Introduction to Engineering Using Robotics Laboratories

Algorithms

Dr. Yinong Chen

Roadmap

- The Concept of Algorithms
- Algorithm Primitives
- Algorithm Complexity
- Examples of Algorithms
- Robotics Algorithms

What is Computer Science?

- is the study of the **theoretical** foundations of **information** (**data**) and computation, and of **practical** techniques for their implementation and application in computer systems;
- is frequently described as the systematic study of algorithmic processes (**algorithms**) that describe and transform information.
- answers the fundamental question:

  What can be (efficiently) automated?

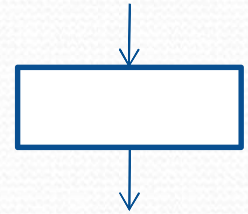
Definition of Algorithms

- An **algorithm** is an ordered set of unambiguous, steps (primitives) that defines a terminating process.
- An algorithm needs to
  - be correct: meet the specification
  - terminate : deliver the result in limited time
    - Computable in limited steps
  - Be efficient
    - Efficient: Computation time in a polynomial function of input size; For example: $T(n) = n^3$
    - Not efficient: Computation time is an exponential function of input size; For example: $T(n) = 2^n$

Pseudo code Primitives

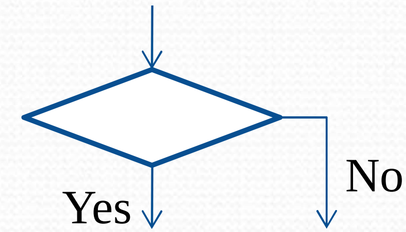
- Assignment

  name ← *expression*



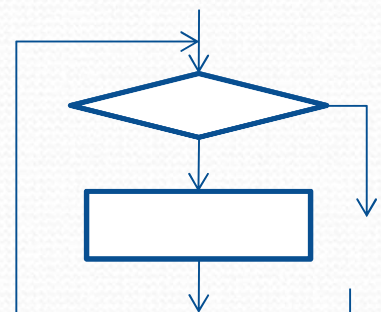
- Conditional selection

  if *condition* **then** *actions*



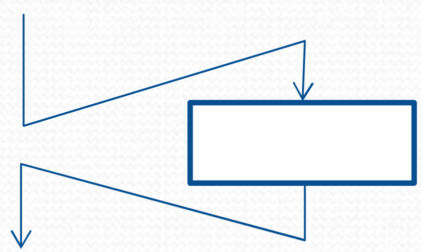
- Repeated execution

  while *condition* **do** *actions*



- Procedure

  procedure *name* (*generic names*)
  actions / activities



A procedure is a block of pseudo code

```
Procedure CountTo10  // activity in VPL
Count ← 0;
While (Count < 10) do
{
        print "The number is " and Count);
        Count ← Count + 1;
}
```

Algorithm Complexity Measurement

Worst-case: (usually)

- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (NEVER)

- Cheat with a slow algorithm that works fast on *some* input.

Algorithm Complexity Considerations

- The real execution time depends on the input: An already sorted sequence is easier to sort. Thus, algorithm analysis considers the worse case or the average case;
- The execution time depends on the input size. Sorting 10 numbers takes longer than sorting 5 numbers. Thus, the input size *n* is considered a parameter (variable);
- Any problem of small size can be easily solved, and thus, algorithm analysis focuses on the execution time when the size is large;
- Execution time is machine-dependent. Algorithm analysis calculates the steps (operations) needed, instead of the time.

Weight Lifting Competition

Problem Definition

***Input*:** **Given a list of numbers, representing the weights lifted by players**

***Output*:** **Find the largest weight, representing the winner**

Player1: W1 = input a number from keyboard

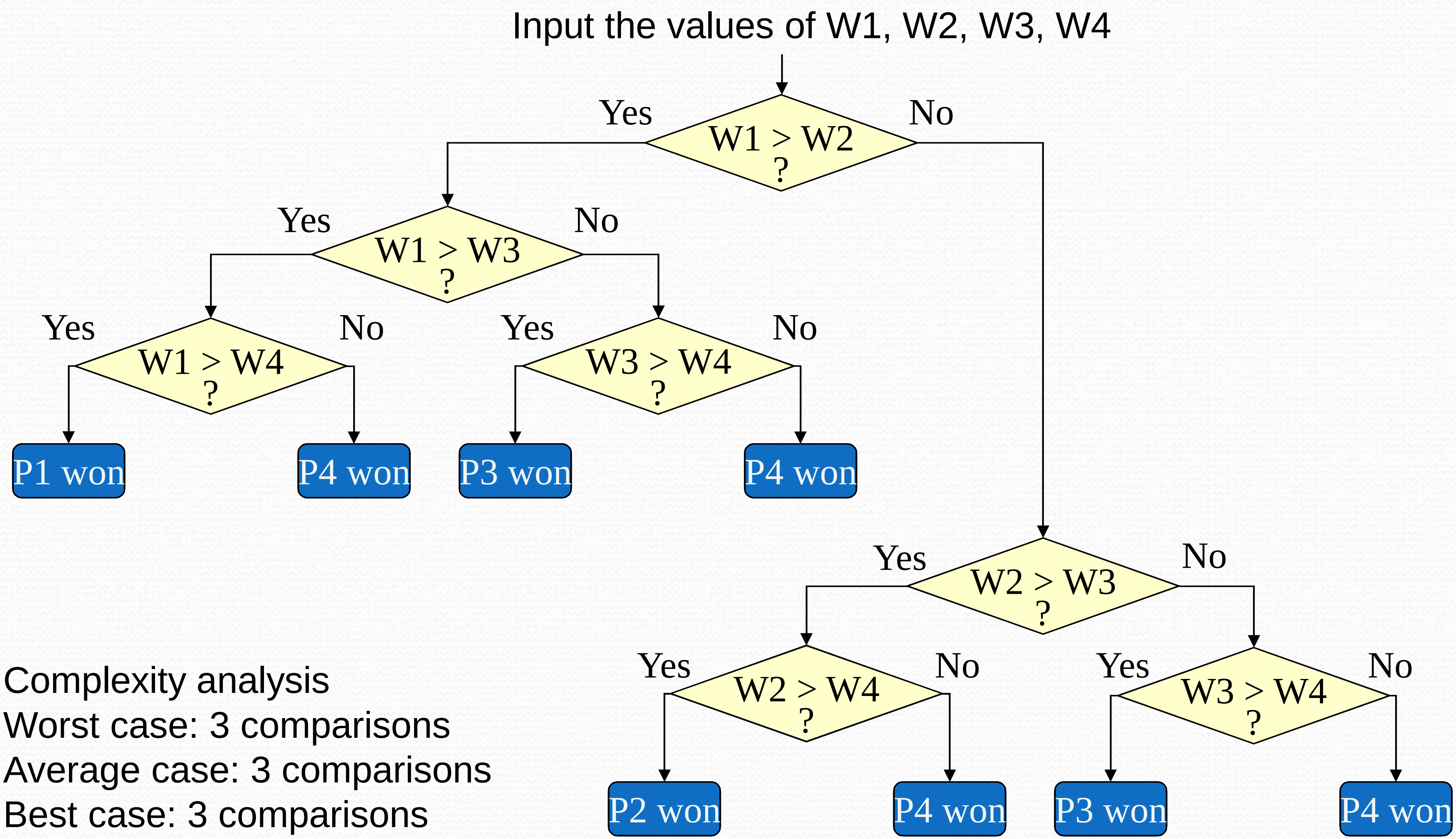
Player2: W2 = input a number from keyboard

Player3: W3 = input a number from keyboard

Player4: W4 = input a number from keyboard

Algorithm 1 (Flowchart)

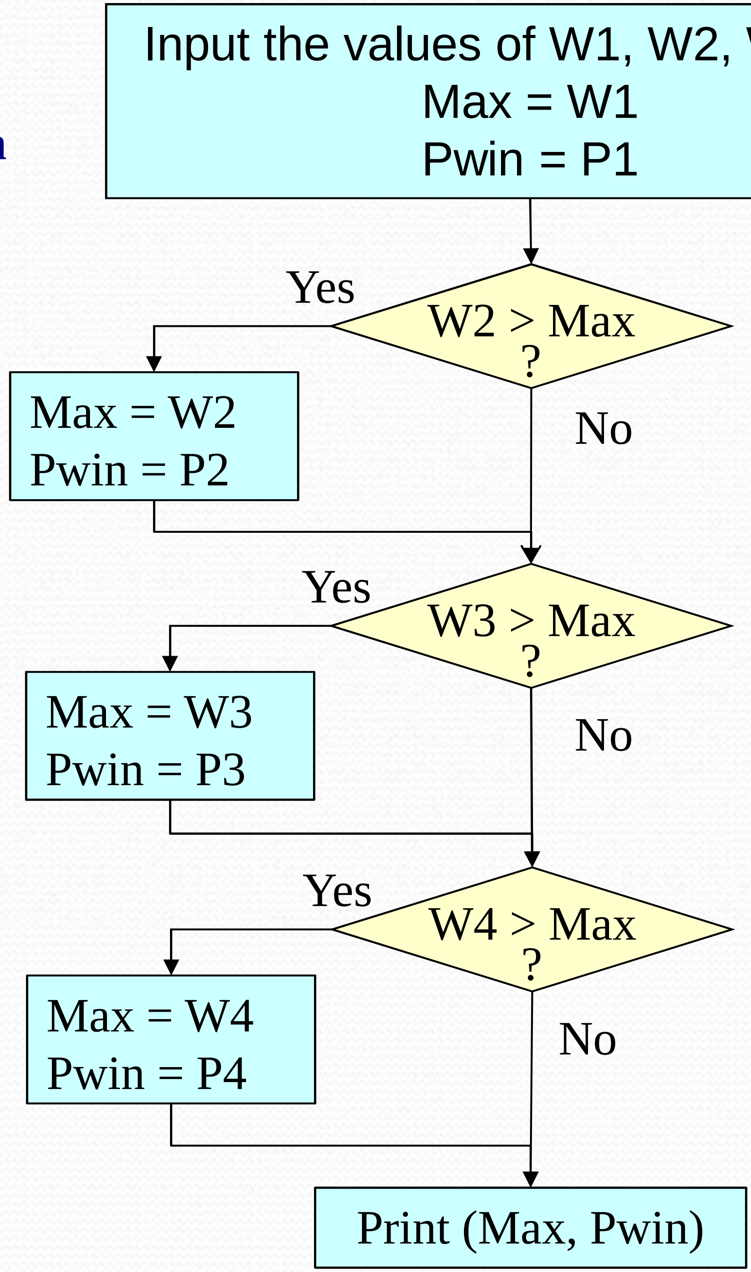
Find the largest number, given four input numbers



Complexity analysis
Worst case: 3 comparisons
Average case: 3 comparisons
Best case: 3 comparisons

Algorithm 2 (Flowchart)

Put the largest weight in Max;
Put the player with the max weight in Pwin



Complexity analysis

Best case:
- 3 comparisons and 2 assignments

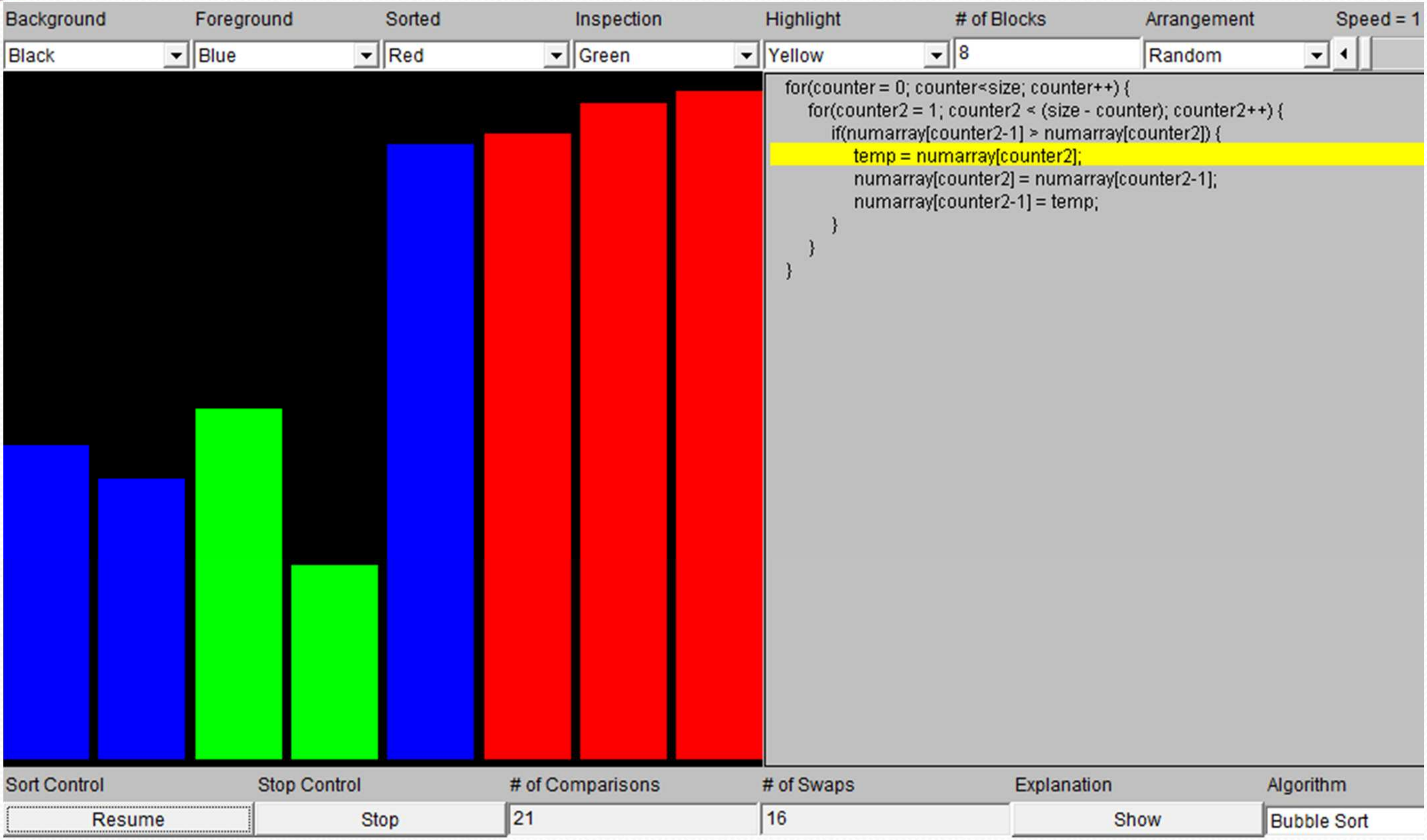
Worst case:
- 3 comparisons and 8 assignments

Average case:
- 3 comparisons and 5 assignments

2+
4+4+4+
6+6+6+
8 = 40
→ 40/8 = 5

Algorithms sorting numbers: Bubble Sort

http://www.cs.hope.edu/~dershem/alganim/animator/Animator.html

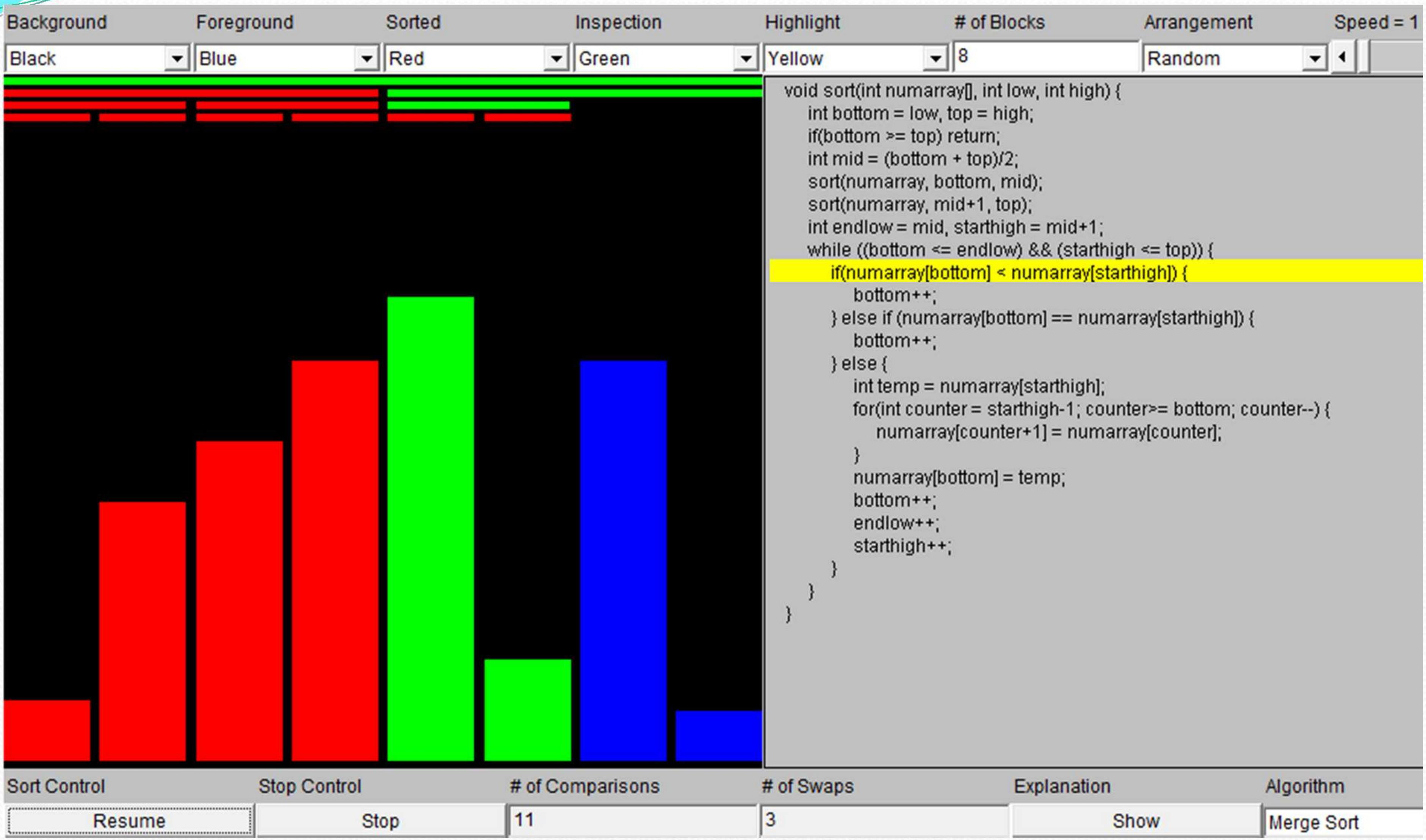


To sort 8 numbers, it takes 28 comparisons and 19 swaps.

To sort 80 numbers, it takes 3160 comparisons and 1469 swaps.

Algorithms sorting numbers: Merge Sort

http://www.cs.hope.edu/~dershem/alganim/animator/Animator.html



To sort 8 numbers, it takes 32comparisons and 9 swaps.

To sort 80 numbers, it takes 800 comparisons and 195 swaps.

Algorithm Complexity Analysis

It concerns the time (number of operations) and space (memory) used when the problem size is large. It is not a concern when the size is small.

The **big-O notation** is used to estimate the upper bound of the complexity.

CSE205: Basic Algorithm Design

CSE310: Algorithm Design and Complexity Analysis

Bubble Sort:

To sort n = 8 numbers, it takes 28 comparisons and 19 swaps.

To sort n = 80 numbers, it takes **3160** comparisons and **1469** swaps.

Complexity = $O(n^2)$

Big-O notation: Upper bound

Merge Sort:

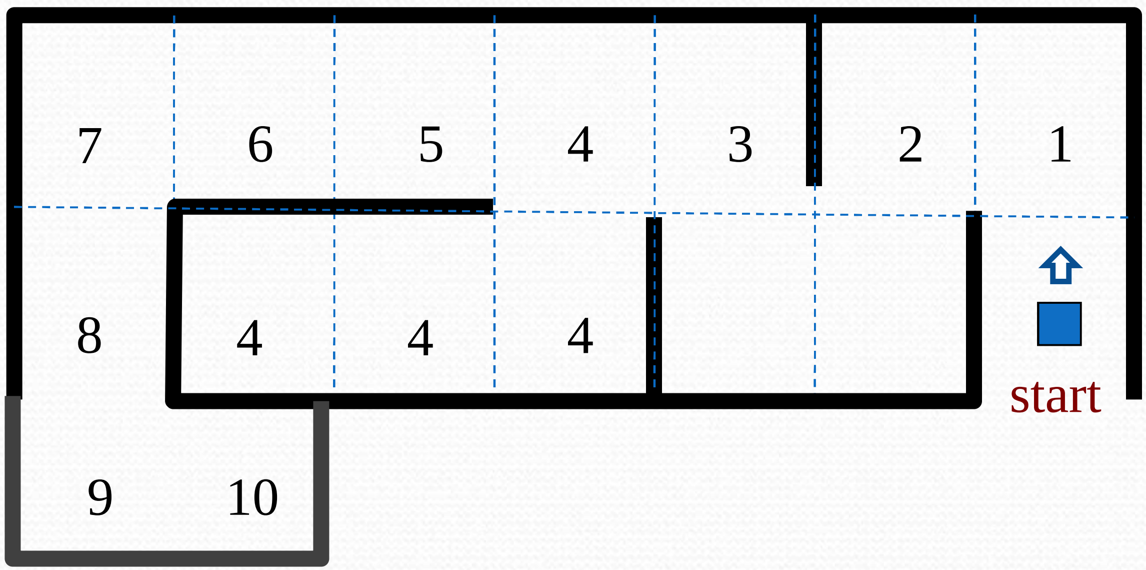
To sort n = 8 numbers, it takes 32 comparisons and 9 swaps.

To sort n = 80 numbers, it takes **800** comparisons and **195** swaps.

Complexity = $O(n \log n)$

Big-O notation: Upper bound

The Complexity of the Maze Algorithms



- Use the number of turns or degrees, and the units of distance needed to travel from start to exit;
- Evaluate different algorithms (Lab 7 manual)
  - Random Algorithm
  - Wall-Following Algorithm
  - Heuristic Algorithm of Local Best Decision
  - Greedy Algorithm based on the First Working Solution
  - Hard Coding

Autonomous Maze Traversing Algorithm

1. The robot is in state "Forward" and moves forward;
2. If the distance measured by the range sensor is less than 400 millimeter, it turns (90 degree) right;
3. After the event "rightFinished" occurs, it saves the distance measured to the variable RightDistance;
4. The robot then spins 180 degree left to measure the distance on the other side;
5. After the event "leftFinished" occurs, it compares the distance measured with the values saved in the variable RightDistance;
6. If the current distance is longer, it transits to the state "Forward" to move forward;
7. Otherwise, it resumes (spins 180 degree) to the other direction;
8. Then, it transits to step 1: to move forward.

Wall-Following Algorithm

1. Variable DV = Ultrasonic sensor measure;
2. The robot repeat all the following steps in a loop, until the touch sensor is pressed;
   1) Robot moves forward;
   2) Robot keeps measures the left-distance in certain interval, and it compares the newly measured distance with the distance stored in variable DV.
   3) If the distance measured is greater than DV + 1, turns one degree left, and then returns to step 2;
   4) If the distance measured is less than DV - 1, the robot turns one degree right, and then returns to step 2;
   5) If the distance measured greater than DV + 5, turns 90 degree right, and then returns to step 2;
   6) Returns to step 2;
3. Touch sensor is pressed; robot moves backward 0.5 rotations, and then turns left 90 degree;
4. Return to step 2.

Complexity of the Robotics Algorithms

- Dealing with the computational steps and mechanic steps representing the robot’s move.
- Which part is more time consuming?
  - Degrees of turning
  - Distance traveled