Estructuras de datos

Profesor James Montealegre Gutiérrez

Ingeniero Telemático

MSc. Ingeniería de Software

Propósitos de la clase

- Control de lectura
- Introducción a las estructuras de datos
- Variables y arreglos
- Ordenamiento de arreglos estáticos
- Actividad

Espacio de memoria

- **¿Por qué es importante la memoria en los programas?**

Todo programa necesita almacenar y acceder a datos en memoria.

Ejemplos de datos en programas:

- Texto en un documento.
- Información en una página web.
- Registro de variedades de café en una base de datos.

También se almacena información interna del programa:

- Número de iteraciones en un bucle.
- Ubicación del personaje en un juego.
- Hora actual del sistema.

Variables

¿Qué es una variable?

Un nombre que representa la dirección de un dato en memoria.

Se utilizan para almacenar información que cambia durante la ejecución.

Ejemplos:

- Contar iteraciones de un bucle.
- Guardar el puntaje de un jugador.
- Registrar errores ortográficos en un editor de texto

Variables

Variables y Organización en Memoria

La memoria del computador se puede visualizar como una columna de contenedores y cada variable ocupa uno o más contenedores contiguos/adyancentes.

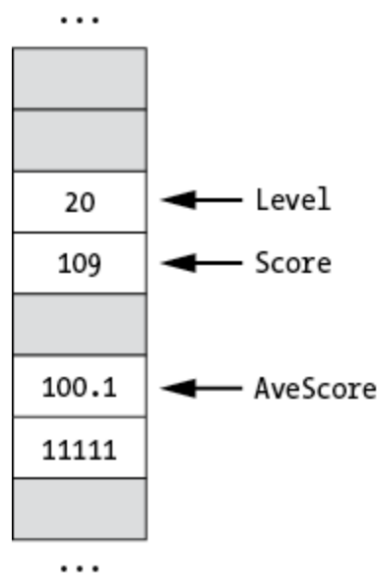
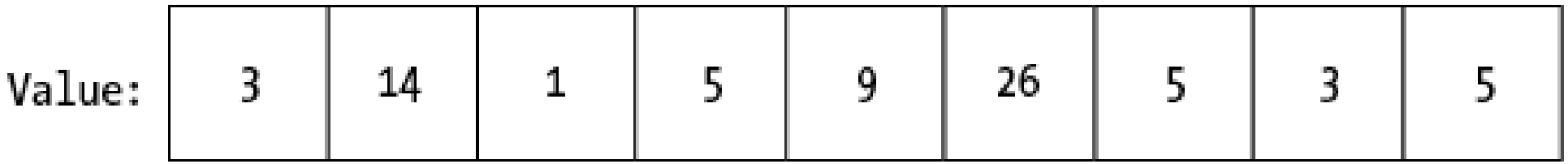


Figure 1-1: Computer memory depicted as a column of bins

Arreglos

- Son una estructura de datos (unidad de almacenamiento) usado generalmente para almacenar de manera eficiente múltiples datos con valores relacionados.
- Los arrays almacenan información de manera adyacente e indexada



Arreglos

- Son una manera eficiente de almacenar múltiples variables en memoria

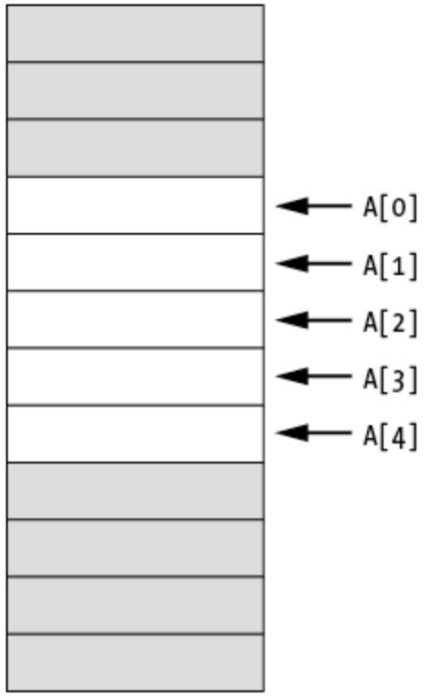


Figure 1-5: A zero-indexed array arranged in computer memory

Insertion Sort

Actividad...

Consideraciones:

ordenar el arreglo
array = [5, 1, 8, 9, 12]

Suponer el primer elemento ya esta ordenado

Considerar un conjunto de datos ordenado

Ir moviendo a la derecha los elementos que son mayores al elemento inicial de comparación

Actualizar las posiciones

Insertion Sort

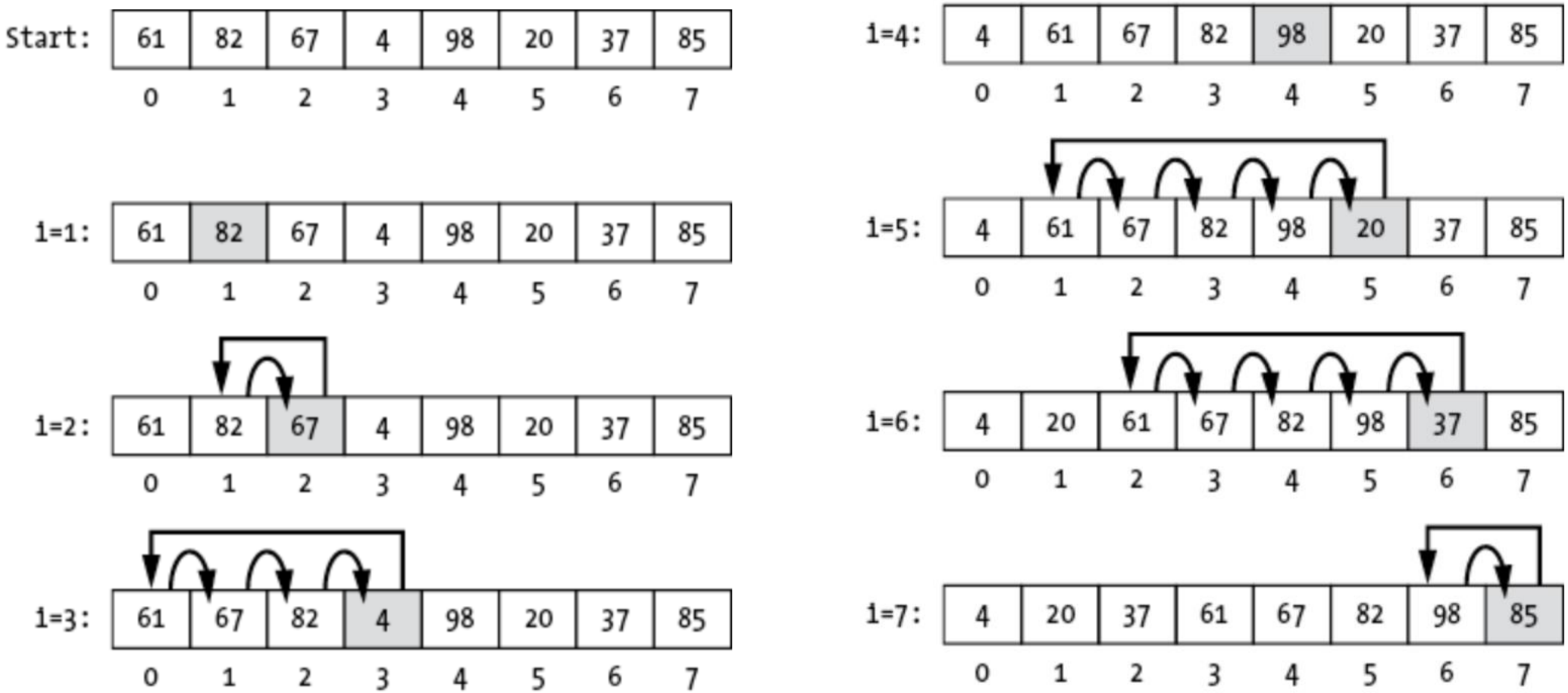


Figure 1-7: Visualization of an insertion sort algorithm

Insertion Sort

```
InsertionSort(array: A):
    Integer: N = length(A)
    Integer: i = 1
 ❶ WHILE i < N:
        Type: current = A[i]
        Integer: j = i - 1
     ❷ WHILE j >= 0 AND A[j] > current:
            A[j + 1] = A[j]
            j = j - 1
        A[j + 1] = current
        i = i + 1
```

Insertion Sort

InsertionSort(array: A): // Recibe como entrada un arreglo A

Integer: N = length(A) // Almacena la longitud del array A

Integer: i = 1 // Suposición primer element ordenado

WHILE i < N: // Se ejecuta un bucle mientras i sea menor que N, es decir, mientras queden elementos por procesar en el array.

current = A[i] // current es el valor que queremos insertar en la parte ordenada del array

j = i – 1 // Comienza en el índice inmediatamente anterior a i.

WHILE j >= 0 AND A[j] > current: // Desplaza los elementos de la parte ordenada que son mayores que current, creando espacio para insertar current en su posición correcta.

Insertion Sort

A[j + 1] = A[j] // mueve A[j] una posición a la derecha, sobrescribiendo A[j + 1]

j = j – 1 // Se reduce el valor de j en 1 para continuar comparando current con los elementos anteriores en la parte ordenada del array.

A[j + 1] = current // Después de desplazar los elementos mayores, current se inserta en su posición correcta dentro de la parte ordenada.

i = i + 1 // Se incrementa i para pasar al siguiente elemento del array y repetir el proceso hasta que todos los elementos estén ordenados.