i | $i+1..n$

Guess the Number Game

◆ Rules

- ◆ 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

- ◆ Guess 1 → too low
- ◆ Guess 2 → too low
- ◆
- ◆ Guess 19 → **correct!**



◆ Can you suggest a more efficient strategy?

Divide-and-conquer Strategy

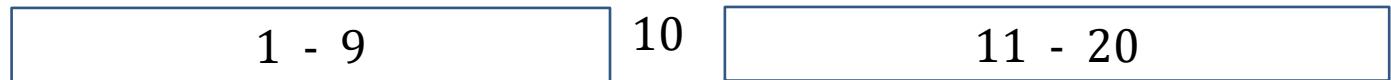
pick
 $X=19$



◆ Guess the number game

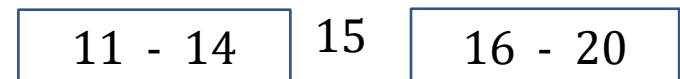
◆ Guess 10 → too low

- ◆ [Think] Is X between 1 and 9? NO
- ◆ [Think] Is X between 11 and 20? YES



◆ Guess 15 → too low

- ◆ [Think] Is X between 11 and 14? NO
- ◆ [Think] Is X between 16 and 20? YES



◆ Guess 18 → too low

◆ Guess 19 → 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 $n1 \leq n2$.
 - ◆ `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);  
}
```

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

- ◆ What happens when we execute `printSeries(1,3)`

```
printSeries(1,3):
```

```
    System.out.print(1 + ", ");    // 1,
```

```
    printSeries(2,3):
```

```
        System.out.print(2 + ", "); // 1, 2,
```

```
        printSeries(3,3):
```

```
            System.out.println(3); // 1, 2, 3 \n
```

```
            return
```

```
        return
```

```
    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***

```
public static void printSeries(int n1, int n2){  
    if(n1 == n2){                // base case  
        System.out.println(n2);  
    } else {  
        System.out.print(n1 + ", ");  
        printSeries(n1 + 1, n2);  
    }  
}
```

- ◆ 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*

```
public static void printSeries(int n1, int n2){  
    if(n1 == n2){                                // base case  
        System.out.println(n2);  
    } else {                                     // recursive case  
        System.out.print(n1 + ", ");  
        printSeries(n1 + 1, n2);  
    }  
}
```

- ◆ 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());  
    }  
}
```

- ◆ Here is a method that uses **recursion** to do the same thing:

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

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:
 - ◆ How 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 要理解递归()  
{  
    if (不理解) {  
        要理解递归();  
    }  
}  
  
int main()  
{  
    要理解递归();  
    return 0;  
}
```

```
void I_Know_Recursion()  
{  
    if(I do not know recursion)  
    {  
        I_Knew_Recursion();  
    }  
}  
  
int main()  
{  
    I_Know_Recursion();  
    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
 - ◆ How to guess the number game ?
- ◆ Iteration
 - ◆ How to print a series of numbers
- ◆ Recursion
 - ◆ How to print a file in reverse order ?
 - ◆ Why iteration is not easy to print a file in revers order?

Thank You!