Lecture 01

Introduction to Data Structures and Algorithms

DATA STRUCTURES AND ALGORITHMS

Fall 2017

University of Windsor

By Dr. Sherif Saad

Agenda

- What is an Algorithm?
- Expressing an Algorithm
- Algorithm Analysis and Big-O Notation
- Data Structures and ADTs
- Arrays and Arrays Operations

Learning Outcomes

By the end of this class you should know on how to

- Define data structures and algorithms
- Describe the properties of a good algorithm
- Explain the goals of algorithm analysis and the importance of algorithm analysis.
- Explain the difference between linear and nonlinear data structures.

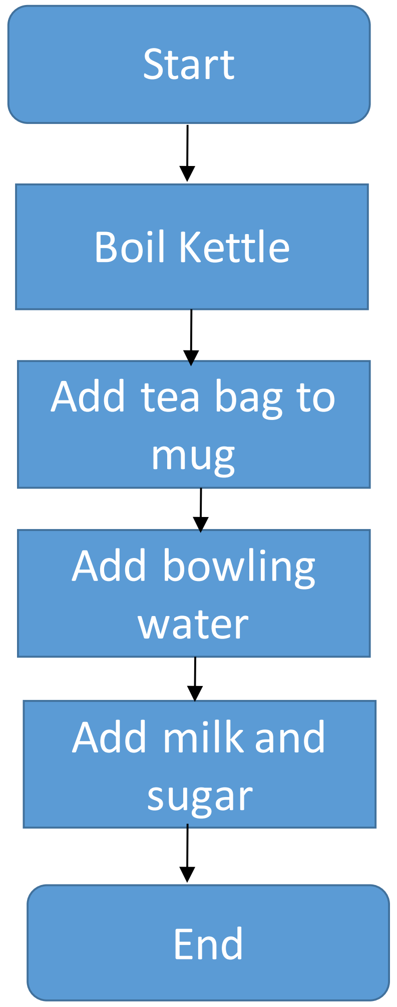
What is an Algorithm?

- The word 'algorithm' is a combination of the Latin word algorismus, named after ***Muhammad ibn Musa al-Khwarizmi***, a 9th-century Persian mathematician and the Greek word arithmos, meaning "number". [Wikipedia]
- An algorithm is a step by step (unambiguous) instructions to solve a given problem.
- An algorithm is a set of precise instructions that transfer the input into the desired output.
- To express an algorithm we could use ***natural language***, ***pseudocode***, ***flowcharts***, or ***programming language***.

Expressing Algorithm using Natural Language

- Making a cup of tea in natural language
  - Find a kettle and add water by running the tap a little, so the water's nicely aerated, and only boil it once to keep the oxygen level up.
  - Pop a tea bag into your mug and pour the hot water and stir briefly, wait for about four minutes for the tea to unlock its flavour.
  - Before removing the tea bag, gently squeeze it against the side of the mug.
  - Finally, customise your tea by adding milk, sugar or honey or nothing at all.

Expressing Algorithm using Flowchart



Expressing Algorithm in Natural Language

Sorting an array of N elements:

1. Starts by comparing the first two elements of an array and swapping if necessary.
2. if you want to sort the elements of the array in ascending order and if the first element is greater than the second then, you need to swap the elements but, if the first element is smaller than the second, you must not swap the element.
3. Then, again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped. This completes the first step of the sort.
4. If there are n elements to be sorted then, the process mentioned above should be repeated n-1 times to get required result

Algorithm in Pseudocode

Sorting an array of N elements:

Input: array of size N

output: sorted array of size N

```
For I = 0 to N - 2
    For J = 0 to N - 2
        If (A(J) > A(J + 1)
            Temp = A(J)
            A(J) = A(J + 1)
            A(J + 1) = Temp
        End-If
    End-For
End-For
```

What is a good algorithm?

Any algorithm should have five properties:

- Precision: an algorithm should have clear and well-defined instructions
- Uniqueness: each step taken in the algorithm should give a definite result
- Feasibility: the algorithm should be practicable in real life.
- Finiteness: the algorithm stops after a finite number of instructions are executed
- Generality: the algorithm applies to a set of inputs

How do we judge an algorithm?

- In general, we use two criteria to judge/evaluate an algorithm. These two criteria are the correctness and efficiency.
- Algorithm correctness: does the algorithm provide the desired solution to the problem in a finite number of steps.
- Algorithm efficiency: how much resources (in term of time and memory) the algorithm consumes to produce the solution.

Algorithm Analysis

What is algorithm analysis?

- Algorithm analysis focuses on estimating the resources mainly the time and memory resources required by the algorithm to solve a given computational problem.

What is the goal of algorithm analysis?

- The goal of algorithm analysis is to compare algorithms to determine which algorithm is the most efficient to solve a particular problem.

Algorithm Analysis(cont…)

What is time complexity analysis?

- It is the process of determining how processing time increase as the size of the problem increases.
- It is a function describing the amount of time the algorithm will take for a given number of elements in the input (e.g. size of an array, size of an image, number of rows in a database table)
- The time complexity of an algorithm is expressed using big O notation.

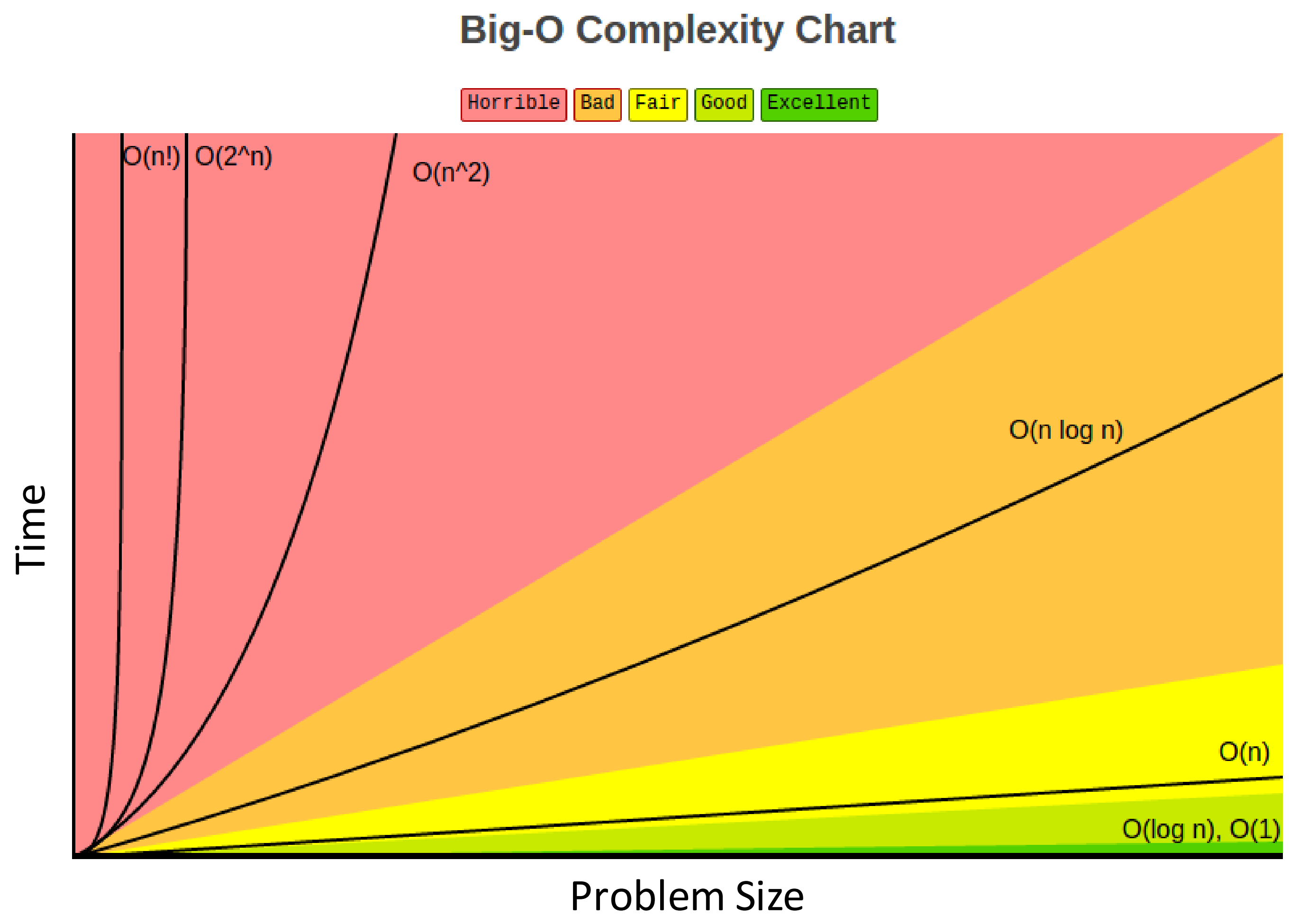
Rate of Growth and Big-O Notation

- The rate of growth shows how the running time of an algorithm increases as a function of the input.
- The execution time is expressed using a function f(n), and the Big-O notation gives the upper bound of f(n)
- The Big-O notation express how the function F(n) behaves as n moves towards ∞
- When we design an algorithm our objective is minimize the rate of growth.
- In general, it is represented as f(n) = O(g(n)), when n is large, the upper-bound of f(n) is g(n).

Rate of Growth and Big-O Notation (cont...)

Time Complexity	Name	Examples
O(1)	Constant	Inserting an element at the beginning of a linked list
O(log n)	Logarithmic	Finding an element in a sorted array
O(n)	Linear	Finding an element in unsorted array
O(n log n)	Linear Logarithmic	Sorting an array using merge sort or heap sort
$O(n^2)$	Quadratic	Sorting an array using bubble sort or selection sort
O(2^n)	Exponential	Calculation of Fibonacci numbers, Towers of Hanoi

Rate of Growth and Big-O Notation (cont…)



Source: http://bigocheatsheet.com/

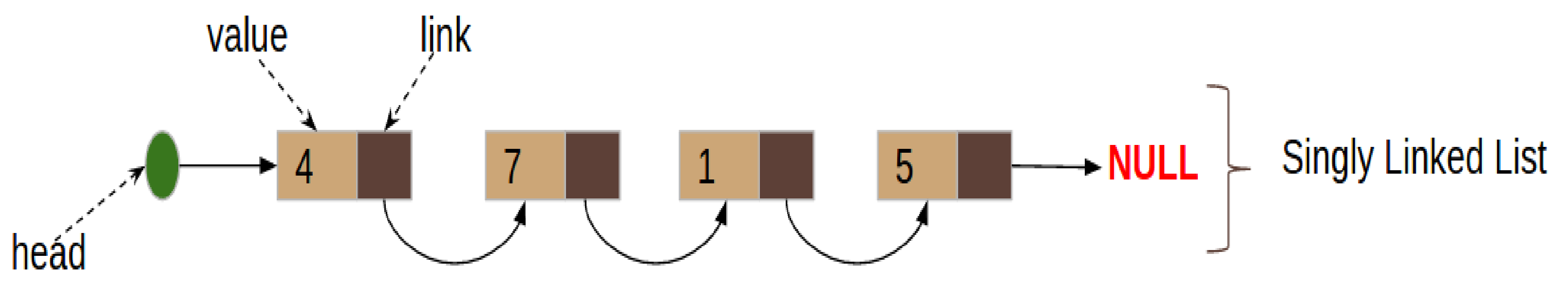
Algorithm Analysis (Best, Worst, and Average)

- Best Case: Describes the algorithm performance under optimal conditions. The algorithm takes the least time to complete.
- Worst Case: Describes the input for which the algorithm takes the longest time to complete.
- Average Case: Provides a predication about the running time of the algorithm. Running the algorithm many times by generating different inputs from some distribution and measure the average time.

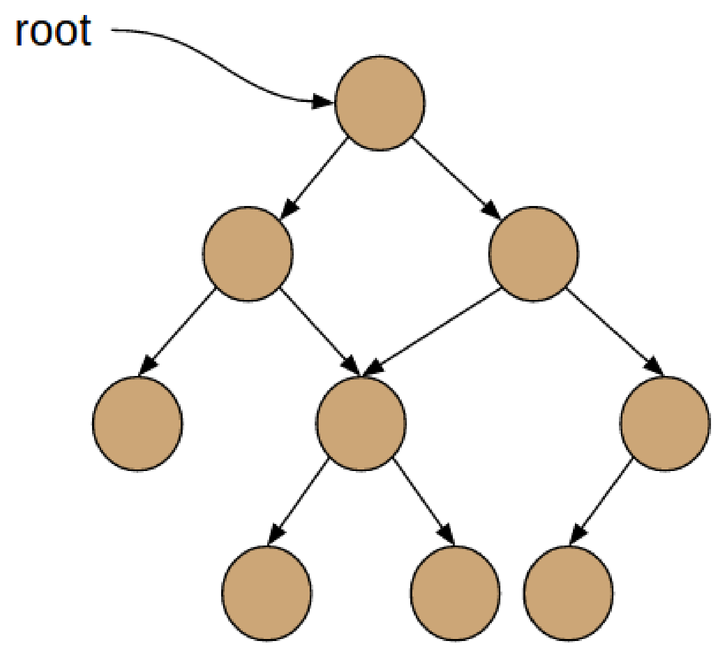
Data Structures

- Data structures is a logical and mathematical model to store and organize data for computational operations so that it can be used efficiently.
- Data structures are categorized into two types based on how we access the data:
  - Linear Data Structures: elements in linear data structures are accessed in a sequential order and linked in some linear order (only one data element can directly be reached). Examples Arrays, Linked List, Stacks and Queues.
  - Nonlinear Data Structures: elements in nonlinear data structures are linked in nonlinear order. Examples are trees and graphs

Linear and Nonlinear Data Structures



Linked List as an example of linear data structures



Tree Data Structure

Tree as an example of nonlinear data structures

Data Structures Operations

1. Traversing: access each data element exactly once so that it can be processed.
2. Inserting: to add a new element to the structure.
3. Deleting: to remove an existing element from the structure
4. Searching: to find an element with a given key value or the location of the element in the structure.
5. Sorting: arranging the elements in the structure in some order.
6. Merging: to combine two or more structures into one structure.

Abstract Data Types (ADTs)

What is abstract data types?

- ADT refers to a set of data values and associated well-defined operations that are independent of any particular implementation.
- We know what a specific data types can do, but we do not care about the implementation details.
- Examples:

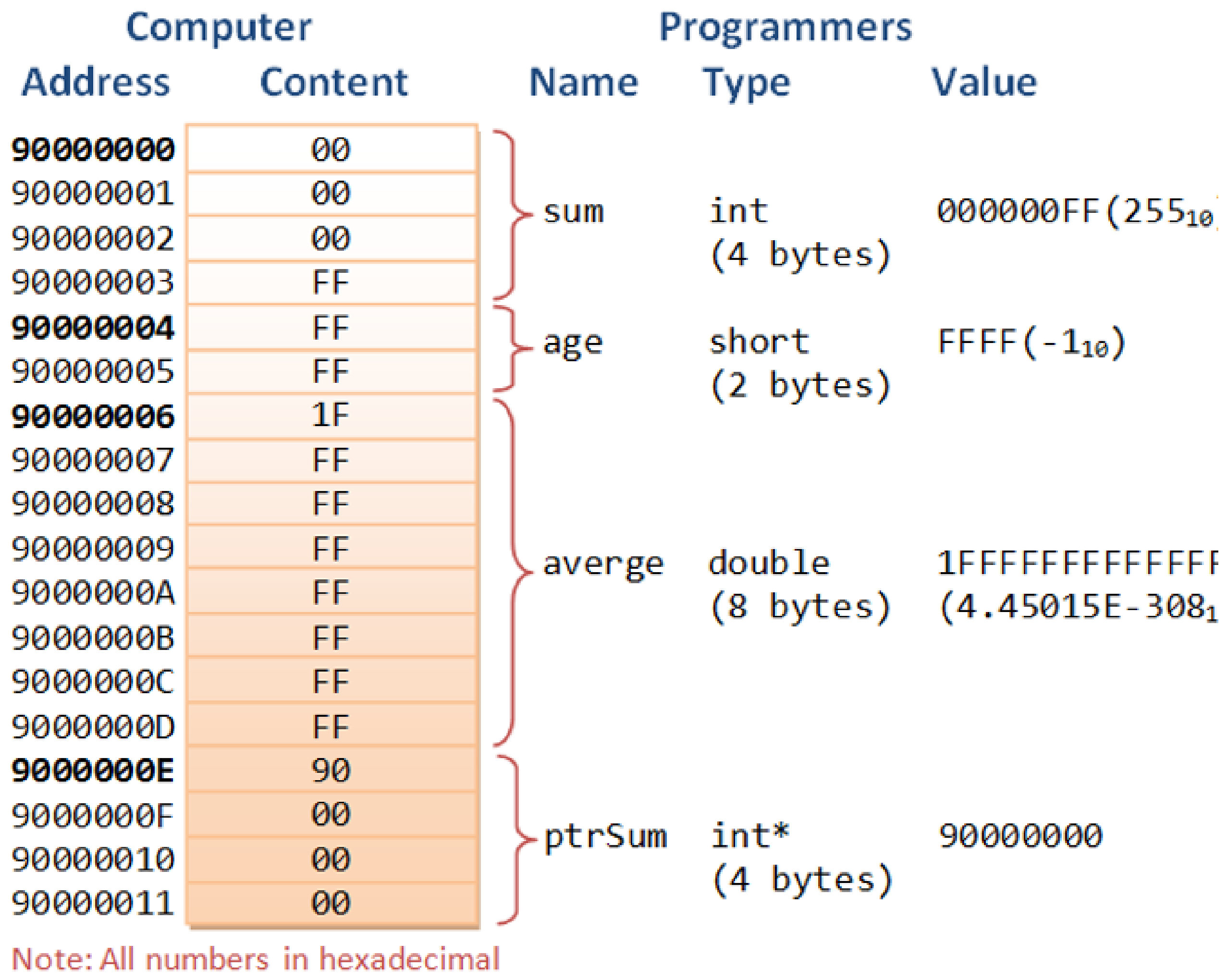
```
short age;
int Numbers[10];
```

- The implementation details of the operations (functions) for a given data types are hidden from the user | programmer.

Abstract Data Types (ADTs)

- How is the array stored in memory?

  int myArray[10];



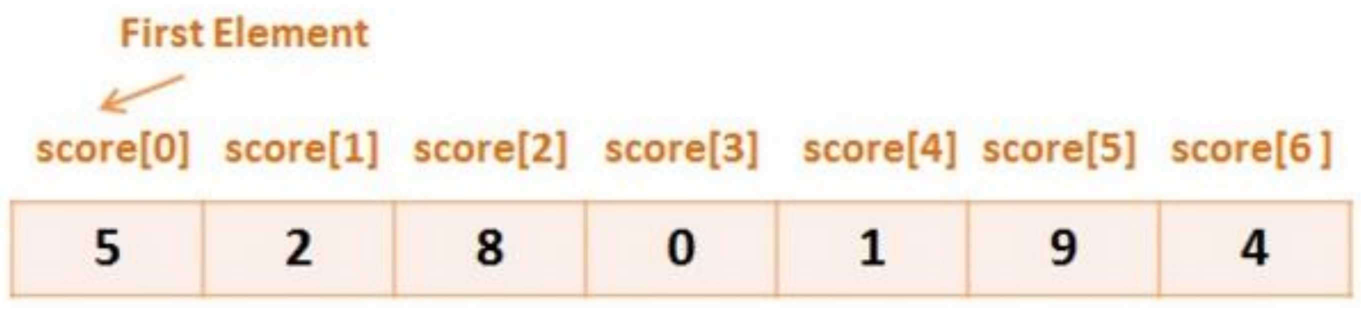
Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF(255$_{10}$
90000001	00			
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1$_{10}$)
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFF (4.45015E-308$_1$
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Arrays: Simple Linear Data Structure

What is an array?

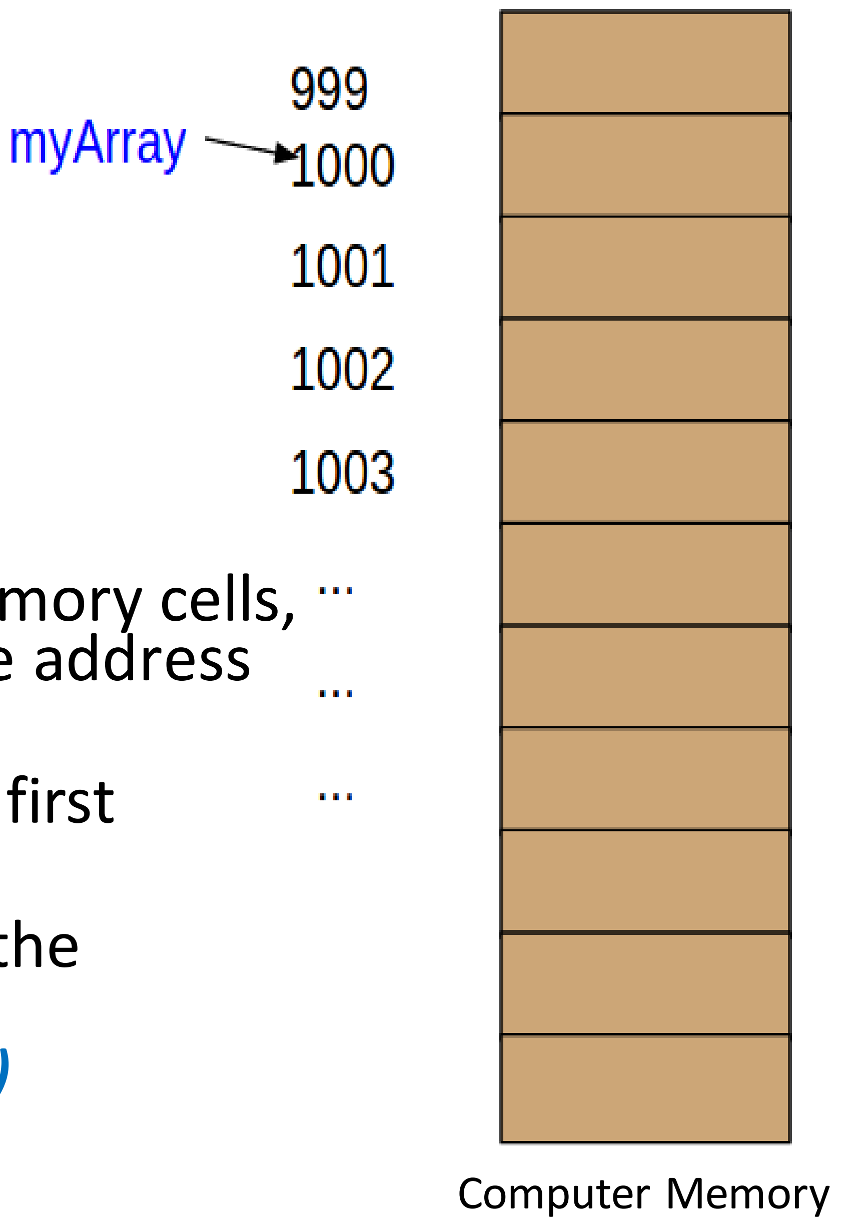
- A number of elements in specific order.
- Allocate contiguous memory words for the elements of arrays.
- Usually all the elements are from the same type.



How is the array stored in memory?

- Declare an array of 10 integers in C/C++
  - myArray[10];
  - myArray[3]=7;
- Because the array is allocated in contiguous memory cells, the computer does not need to keep track of the address of every element.
- It only needs to keep track of the address of the first element in myArray, denoted by ***Base(myArray)***
- To access any element with index K in the array the computer calculate the address as follows

  Address(myArray[k]) = Base(myArray) + K*sizeof(int)

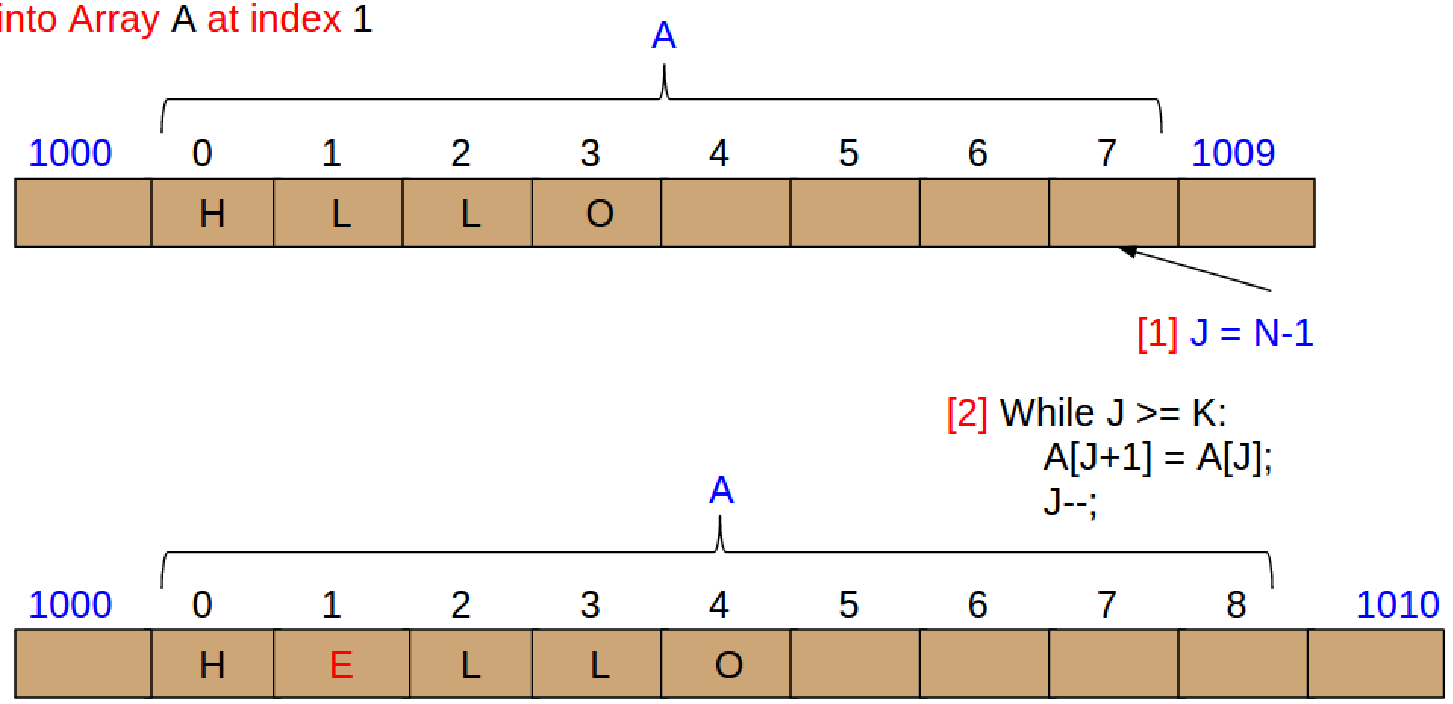


Question 1:

Consider the array AUTO which records the number of automobiles sold each year from 2010 through 2016. The base address of AUTO in memory is 400 what is the memory address of the automobiles sold in 2014?

Array Operations: Inserting

Insert E into Array A at index 1



[1] J = N-1

```
[2] While J >= K:
        A[J+1] = A[J];
        J--;
```

```
[3] A[k] = E
    N = N+1
```

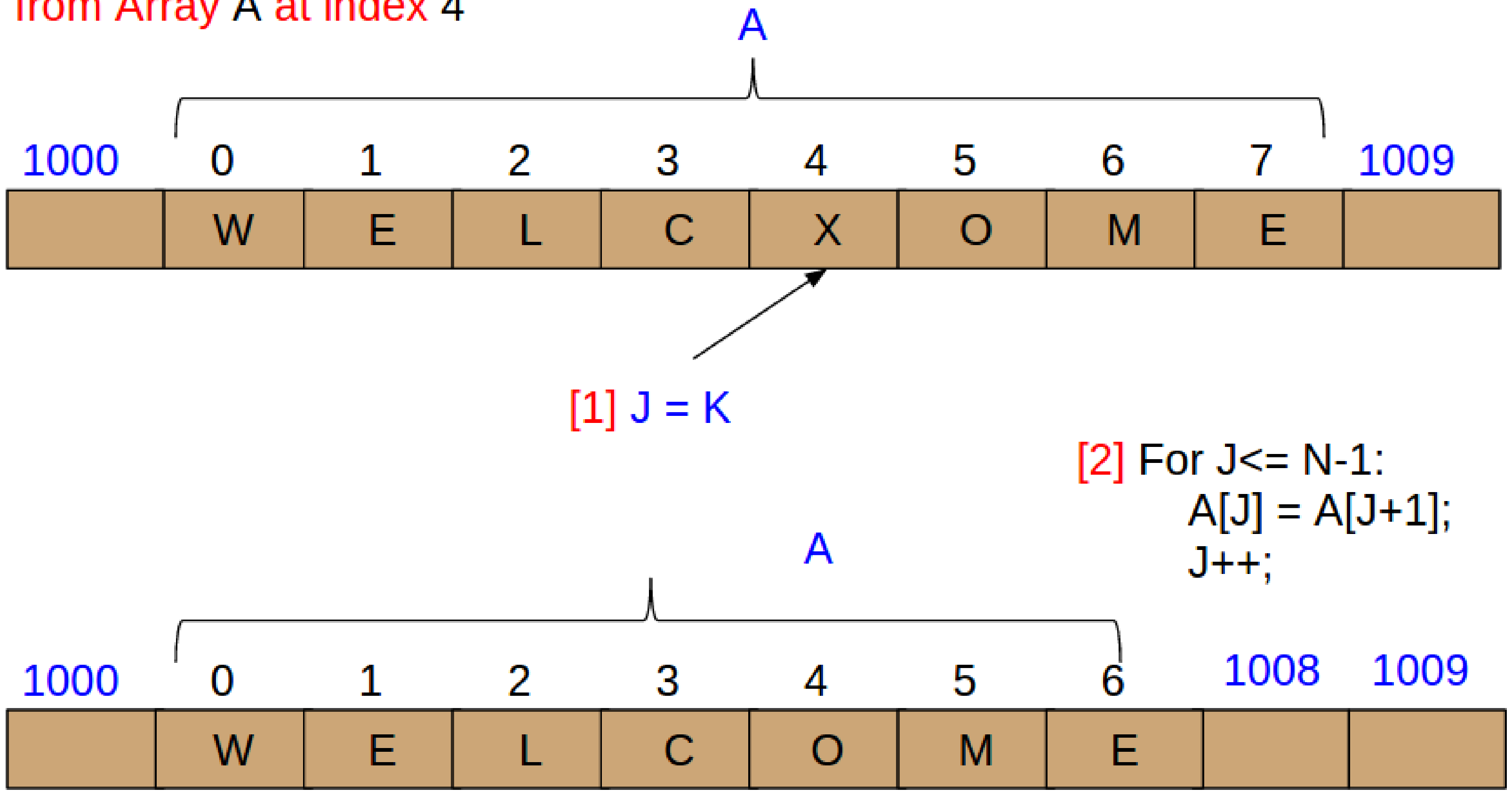
Array Operations: Inserting

A: array, N: array size, K: insertion position, V: value to insert

```
1.  Set J = N // J is a counter
2.  Repeat Steps 3 and 4 while J >= K:
3.      Set A[J+1] = A[J]  // move Jth element downward
4.      Set J = J –1
5.  Set A[K] = V // insert the new element
6.  Set N = N+1 // update the size of the array.
```

Arrays Operations: Deleting

Delete X from Array A at index 4



[1] J = K

```
[2] For J<= N-1:
        A[J] = A[J+1];
        J++;
```

[3] N = N-1

Array Operations: Deleting

A: array, N: array size, K: deletion position

```
1.  Repeat for J = K to N-1:
2.      A[J] = A[J+1] // move the J +1st element upward
3.      Set J = J+1
4.  Set N = N-1    // update the size of the array
```

Array Operations: Searching

A: array, N: array size, ITEM: search key

```
1.  Set J = 0 // J is a counter
2.  F = 0 // flag
3.  Repeat Steps 3 and 4 while J <= N:
4.      If A[J]  equal ITEM then F = 1 break
5.      Set J = J –1
6.  Print F and J
```

Self-Assessments

1. What is the run-time complexity of adding an item to an array?
2. What is the run-time complexity of deleting an item from an array?
3. What is the worst-case and best-case complexity for finding an item in an unsorted array?
4. What is the run-time complexity for adding an item to end of an array?

Self-Assessments

Monkey Sort | BogoSort or Permutation Sort is sorting algorithm. The algorithm successively generates permutations of its input until it finds one that is sorted.

```
1. Let A be an array of N Items:
2. While not Sorted(A) do:
3.        Shuffle(A)
4. End
```

In your opinion what monkey sort is a bad sorting algorithm?

Questions