Introduction, Recursion and Complexity of Algorithms

Data Structures and Algorithms

Quang-Huan Luu, MsC
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Outcomes

- **L.O.1.**Determine the complexity of simple algorithms (polynomial time - nested loop - no recursive)
    - **L.O.1.1** Give definition of Big-O notation.
    - **L.O.1.2** Determine complexity of simple polynomial algorithms.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Overview

1 Data structures and Algorithms: Basic concepts
- Algorithm
- Pseudocode
- Data structures
- Classes
- Pointers
- Arrays

2 Recursion
- Properties of recursion
- Designing recursive algorithms
- Recursion and backtracking
- Recursion implementation in C/C++

3 Complexity of Algorithms
- Algorithm Efficiency
- Asymptotic Analysis
- Problems and common complexities
- P and NP Problems

Sources of Materials

1. We would like to thank **Dr. The-Nhan LUONG**, a former instructor of our Department, for the composing of this document.
2. This document also uses figure, sentences and demo source code from the following sources:
    - The old presentation for course *Data Structures and Algorithms* edited by other members in our Department
    - Book entitled **Data Structures - A Pseudocode Approach with C++ (first edition, 2001)** written by Richard F. Gilberg and Behrouz A. Forouzan

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Basic concepts

What is Data?

(Source:

Quang-Huan Luu, MsC

Data structures and Algorithms: Basic concepts

Algorithm

Pseudocode

Data structures

Classes

Pointers

Arrays

Recursion

Properties of recursion

Designing recursive algorithms

Recursion and backtracking

Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency

Asymptotic Analysis

Problems and common complexities

P and NP Problems

What is Data?

Data

Data is information that has been translated into a form that is more convenient to calculate, analyze.

Example

- Numbers, words, measurements, observations or descriptions of things.

- Qualitative data: descriptive information,
- Quantitative data: numerical information (numbers).
  - Discrete data can only take certain values (like whole numbers)
  - Continuous data can take any value (within a range)

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data type

Class of data objects that have the same properties.

Data type

1. A set of values
2. A set of operations on values

Example

Type	Values	Operations
integer	$-\infty, ..., -2, -1, 0, 1, 2, ..., \infty$	$*, +, -, \%, /, ++, --, ...$
floating point	$-\infty, ..., 0.0, ..., \infty$	$*, +, -, /, ...$
character	$\backslash 0$, ..., 'A', 'B', ..., 'a', 'b', ..., $\sim$	$<, >, ...$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structure

What is a data structure?

1. A combination of elements in which each is either a data type or another data structure
2. A set of associations or relationships (structure) that holds the data together

Example

An array is a number of elements of the same type in a specific order.

1	2	3	5	8	13	21	34

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Abstract data type

The concept of abstraction:

- Users know what a data type can do.
- How it is done is hidden.

Definition

An **abstract data type** is a data declaration packaged together with the operations that are meaningful for the data type.

1. Declaration of data
2. Declaration of operations
3. Encapsulation of data and operations

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Abstract data type

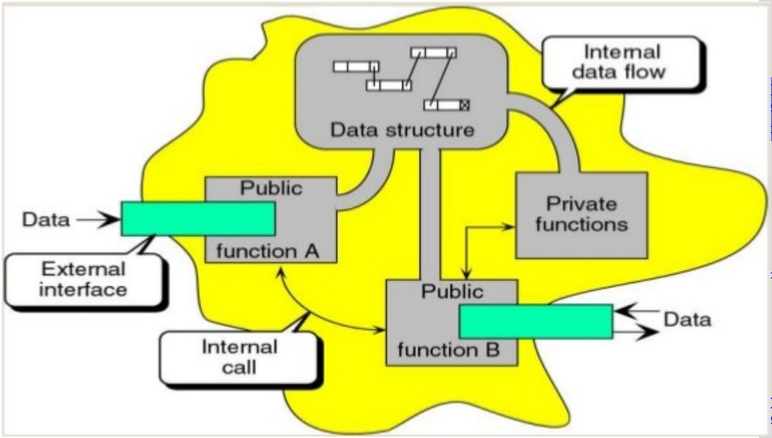


Figure: Abstract data type model (source: Slideshare)

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Example: List

Interface

- **Data**: sequence of elements of a particular data type
- **Operations**: accessing, insertion, deletion

Implementation

- Array
- Linked list

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Algorithm

What is an algorithm?

The logical steps to solve a problem.

What is a program?

Program = Data structures + Algorithms (Niklaus Wirth)

Data structures and Algorithms: Basic concepts
- Algorithm
- Pseudocode
- Data structures
- Classes
- Pointers
- Arrays

Recursion
- Properties of recursion
- Designing recursive algorithms
- Recursion and backtracking
- Recursion implementation in C/C++

Complexity of Algorithms
- Algorithm Efficiency
- Asymptotic Analysis
- Problems and common complexities
- P and NP Problems

Pseudocode

- The most common tool to define algorithms
- English-like representation of the algorithm logic
- Pseudocode = **English** + **code**

relaxed syntax being easy to read

instructions using basic control structures (sequential, conditional, iterative)

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pseudocode

Algorithm Header

- Name
- Parameters and their types
- Purpose: what the algorithm does
- Precondition: precursor requirements for the parameters
- Postcondition: taken action and status of the parameters
- Return condition: returned value

Algorithm Body

- Statements
- Statement numbers: decimal notation to express levels
- Variables: important data
- Algorithm analysis: comments to explain salient points
- Statement constructs: sequence, selection, iteration

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pseudocode: Example

Algorithm average

Pre nothing

Post the average of the input numbers is printed

```
i = 0
sum = 0
while *all numbers not read* do
    i = i + 1
    read number
    sum = sum + number
end
average = sum / i
print average
End average
```

Algorithm 1: How to calculate the average

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures

Data structures can be declared in C++ using the following syntax:

```
struct [type_name] {
   member_type1 member_name1;
   member_type2 member_name2;
   member_type3 member_name3;
   ...
} [object_names];
```

- Where type_name is a name for the structure type, object_names can be a set of valid identifiers for objects that have the type of this structure.
- Within braces { }, there is a list with the data members, each one is specified with a type and a valid identifier as its name.
- **struct** requires either a type_name or at least one name in object_names, but not necessarily both.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures

Example

```
struct car_t {
   int year;
   string brand;
};

car_t toyota;
car_t mercedes, bmw;
```

Example

```
struct {
   int year;
   string brand;
} toyota, mercedes, bmw;
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures

A member of an object can be accessed directly by a dot (.) inserted between the object name and the member name.

Example

```
toyota.year
toyota.brand
mercedes.year
mercedes.brand
bmw.year
bmw.brand
```

- `toyota.year`, `mercedes.year`, and `bmw.year` are of type `int`.
- `toyota.brand`, `mercedes.brand`, and `bmw.brand` are of type `string`.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures

Example

```
// example about structures
#include <iostream>

using namespace std;

struct car_t {
   int year;
   string brand;
} mycar;

int main () {
   mycar.brand = "Audi";
   mycar.year = 2011;
   cout << "My␣favorite␣car␣is:" << endl;
   cout << mycar.brand << "␣(" << mycar.year << ")";
   return 0;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures

Example

```
#include <iostream>
using namespace std;

struct car_t {
   int year;
   string brand;
} mycar;
void printcar(car_t);

int main () {
   mycar.brand = "Audi";
   mycar.year = 2011;
   printcar(mycar);
   return 0;
}
void printcar(car_t c) {
   cout << "My␣favorite␣car␣is:" << endl;
   cout << c.brand << "␣(" << c.year << ")";
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures

Exercise

- Define a data structure `student_t` containing a student's name, firstname and age.
- Write a code in C++ to take input your data and display it.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Data structures

Exercise

```
#include <iostream>
#include <sstream>
using namespace std;
struct student_t {
   string name;
   string firstname;
   int age;
};
void infostudent(student_t);

int main () {
   student_t sv;
   string str;
   cout << "Enter␣your␣name:␣";
   getline (cin, sv.name);
   cout << "Enter␣your␣firstname:␣";
   getline (cin, sv.firstname);
   cout << "Enter␣your␣age:␣";
   getline (cin, str);
   stringstream(str) >> sv.age;
   infostudent(sv);
   return 0;
}
void infostudent(student_t s) {
   cout << "My␣name␣is␣" << s.name << "␣" << s.firstname << endl;
   cout << "I␣am␣" << s.age << "␣years␣old." << endl;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Classes

Classes are defined using keyword `class`, with the following syntax:

```
class class_name {
   access_specifier_1: member1;
   access_specifier_2: member2;
   ...
} object_names;
```

- Where `class_name` is a valid identifier for the class, `object_names` is an optional list of names for objects of this class.
- The body of the declaration can contain `members`, which can either be data or function declarations, and optionally `access_specifiers`.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Classes

Example

```
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area (void);
} rect;
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Classes

Example

```
#include <iostream>
using namespace std;
class Rectangle {
        int width, height;
    public:
        void set_values (int,int);
        int area (void);
};

void Rectangle::set_values (int x, int y) {
    width = x;
    height = y;
}

int Rectangle::area () {
    return width*height;
}

int main () {
    Rectangle rectA, rectB;
    rectA.set_values (3,4);
    rectB.set_values (5,6);
    cout << "rectA area: " << rectA.area() << endl;
    cout << "rectB area: " << rectB.area() << endl;
    return 0;
}
```

Quang-Huan Luu, MsC

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Classes

Constructors

- Automatically called whenever a new object of a class is created.
- Initializing member variables or allocate storage of the object.
- Declared with a name that matches the class name and without any return type; not even `void`.

Example

```
class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area (void);
};
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Classes

Example

```
#include <iostream>
using namespace std;
class Rectangle {
        int width, height;
    public:
        Rectangle (int,int);
        int area (void);
};

Rectangle::Rectangle (int x, int y) {
    width = x;
    height = y;
}

int Rectangle::area () {
    return width*height;
}

int main () {
    Rectangle rectA (3,4);
    Rectangle rectB (5,6);
    cout << "rectA area: " << rectA.area() << endl;
    cout << "rectB area: " << rectB.area() << endl;
    return 0;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Classes

Initialization

- Member initialization:

```
class Rectangle {
   int width;
   const int height;
  public:
   Rectangle(int, int);
   ...
};
Rectangle(int x, int y) : height(y) {
   width = x;
}

int main() {
   Rectangle rectA(3,4);
   ...
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers

Definition

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location.

Address-of operator (&)

The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (**&**), known as `address-of operator`. For example:

```
p = &value;
```

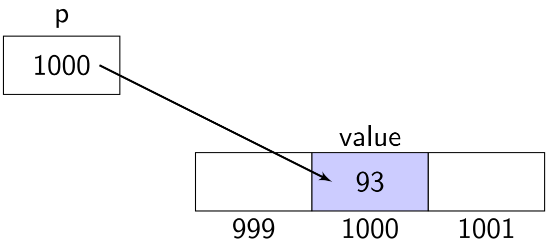
Dereference operator (*)

To access the variable pointed to by a pointer, we precede the pointer name with the `dereference operator` (*****).

```
value = *p;
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers



```
p = &value;
value = *p;
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers

Example

```
int main ()
{
  int v1 = 5, v2 = 15;
  int * p1, * p2;
  p1 = &v1;
  p2 = &v2;
  *p1 = 10;
  *p2 = *p1;
  p1 = p2;
  *p1 = 20;
  cout << "v1␣=␣" << v1 << '\n';
  cout << "v2␣=␣" << v2 << '\n';
  return 0;
}
```

Exercise

What is the output?

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers

Exercise

```
int main ()
{
  int v1 = 5, v2 = 15;
  int * p1, * p2;
  p1 = &v1;     // p1 = address of v1, p1 points to v1
  p2 = &v2;     // p2 = address of v2, p2 points to v2
  *p1 = 10;     // value pointed to by p1 = 10, v1 = 10
  *p2 = *p1;    // value pointed to by p2 = value pointed by p1, v2 = 10
  p1 = p2;      // value of pointer is copied, p1 points to v2
  *p1 = 20;     // value pointed by p1 = 20, v2 = 20
  cout << "v1 = " << v1 << '\n';
  cout << "v2 = " << v2 << '\n';
  return 0;
}
```

Output

```
v1 = 10
v2 = 20
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Arrays

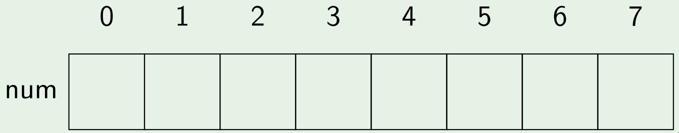
Definition

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by a unique identifier with an index.

```
type var_name[number_of_elements];
```

Example

```
int num[8];
```



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Arrays

Initializing arrays

```
int num[8];

int num[8] = { };

int num[8] = { 1, 2, 3, 5, 8, 13, 21, 34 };

int num[8] = { 1, 2, 3, 5, 8 };

int num[] = { 1, 2, 3, 5, 8, 13, 21, 34 };

int num[] { 1, 2, 3, 5, 8, 13, 21, 34 };
```

Exercise

For each declaration of `num`, what is the output?

```
for (int i=0; i<8; i++) {
    cout << num[i] << endl;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers and arrays

The concept of arrays is related to that of pointers. Arrays work very much like pointers to their first elements, and, actually, an array can always be implicitly converted to the pointer of the proper type.

For example, consider these two declarations:

```
int myarray [10];
int * mypointer;
```

The following assignment operation would be valid:

```
mypointer = myarray;
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers and arrays

Example

```
#include <iostream>
using namespace std;
int main ()
{
  int num[5];
  int * p;
  p = num;  *p = 1;
  p++;  *p = 2;
  p = &num[2];  *p = 3;
  p = num + 3;  *p = 5;
  p = num;  *(p+4) = 8;
  for (int n=0; n<5; n++)
    cout << num[n] << ", ";
  return 0;
}
```

Exercise

What is the output? Explain.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to structures

Structures can be pointed to by its own type of pointers:

```
struct car_t {
  string brand;
  int year;
};

car_t mycar;
car_t * pcar;
```

- `mycar` is an object of structure type `car_t`.
- `pcar` is a pointer to point to an object of structure type `car_t`.

The following code is valid:

```
pcar = &mycar;
```

The value of the pointer `pcar` would be assigned the address of object `mycar`.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to structures

arrow operator (->)

The *arrow operator* (->) is a dereference operator that is used exclusively with pointers to objects that have members. This operator serves to access the member of an object directly from its address.

```
        pcar->year
```

Difference:

- Two expressions pcar->year and (*pcar).year are equivalent, and both access the member year of the data structure pointed by a pointer called pcar.
- Two expressions *mycar.year or *(mycar.year) are equivalent. This would access the value pointed by a hypothetical pointer member called year of the structure object mycar (which is not the case, since year is not a pointer type).

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to structures

Combinations of the operators for pointers and for structure members:

Expression	Equivalent	What is evaluated
a.b		Member b of object a
pa->b	(*pa).b	Member b of object pointed to by pa
*a.b	*(a.b)	Value pointed to by member b of object a

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to structures

Exercise

- Define a data structure `student_t` containing a student's name, firstname and age.
- Write a code in C++ using pointers to structures to take input your data and display it.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to structures

Exercise

```
#include <iostream>
#include <sstream>
using namespace std;
struct student_t {
   string name;
   string firstname;
   int age;
};
void infostudent(student_t*);

int main () {
   student_t sv;
   student_t *psv = &sv;
   string str;
   cout << "Enter your name: ";
   getline (cin, psv->name);
   cout << "Enter your firstname: ";
   getline (cin, psv->firstname);
   cout << "Enter your age: ";
   getline (cin, str);
   stringstream(str) >> psv->age;
   infostudent(psv);
   return 0;
}
void infostudent(student_t *s) {
   cout << "My name is " << s->name << " " << s->firstname << endl;
   cout << "I am " << s->age << " years old." << endl;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to structures

Structures can also be nested in such a way that an element of a structure is itself another structure:

Example

```
struct car_t {
   string brand;
   int year;
};

struct friends_t {
   string name;
   string email;
   car_t favorite_car;
} bobby, tommy;

friends_t *pfriend = &bobby;
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to structures

After the previous declarations, all of the following expressions would be valid:

Example

```
tommy.name
tommy.email
tommy.favorite_car.brand
tommy.favorite_car.year


bobby.name | pfriend->name
bobby.email | pfriend->email
bobby.favorite_car.brand | pfriend->favorite_car.brand
bobby.favorite_car.year | pfriend->favorite_car.year
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Pointers to classes

Example

```
#include <iostream>
using namespace std;
class Rectangle {
      int width, height;
   public:
      Rectangle(int x, int y) : width(x), height(y) {}
      int area(void) { return width * height; }
};

int main () {
   Rectangle rectA (3, 4);
   Rectangle * rectB = &rectA;
   Rectangle * rectC = new Rectangle (5, 6);

   cout << "rectA area: " << rectA.area() << endl;
   cout << "rectB area: " << rectB->area() << endl;
   cout << "rectC area: " << rectC->area() << endl;
   delete rectB;
   delete rectC;
   return 0;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Recursion and the basic components of recursive algorithms

Data structures and Algorithms: Basic concepts

Algorithm

Pseudocode

Data structures

Classes

Pointers

Arrays

Recursion

Properties of recursion

Designing recursive algorithms

Recursion and backtracking

Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency

Asymptotic Analysis

Problems and common complexities

P and NP Problems

Recursion

Definition

Recursion is a repetitive process in which an algorithm calls itself.

- Direct : A $\rightarrow$ A
- Indirect : A $\rightarrow$ B $\rightarrow$ A

Example

Factorial

$$Factorial(n) = \left[\begin{array}{ll} 1 & \text{if } n = 0 \\ n \times (n-1) \times ... \times 2 \times 1 & \text{if } n > 0 \end{array}\right.$$

Using recursion:

$$Factorial(n) = \left[\begin{array}{ll} 1 & \text{if } n = 0 \\ n \times Factorial(n-1) & \text{if } n > 0 \end{array}\right.$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Basic components of recursive algorithms

Two main components of a Recursive Algorithm

1. Base case (i.e. stopping case)
2. General case (i.e. recursive case)

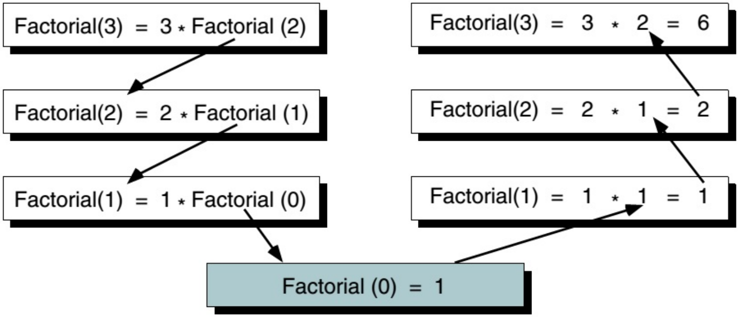
Example

Factorial

$$Factorial(n) = \left[\begin{array}{lll} 1 & \text{if } n = 0 & \text{base} \\ n \times Factorial(n-1) & \text{if } n > 0 & \text{general} \end{array} \right.$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Recursion



Factorial(3) = 3 * Factorial (2)
Factorial(2) = 2 * Factorial (1)
Factorial(1) = 1 * Factorial (0)
Factorial (0) = 1
Factorial(1) = 1 * 1 = 1
Factorial(2) = 2 * 1 = 2
Factorial(3) = 3 * 2 = 6

Hình: Factorial (3) Recursively
(Source: Data Structure - A pseudocode Approach with C++

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Recursion

Factorial: Iterative Solution

```
Algorithm iterativeFactorial(n)
Calculates the factorial of a number using a loop.
Pre: n is the number to be raised factorially
Post: n! is returned - result in factoN

i = 1
factoN = 1
while $i <= n$ do
    factoN = factoN * i
    i = i + 1
end
return factoN
End iterativeFactorial
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Recursion

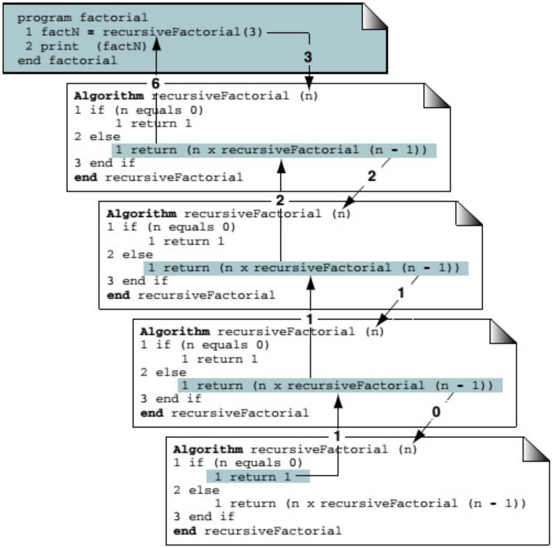
Factorial: Recursive Solution

```
Algorithm recursiveFactorial(n)
Calculates the factorial of a number using a recursion.
Pre: n is the number to be raised factorially
Post: n! is returned

if $n = 0$ then
    return 1
else
    return n * recursiveFactorial(n-1)
end
End recursiveFactorial
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Recursion



```
program factorial
 1 factN = recursiveFactorial(3)
 2 print  (factN)
end factorial
```

```
Algorithm recursiveFactorial (n)
1 if (n equals 0)
     1 return 1
2 else
     1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
```

```
Algorithm recursiveFactorial (n)
1 if (n equals 0)
     1 return 1
2 else
     1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
```

```
Algorithm recursiveFactorial (n)
1 if (n equals 0)
     1 return 1
2 else
     1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
```

```
Algorithm recursiveFactorial (n)
1 if (n equals 0)
     1 return 1
2 else
     1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
```

labelformat=empty

Hình: Calling a Recursive Algorithm (source: Data Structure - A pseudocode Approach with C++)

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Properties of recursion

Data structures and Algorithms: Basic concepts

Algorithm

Pseudocode

Data structures

Classes

Pointers

Arrays

Recursion

Properties of recursion

Designing recursive algorithms

Recursion and backtracking

Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency

Asymptotic Analysis

Problems and common complexities

P and NP Problems

Properties of all recursive algorithms

- A recursive algorithm solves the large problem by using its solution to a simpler sub-problem
- Eventually the sub-problem is simple enough that it can be solved without applying the algorithm to it recursively. $\rightarrow$ This is called the base case.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Designing recursive algorithms

Data structures and Algorithms: Basic concepts

Algorithm

Pseudocode

Data structures

Classes

Pointers

Arrays

Recursion

Properties of recursion

Designing recursive algorithms

Recursion and backtracking

Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency

Asymptotic Analysis

Problems and common complexities

P and NP Problems

The Design Methodology

Every recursive call must either solve a part of the problem or reduce the size of the problem.

Rules for designing a recursive algorithm

1. Determine the base case (stopping case).
2. Then determine the general case (recursive case).
3. Combine the base case and the general cases into an algorithm.

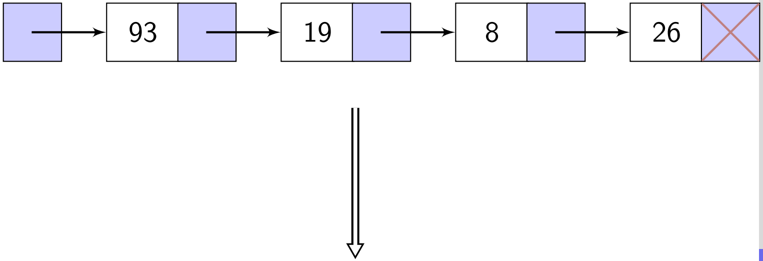
Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Limitations of Recursion

- A recursive algorithm generally runs more slowly than its nonrecursive implementation.
- BUT, the recursive solution shorter and more understandable.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

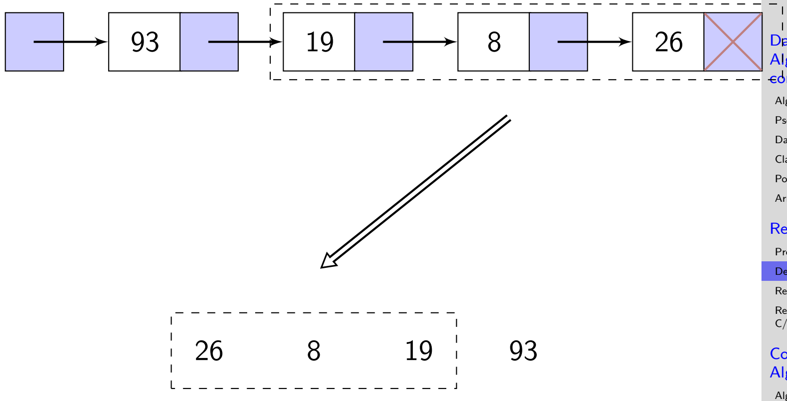
Print List in Reverse



26 8 19 93

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Print List in Reverse



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Print List in Reverse

```
Algorithm printReverse(list)
Prints a linked list in reverse.
Pre: list has been built
Post: list printed in reverse

if list is null then
    return
end
printReverse (list -> next)
print (list -> data)
End printReverse
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Greatest Common Divisor

Definition

$$\gcd(a,b) = \left[\begin{array}{ll} a & \text{if } b = 0 \\ b & \text{if } a = 0 \\ \gcd(b, a \mod b) & \text{otherwise} \end{array}\right.$$

Example

$\gcd(12, 18) = 6$
$\gcd(5, 20) = 5$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Greatest Common Divisor

```
Algorithm gcd(a, b)
Calculates greatest common divisor using the Euclidean
  algorithm.
Pre: a and b are integers
Post: greatest common divisor returned

if $b = 0$ then
    return a
end
if $a = 0$ then
    return b
end
return gcd(b, a mod b)
End gcd
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

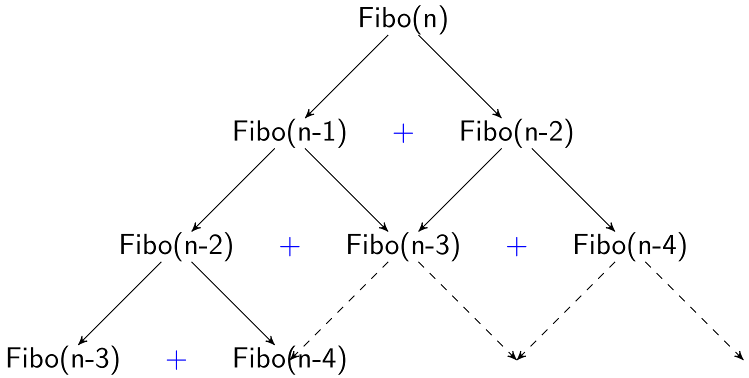
Fibonacci Numbers

Definition

$$Fibo(n) = \left[\begin{array}{ll} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ Fibo(n-1) + Fibo(n-2) & \text{otherwise} \end{array} \right.$$

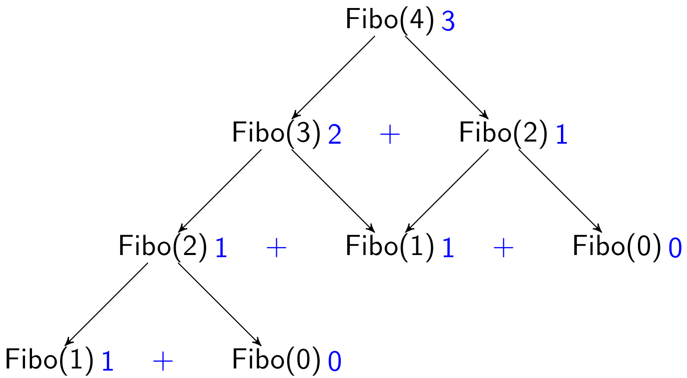
Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Fibonacci Numbers



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Fibonacci Numbers



Result

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Fibonacci Numbers

```
Algorithm Fibo(n)
Calculates the $n^{th}$ Fibonacci number.
Pre: n is postive integer
Post: the $n^{th}$ Fibonnacci number returned

if $n = 0$ or $n = 1$ then
    return n
end
return Fibo(n-1) + Fibo(n-2)
End fib
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Fibonacci Numbers

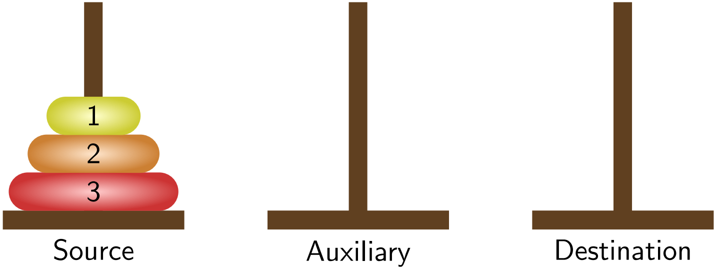
No	Calls	Time	No	Calls	Time
1	1	< 1 sec.	11	287	< 1 sec.
2	3	< 1 sec.	12	465	< 1 sec.
3	5	< 1 sec.	13	753	< 1 sec.
4	9	< 1 sec.	14	1,219	< 1 sec.
5	15	< 1 sec.	15	1,973	< 1 sec.
6	25	< 1 sec.	20	21,891	< 1 sec.
7	41	< 1 sec.	25	242,785	1 sec.
8	67	< 1 sec.	30	2,692,573	7 sec.
9	109	< 1 sec.	35	29,860,703	1 min.
10	177	< 1 sec.	40	331,160,281	13 min.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi

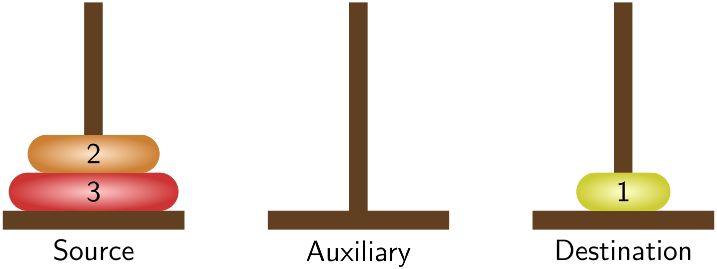
Move disks from Source to Destination using Auxiliary:

1. Only one disk could be moved at a time.
2. A larger disk must never be stacked above a smaller one.
3. Only one auxiliary needle could be used for the intermediate storage of disks.



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi



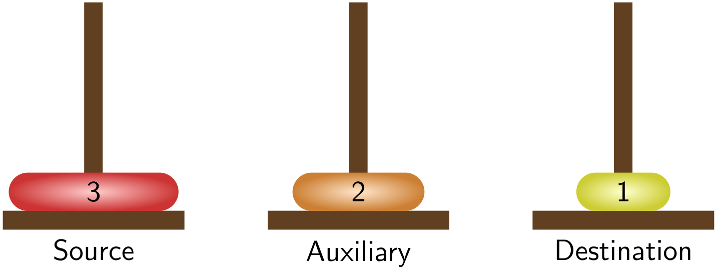
Moved disc from pole 1 to pole 3.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

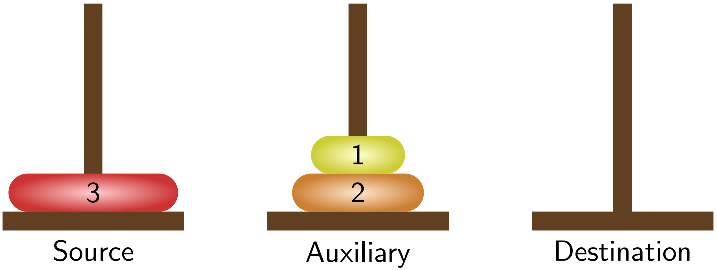
The Towers of Hanoi



Moved disc from pole 1 to pole 2.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi



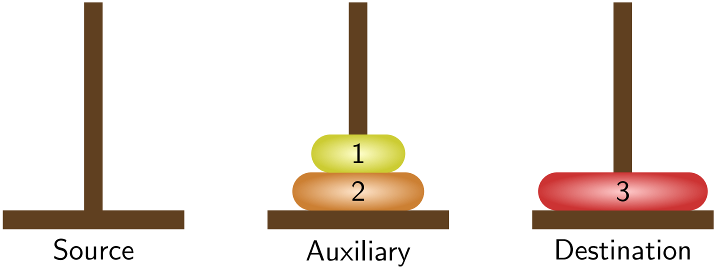
Moved disc from pole 3 to pole 2.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

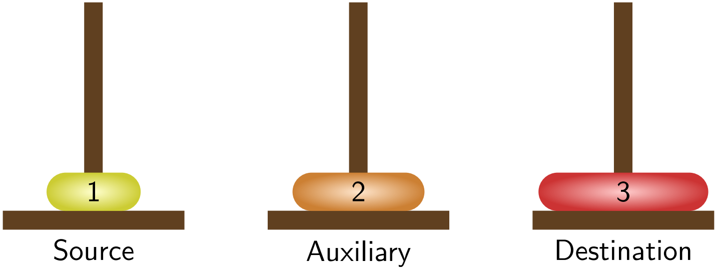
The Towers of Hanoi



Moved disc from pole 1 to pole 3.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

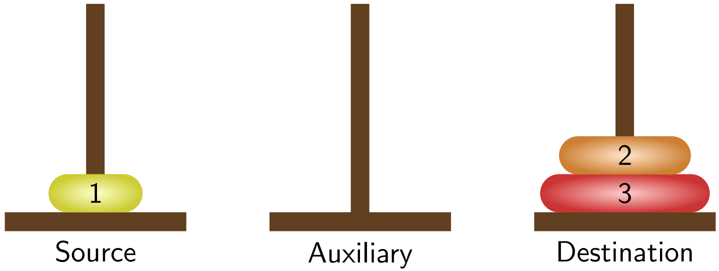
The Towers of Hanoi



Moved disc from pole 2 to pole 1.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

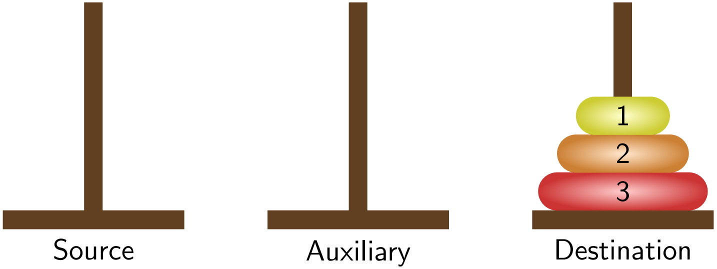
The Towers of Hanoi



Moved disc from pole 2 to pole 3.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

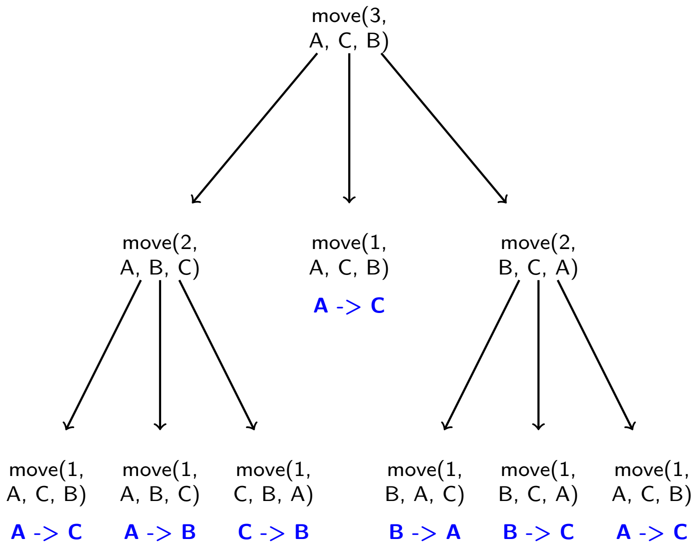
The Towers of Hanoi



Moved disc from pole 1 to pole 3.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi : General

$$\begin{tikzcd}
 & \text{move(n, A, C, B)} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{move(n-1, A, B, C)} & \text{move(1, A, C, B)} & \text{move(n-1, B, C, A)}
\end{tikzcd}$$

Complexity

$$T(n) = 1 + 2T(n-1)$$

Data structures and Algorithms: Basic concepts
- Algorithm
- Pseudocode
- Data structures
- Classes
- Pointers
- Arrays

Recursion
- Properties of recursion
- Designing recursive algorithms
- Recursion and backtracking
- Recursion implementation in C/C++

Complexity of Algorithms
- Algorithm Efficiency
- Asymptotic Analysis
- Problems and common complexities
- P and NP Problems

The Towers of Hanoi

Complexity

$$T(n) = 1 + 2T(n-1)$$
$$=> T(n) = 1 + 2 + 2^2 + ... + 2^{n-1}$$
$$=> T(n) = 2^n - 1$$
$$=> T(n) = O(2^n)$$

- With 64 disks, total number of moves: $2^{64} - 1 \approx 2^4 \times 2^{60} \approx 2^4 \times 10^{18} = 1.6 \times 10^{19}$
- If one move takes 1s, 2^{64} moves take about 5×10^{11} years (500 billions years).

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi

```
Algorithm move(val disks <integer>, val source
  <character>, val destination <character>, val
  auxiliary <character>)
Move disks from source to destination.
Pre: disks is the number of disks to be moved
Post: steps for moves printed
print("Towers: ", disks, source, destination, auxiliary)
if $disks = 1$ then
    print ("Move from", source, "to", destination)
else
    move(disks - 1, source, auxiliary, destination)
    move(1, source, destination, auxiliary)
    move(disks - 1, auxiliary, destination, source)
end
return
End move
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

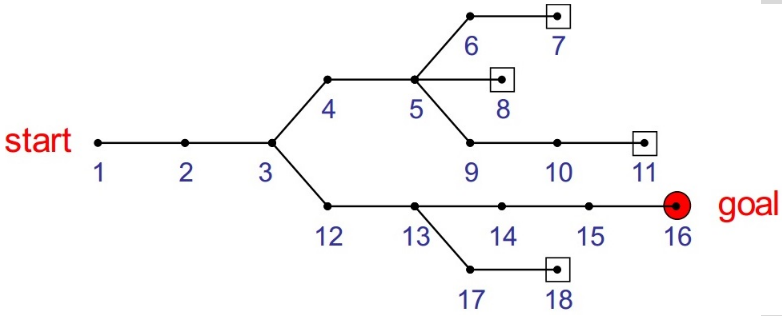
Recursion and backtracking

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Backtracking

Definition

A process to go back to previous steps to try unexplored alternatives.

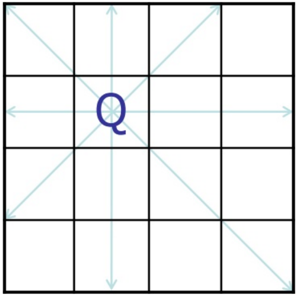


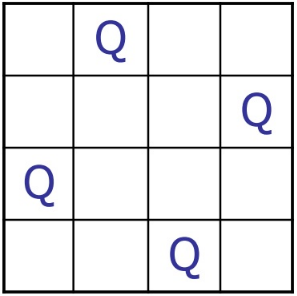
Hình: Goal seeking

Data structures and Algorithms: Basic concepts
lgorithm
seudocode
ata structures
lasses
ointers
rrays
ecursion
roperties of recursion
esigning recursive algorithms
ecursion and backtracking
ecursion implementation in /C++
omplexity of lgorithms
lgorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Eight Queens Problem

Place eight queens on the chess board in such a way that no queen can capture another.





Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Eight Queens Problem

```
1 Algorithm putQueen(ref board <array>,
    val r <integer>)
2 Place remaining queens safely from a row
    of a chess board.

3 Pre: board is nxn array representing a
    chess board
4 r is the row to place queens onwards

5 Post: all the remaining queens are safely
    placed on the board; or backtracking to
    the previous rows is required
```

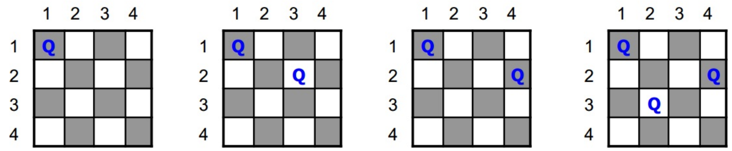
Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Eight Queens Problem

```
for every column c on the same row r do
    if cell r,c is safe then
        place the next queen in cell r,c
        if $r < n-1$ then
            putQueen (board, r + 1)
        else
            output successful placement
        end
        remove the queen from cell r,c
    end
end
return
End putQueen
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Eight Queens Problem



]

Data structures and Algorithms: Basic concepts

Algorithm

Pseudocode

Data structures

Classes

Pointers

Arrays

Recursion

Properties of recursion

Designing recursive algorithms

Recursion and backtracking

Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency

Asymptotic Analysis

Problems and common complexities

P and NP Problems

Recursion implementation in C/C++

Fibonacci Numbers

```
#include <iostream>
using namespace std;

long fib(long num);

int main () {
  int num;
  cout << "What␣Fibonacci␣number
␣␣␣␣␣␣␣␣␣␣do␣you␣want␣to␣calculate?␣";
  cin >> num;
  cout << "The␣" << num << "th␣Fibonacci␣number
␣␣␣␣␣␣␣␣␣␣is:␣" << fib(num) << endl;
  return 0;
}

long fib(long num) {
  if (num == 0 || num == 1)
    return num;
  return fib(num-1) + fib(num-2);
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi

```
#include <iostream>
using namespace std;

void move(int n, char source,
          char destination, char auxiliary);

int main () {
  int numDisks;
  cout << "Please␣enter␣number␣of␣disks:␣";
  cin >> numDisks;
  cout << "Start␣Towers␣of␣Hanoi" << endl;
  move(numDisks, 'A', 'C', 'B');
  return 0;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

The Towers of Hanoi

```
void move(int n, char source,
          char destination, char auxiliary){
  static int step = 0;

  if (n == 1)
    cout << "Step␣" << ++step << ":␣Move␣from␣"
      << source << "␣to␣" << destination << endl;
  else {
    move(n-1, source, auxiliary, destination);
    move(1, source, destination, auxiliary);
    move(n - 1, auxiliary, destination, source);
  }
  return;
}
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Algorithm Efficiency

Data structures and Algorithms: Basic concepts

Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion

Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Compare Algorithms

- Given 2 or more algorithms to solve the same problem, how do we select the best one?
- Some criteria for selecting an algorithm
  - Is it easy to implement, understand, modify?
  - How long does it take to run it to completion?
  - How much of computer memory does it use?
- Software engineering is primarily concerned with the first criteria
- In this course we are interested in the second and third criteria

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

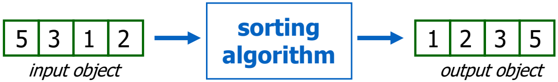
Algorithm Efficiency

- **Time complexity**: the amount of time that an algorithm needs to run to completion.
- **Space complexity**: the amount of memory an algorithm needs to run.
- We will occasionally look at space complexity, but we are mostly interested in time complexity in this course.
- Thus in this course the better algorithm is the one which runs faster (has smaller time complexity).

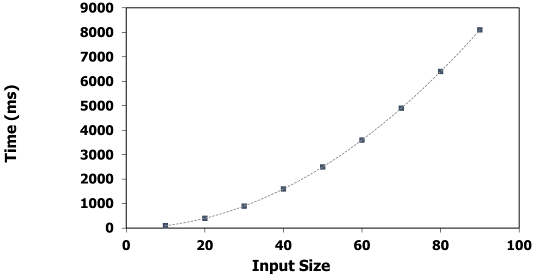
Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

How to Calculate Running time

- Most algorithms transform input objects into output objects.



- The running time of an algorithm typically grows with the input size $f(n)$.



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

How to Calculate Running Time

- Even on inputs of the same size, running time can be very different.
    - Example: algorithm that searches an array containing n integers to find the one with a particular value K (assume that K appears exactly once in the array)
- Idea: analyze running time in the
    - best case
    - worst case
    - average case

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

How to Calculate Running Time

- Best case running time is usually useless
- Average case time is very useful but often difficult to determine
- We focus on the worst case running time
    - Easier to analyze
    - Crucial to applications such as games, finance and robotics

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

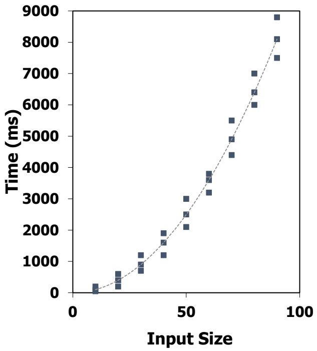
Evaluations of Running Time

- There are two ways to compare running time between agorithms:
    - Experimental evaluation
    - Theoretical evaluation

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Experimental Evaluation of Running Time

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Measure accurately the actual running time
- Plot the results



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment
- In order to compare two algorithms, the same hardware and software environments must be used

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Theoretical Analysis of Running Time

- Uses a pseudo-code description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, **n**
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Primitive Operations

- For theoretical analysis, we will count **primitive** or **basic** operations, which are simple computations performed by an algorithm
- Example:
    - Evaluating an expression: $x^2 + 3x$
    - Assigning a value to a variable: $x = y$
    - Indexing into an array: $a[10]$
    - Calling a function: $mySwap(a, b)$
    - Returing from a function: $return(x)$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Counting Primitive Operations

- By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size

Algorithm findMax(a, n)

```
currentMax = a[0]              //2
i = 1                          //2
while (i < n)                  //n
  if (currentMax < a[i])       //2n-2
    currentMax = a[i]          //2n-2
  i = i + 1                    //2n-2
return currentMax              //1
```

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Estimating Running Time

- Algorithm findMax() executes $7n - 1$ primitive operations in the worst case. Define:
    - a = Time taken by the fastest primitive operation
    - b = Time taken by the slowest primitive operation
- Let f(n) be worst-case time of arrayMax. Then $a(7n - 1) \leq f(n) \leq b(7n - 1)$
- Hence, the running time f(n) is bounded by two linear functions

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Growth Rate of Running Time

- Changing the hardware/software environment
  - Affects $f(n)$ by a constant factor, but
  - Does not alter the growth rate of $f(n)$
- Thus we focus on the big-picture which is the growth rate of an algorithm
- The linear growth rate of the running time $f(n)$ is an intrinsic property of algorithm findMax()

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Linear Loops

```
for (i = 0; i < n; i++)
   application code
```

The number of times the body of the loop is replicated is n.

$$\mathbf{f(n) = n}$$

```
for (i = 0; i < n; i += 2)
   application code
```

The number of times the body of the loop is replicated is $n/2$.

$$\mathbf{f(n) = n/2}$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Logarithmic Loops

Multiply loops

```
i = 1
while (i <= n)
   application code
   i = i x 2
```

Divide loops

```
i = n
while (i >= 1)
   application code
   i = i / 2
```

The number of times the body of the loop is replicated is

$$\mathbf{f(n)} = \log_2 n$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Nested Loops

Iterations = Outer loop iterations × Inner loop iterations

Example

```
i = 1
while (i <= n)
   j = 1
   while (j <= n)
      application code
      j = j * 2
   i = i + 1
```

The number of times the body of the loop is replicated is

$$f(n) = n \log_2 n$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Quadratic Loops

Example

```
i = 1
while (i <= n)
   j = 1
   while (j <= n)
      application code
      j = j + 1
   i = i + 1
```

The number of times the body of the loop is replicated is

$$f(n) = n^2$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Dependent Quadratic Loops

Example

```
i = 1
while (i <= n)
   j = 1
   while (j <= i)
      application code
      j = j + 1
   i = i + 1
```

The number of times the body of the loop is replicated is

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

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Constant Factors

- The growth rate is not affected by
  - constant factors or
  - lower-order terms
- Examples
  - $10^3n + 10^5$ is a linear function
  - $10n^2 + 10^4n$ is a quadratic function
- How do we get rid of the constant factors to focus on the essential part of the running time?

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Asymptotic Analysis

- Algorithm efficiency is considered with only big problem sizes.
- We are not concerned with an exact measurement of an algorithm's efficiency.
- Terms that do not substantially change the function's magnitude are eliminated.

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Asymptotic Analysis

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Asymptotic Notations

There are three common notations for asymptotic analysis:

- Big-Oh: $O(.)$
- Big-Omega: $\Omega(.)$
- Big-Theta: $\Theta(.)$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Big-Oh notation

- The big-Oh notation is used widely to characterize running times and space bounds
- The big-Oh notation allows us to ignore constant factors and lower order terms and focus on the main components of a function which affect its growth

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Big-Oh notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that: $f(n) \leq c.g(n)$ for $n \geq n_0$
- Example: $2n + 10$ is $O(n)$
- Why?

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Big-Oh Rules

Example

$f(n) = c.n \Rightarrow f(n) = O(n)$
$f(n) = n(n+1)/2 = n^2/2 + n/2 \Rightarrow f(n) = O(n^2)$

- Set the coefficient of the term to one.
- Keep the largest term and discard the others.

Some example of Big-O:
$1 \quad \log_2 n \quad n \quad n\log_2 n \quad n^2 \ldots n^k \ldots 2^n \quad n!$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Big-Oh and Growth Rate

- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Standard Measures of Efficiency

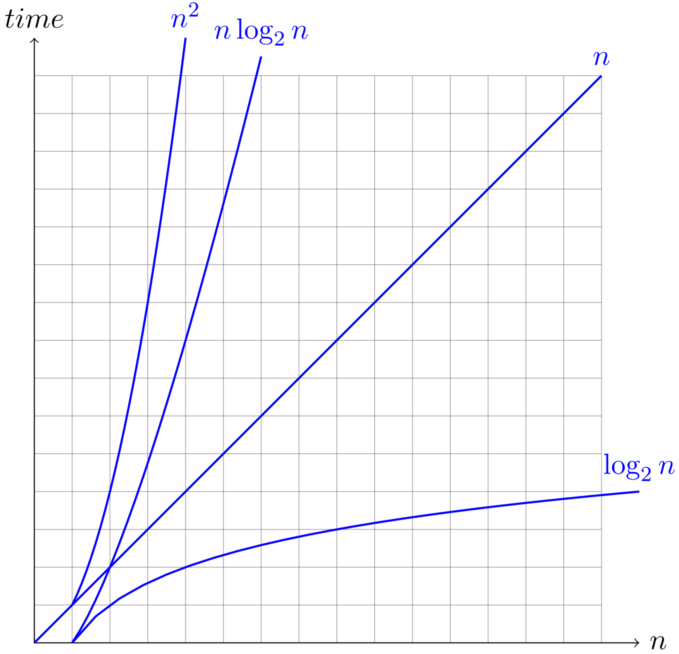
Efficiency	Big-O	Iterations	Est. Time
logarithmic	O($\log_2 n$)	14	microseconds
linear	O(n)	10 000	0.1 seconds
linear log	O($n \log_2 n$)	140 000	2 seconds
quadratic	O(n^2)	10000^2	15-20 min.
polynomial	O(n^k)	10000^k	hours
exponential	O(2^n)	2^{10000}	intractable
factorial	O($n!$)	$10000!$	intractable

Assume instruction speed of 1 microsecond and 10 instructions in loop.

$n = 10000$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Standard Measures of Efficiency



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Relatives of Big-Oh

- big-Omega
  - $f(n)$ is $\Omega(g(n))$ if there is a constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq c.g(n)$ for $n \geq n_0$
- big-Theta
  - $f(n)$ is $\Theta(g(n))$ if there are constants $c' > 0$ and $c'' > 0$ and an integer constant $n_0 \geq 1$ such that $c'.g(n) \geq f(n) \geq c''.g(n)$ for $n \geq n_0$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Intuition for Asymptotic Notation

- Big-Oh: $f(n)$ is $O(g(n))$ if $f(n)$ is asymptotically less than or equal to $g(n)$
- Big-Omega: $f(n)$ is $\Omega(g(n))$ if $f(n)$ is asymptotically greater than or equal to $g(n)$
- Big-Theta: $f(n)$ is $\Theta(g(n))$ if $f(n)$ is asymptotically equal to $g(n)$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Problems and common complexities

Data structures and Algorithms: Basic concepts

Algorithm

Pseudocode

Data structures

Classes

Pointers

Arrays

Recursion

Properties of recursion

Designing recursive algorithms

Recursion and backtracking

Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency

Asymptotic Analysis

Problems and common complexities

P and NP Problems

Binary search

Recurrence Equation (Phương trình hồi quy)

An equation or inequality that describes a function in terms of its value on smaller input.

1	2	3	5	8	13	21	34	55	89

$$T(n) = 1 + T(n/2) \Rightarrow T(n) = O(\log_2 n)$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Binary search

- Best case: when the number of steps is smallest. $T(n) = O(1)$
- Worst case: when the number of steps is largest. $T(n) = O(\log_2 n)$
- Average case: in between. $T(n) = O(\log_2 n)$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Sequential search

8	5	21	2	1	13	4	34	7	18

- Best case: $T(n) = O(1)$
- Worst case: $T(n) = O(n)$
- Average case: $T(n) = \sum_{i=1}^{n} i.p_i$
  p_i : probability for the target being at a[i]
  $p_i = 1/n \Rightarrow T(n) = (\sum_{i=1}^{n} i)/n = O(n(n+1)/2n) = O(n)$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Quick sort

19	8	3	15	28	10	22	4	12	83

Recurrence Equation

$$T(n) = O(n) + 2T(n/2)$$

- Best case: $T(n) = O(n \log_2 n)$
- Worst case: $T(n) = O(n^2)$
- Average case: $T(n) = O(n \log_2 n)$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

P and NP Problems

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays

Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++

Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

P and NP Problems

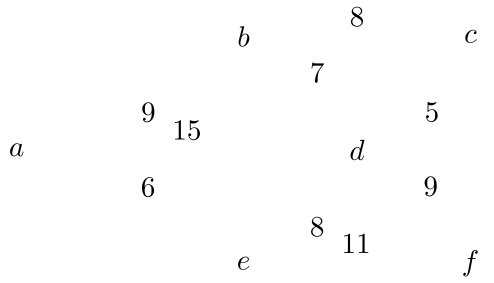
- P: Polynomial (can be solved in polynomial time on a deterministic machine).
- NP: Nondeterministic Polynomial (can be solved in polynomial time on a non-deterministic machine).

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

P and NP Problems

Travelling Salesman Problem:

A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list.
Find the route the salesman should follow for the shortest possible round trip that both starts and finishes at any one of the cities.

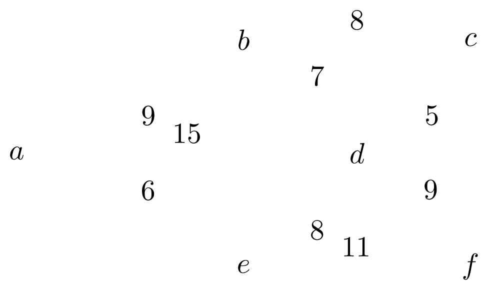


Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

P and NP Problems

Travelling Salesman Problem:

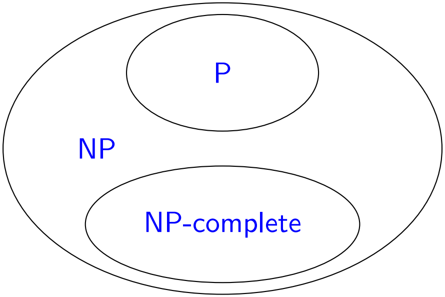
Deterministic machine:
$f(n) = n(n-1)(n-2)\dots 1 = O(n!)$
$\Rightarrow$ NP problem



Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

P and NP Problems

NP-complete: NP and every other problem in NP is polynomially reducible to it.



$$P = NP?$$

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Read more on

Data Structures and Algorithm Analysis Edition 3.2 (C++ Version)

Part I, Chapter 1, Chapter 2, Chapter 3

Data structures and Algorithms: Basic concepts
Algorithm
Pseudocode
Data structures
Classes
Pointers
Arrays
Recursion
Properties of recursion
Designing recursive algorithms
Recursion and backtracking
Recursion implementation in C/C++
Complexity of Algorithms
Algorithm Efficiency
Asymptotic Analysis
Problems and common complexities
P and NP Problems

Contact

Quang-Huan Luu, MSC

huanlq@hcmut.edu.vn

Data structures and Algorithms: Basic concepts

Algorithm

Pseudocode

Data structures

Classes

Pointers

Arrays

Recursion

Properties of recursion

Designing recursive algorithms

Recursion and backtracking

Recursion implementation in C/C++

Complexity of Algorithms

Algorithm Efficiency

Asymptotic Analysis

Problems and common complexities

P and NP Problems