

Welcome

Design and Analysis of Algorithms

CS404/504

Razvan Bunescu

School of EECS

bunescu@ohio.edu

# Course Description

## Course Description:

This course provides an introduction to the modern study of computer algorithms. Through this course students should be able to:

- 1) Analyze algorithm performance using complexity measurement.
- 2) Master major algorithms design techniques such as divide and conquer, greedy and dynamic programming.
- 3) Apply above approaches to solve a variety of practical problems such as sorting and selection, graph problems, and other optimization problems.
- 4) Understand the theory of NP-completeness.

# What is an algorithm?

**Definition:** An **algorithm** is a computational procedure that takes values as *input* and produces values as *output*, in order to solve a well defined *computational problem*.

- The statement of the *problem* specifies a desired relationship between the *input* and the *output*.
- The algorithm specifies how to achieve that *input/output* relationship.
- A particular value of the *input* corresponds to an instance of the *problem*.

# How to describe an algorithm?

## We use Pseudo code:

1) **assignment** statement:

variable := value

2) **for loop:**

```
for variable := value1 to value2 do {  
    <statement 1>;  
    <statement 2>;  
    ...  
}
```

3) **while loop:**

```
while <condition> do {  
    <statement 1>;  
    <statement 2>;  
    ..  
}
```

4) **repeat loop:**

```
repeat {  
    <statement 1>;  
    <statement 2>;  
    ..  
} until < condition >
```

# How to describe an algorithm?

## We use Pseudo code:

5) **if-then** statement:

```
if < condition > {  
    <statement 1>;  
    <statement 2>;  
    ..  
}
```

6) **if-then-else** statement:

```
if < condition > {  
    <statement 1>;  
    <statement 2>;  
    ..  
} else {  
    <statement 1'>;  
    <statement 2'>;  
    ..  
}
```

## Examples

1) **Find the maximum:** given an array of  $n$  numbers  $a[1..n]$ , what is the maximum?

```
algorithm max(a, n)
{
    max := a[1];
    for i:= 2 to n do
        if (a[i] > max)
            max := a[i];

    return max;
}
```

## Examples, cont'd

2) **Calculate the sum:** given an array of  $n$  numbers  $a[1..n]$ , what is their sum?

```
algorithm sum(a, n)
{
    result := a[1];
    for i := 2 to n do
        result := a[i] + result;

    return result;
}
```

## Examples, cont'd

### 3) **Sorting:**

Input: a sequence of  $n$  numbers  $a_1, a_2 \dots a_n$ .

Output: a reordering of the input sequence such that in the resulting sequence each number is smaller than all the numbers before, and larger than the numbers after.

Output: a permutation  $\pi$  s.t.  $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$

# What kind of algorithms are we looking for?

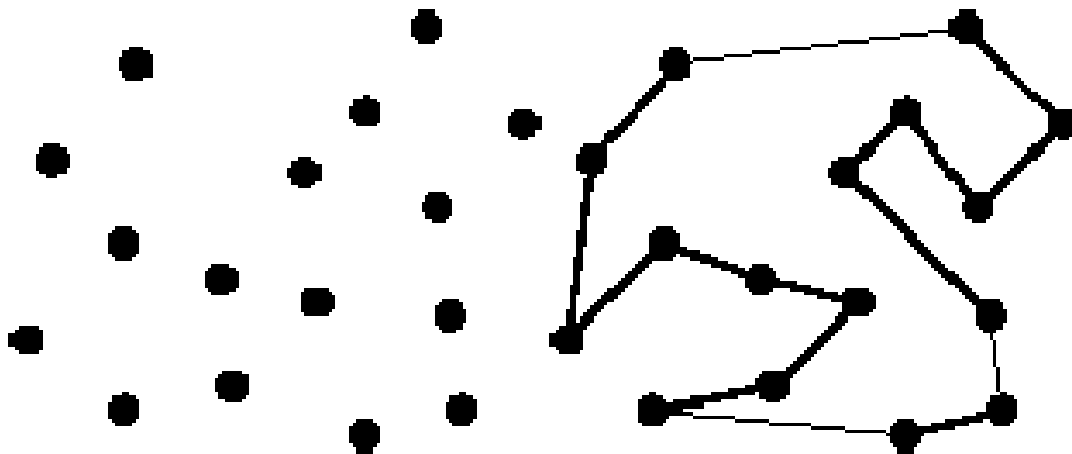
## 1. **Correct.**

How to prove an algorithm is correct/incorrect?

## 2. **Efficient.** Including time and space efficiency.

**Correctness is not obvious!**

An example: Travelling Salesman Problem (TSP)



## Nearest Neighbor Tour

Algorithm:  $\text{NNT}(\mathcal{P} = \{P_1, \dots, P_n\})$

Pick and visit an initial vertex  $P_i$

Set current vertex  $P$  to  $P_i$

*while* there are still unvisited vertices in  $\mathcal{P}$  *do*

    Let  $V$  be the closest vertex to  $P$  that is unvisited

    Visit  $V$

    Set current vertex  $P$  to  $V$

Return to  $P_i$  from  $P$

## Is that correct?

What if the input is as follows?

## A correct solution

- We could try all possible orderings of the points, then select the ordering which minimizes the total length.
- **How many possibilities do we need to check?  $N!$**

**How big is  $N!$ ?**

When  $N = 10$ ,  $N! = 3628800$ ,  $N = 20$ ,  
 $N! = 2.4329 * 10^{18}$ ,  $N = 100$ ,  $N! = 9.32 * 10^{157}$ .