Arrays

Linear Arrays

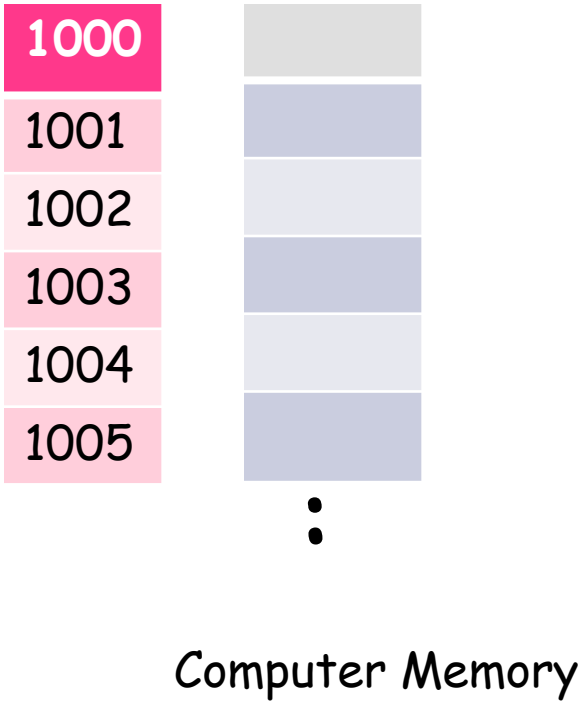
- A linear array is a list of a finite number of n homogeneous data elements (that is data elements of the same type) such that
  - The elements of the arrays are referenced respectively by an index set consisting of n consecutive numbers
  - The elements of the arrays are stored respectively in successive memory locations in Linear Arrays

Linear Arrays

- The number n of elements is called the length or size of the array.
- The index set consists of the integer 1, 2, … n
- Length or the number of data elements of the array can be obtained from the index set by

Length = UB – LB + 1 where UB is the largest index called the upper bound and LB is the smallest index called the lower bound of the arrays

Representation of Linear Array in Memory



Computer Memory

Representation of Linear Array in Memory

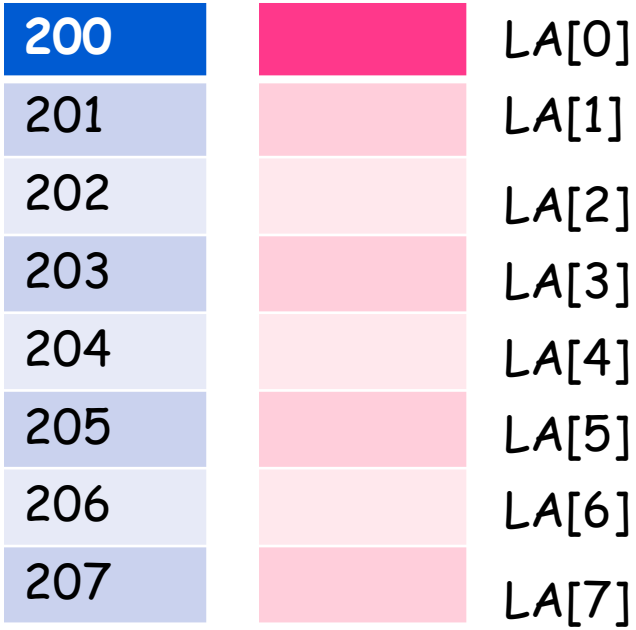
- Let **LA** be a linear array in the memory of the computer
- **LOC(LA[K]) = address of the element LA[K] of the array LA**
- The element of **LA** are stored in the successive memory cells
- Computer does not need to keep track of the address of every element of **LA,** but need to track only the address of the first element of the array denoted by **Base(LA)** called the **base address** of LA

Representation of Linear Array in Memory

- **LOC(LA[K]) = Base(LA) + w(K – LB)**
  where **w** is the number of words per memory cell of the array LA [**w** is the size of the **data type**]

Example 1

Find the address for LA[6]
Each element of the array occupy 1 byte

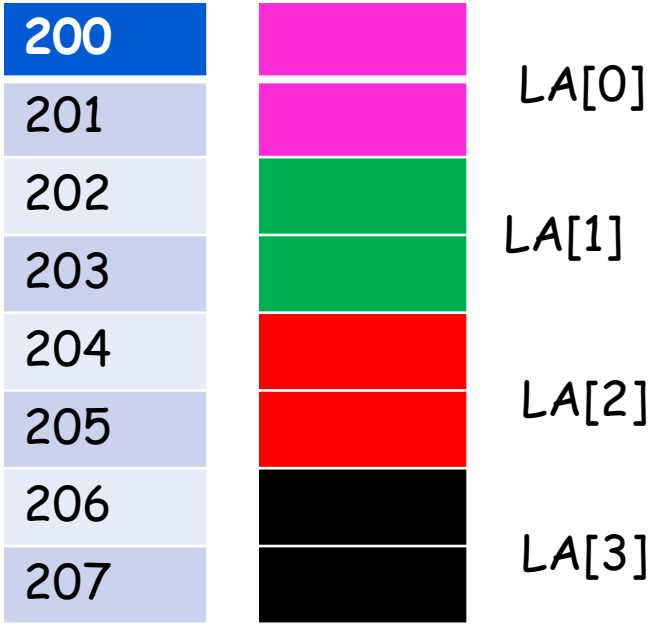


LOC(LA[K]) = Base(LA) + w(K – lower bound)

LOC(LA[6]) = 200 + 1(6 – 0) = 206

Example 2

Find the address for LA[15]
Each element of the array occupy 2 bytes



LOC(LA[K]) = Base(LA) + w(K – lower bound)

LOC(LA[15]) = 200 + 2(15 – 0) = 230

Operation on Linear Structure

- **Traversal** : Processing each element in the list
- **Search** : Finding the location of the element with a given value or the record with a given key
- **Insertion**: Adding a new element to the list
- **Deletion**: Removing an element from the list
- **Sorting** : Arranging the elements in some type of order
- **Merging** : Combining two list into a single list

Traversing Linear Arrays

- Traversing is accessing and processing each element of the data structure exactly once.

Algorithm for traversing an array

(Traversing a Linear Array) Here LA is a linear array with lower bound LB and upper bound UB. This algorithm traverses LA applying an operation PROCESS to each element of LA.

1. **[Initialize counter]** Set K := LB.
2. Repeat steps 3 and 4 while K <= UB.
3. **[Visit element]** Apply PROCESS to LA[K].
4. **[Increment counter]** Set K := K + 1.

   [End of Step 2 loop]
5. Exit.

Inserting and Deleting

- **Insertion**: Adding an element
  - Beginning
  - Middle
  - End

- **Deletion**: Removing an element
  - Beginning
  - Middle
  - End

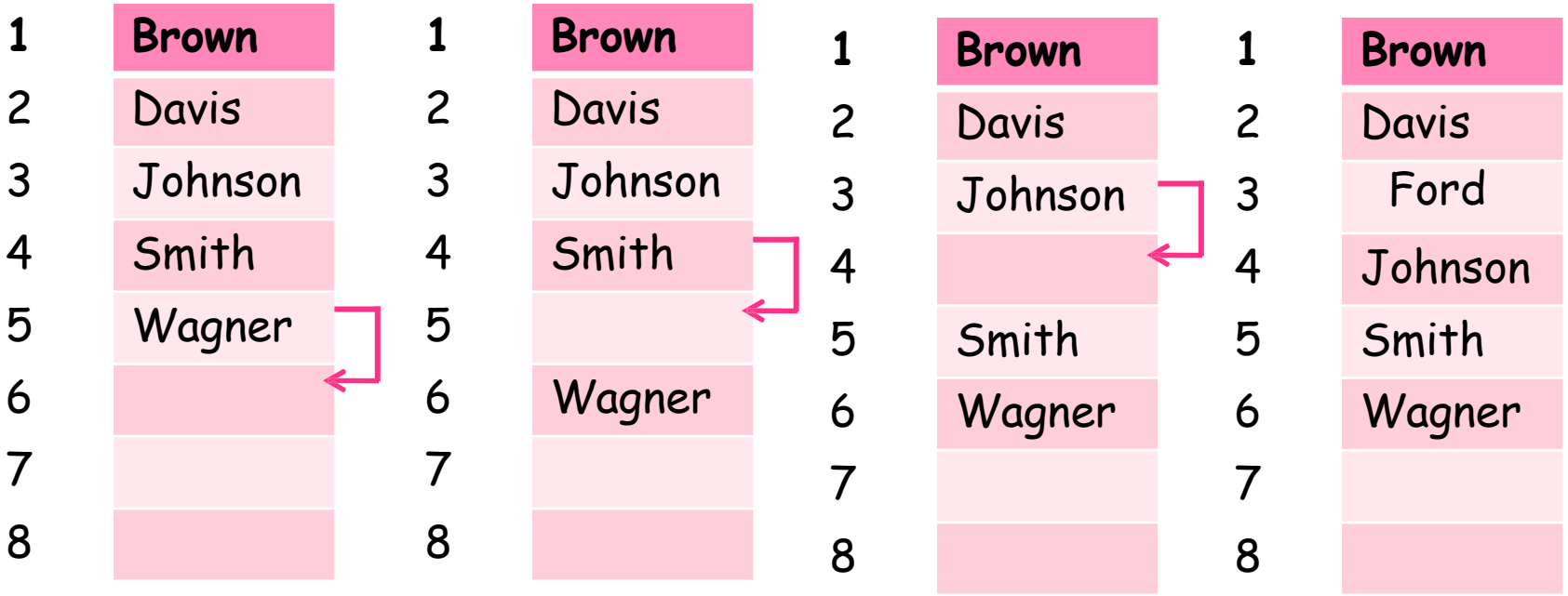
Insertion

1	**Brown**
2	Davis
3	Johnson
4	Smith
5	Wagner
6	
7	
8	

1	**Brown**
2	Davis
3	Johnson
4	Smith
5	Wagner
6	Ford
7	
8	

Insert Ford at the End of Array

Insertion



Insert Ford as the 3rd Element of Array

Insertion is not Possible without loss of data if the array is FULL

Algorithm for inserting an element into an array

(Inserting into a linear array) INSERT (LA, N, K, ITEM)

Here LA is a linear array with N elements and K is a positive integer such that K <= N. This algorithm inserts an element ITEM into the Kth position in LA.

1. **[Initialize counter.]** Set J := N.
2. Repeat Steps 3 and 4 while J >= K.
3. **[Move Jth element downward.]** Set LA[J+1] := LA[J].
4. **[Decrease counter.]** Set J := J – 1.

   [End of Step 2 loop.]
5. **[Insert element.]** Set LA[K] := ITEM.
6. **[Reset N.]** Set N := N + 1.
7. Exit.

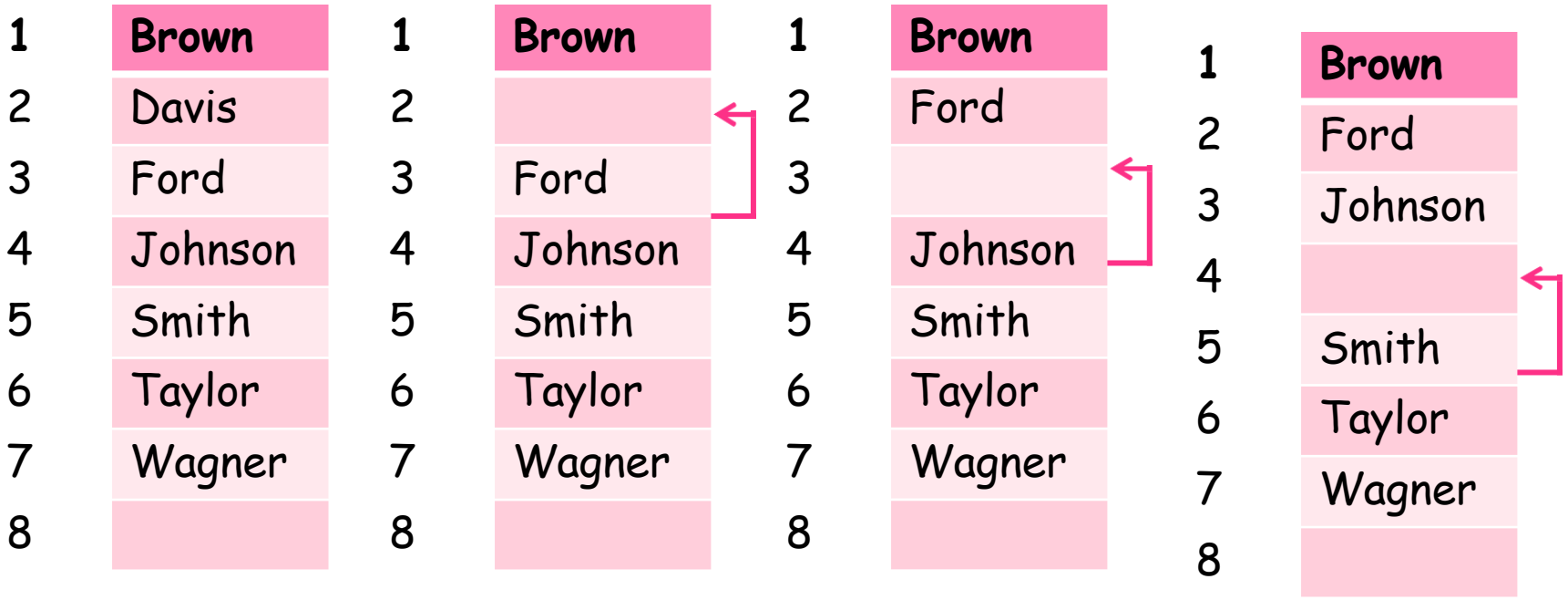
Deletion

1	**Brown**
2	Davis
3	Ford
4	Johnson
5	Smith
6	Taylor
7	Wagner
8	

1	**Brown**
2	Davis
3	Ford
4	Johnson
5	Smith
6	Taylor
7	
8	

Deletion of Wagner at the End of Array

Deletion



Deletion of Davis from the Array

Algorithm for deleting an element from an array

(Deleting from a linear array) DELETE (LA, N, K, ITEM)

Here LA is a linear array with N elements and K is a positive integer such that K <= N. This algorithm deletes the Kth element from LA.

1. Set ITEM := LA[K].
2. Repeat for J = K to N – 1

   [Move J + 1st element upward.] Set LA[J] := LA[J + 1].

   [End of loop.]
3. **[Reset the number N of elements in LA.]** Set N := N - 1.
4. Exit.

Merging Algorithm

- Suppose A is a sorted list with r elements and B is a sorted list with s elements. The operation that combines the element of A and B into a single sorted list C with n=r + s elements is called merging.
- Complexity of merging alg.
- Time:- O(n^2)
- Space:- O(1)

Merging Algorithm

- Algorithm: Merging (A, R,B,S,C)

  Here A and B be sorted arrays with R and S elements respectively. This algorithm merges A and B into an array C with N=R+ S elements.

- Step 1: Set NA=1, NB=1 and NC=1
- Step 2: Repeat while NA <= R or NB <= S:

  if A[NA] ≤ B[NB], then:

  Set C[NC] = A[NA]

  Set NA = NA +1

  else

  Set C[NC] = B[NB]

  Set NB = NB +1

  [End of if structure]

  Set NC= NC +1

[End of Loop]

Merging Algorithm

- Step 3: If NA <= R, then:

  Repeat while NB <= S:

  Set C[NC] = B[NB]

  Set NB = NB+1

  Set NC = NC +1

  [End of Loop]

  else

  Repeat while NA <= R:

  Set C[NC] = A[NA]

  Set NC = NC + 1

  Set NA = NA +1

  [End of loop]

  [End of if structure]

- Step 4: Return C[NC]

Important points

- Arrays can be declared as: AUTO(1932: 1984); where AUTO[K] = value at K.
- Length = UB – LB + 1
- Address of any element of LA is calculated as:

  LOC(LA[K]) = Base(LA) + w(K – LB)
- Ex: If Base address = 200; w = 4 words, then find address of K=1965 (Ans: 332)

Address Calculation in single (one) Dimension Array

- Address of A [I] = B + W * (I – LB)
- Where,
  B = Base address
  W = Storage Size of one element stored in the array (in byte)
  I = Subscript of element whose address is to be found
  LB = Lower limit / Lower Bound of subscript, if not specified assume 0 (zero)

Example:
Given the base address of an array **B[1300…..1900]** as 1020 and size of each element is 2 bytes in the memory. Find the address of **B[1700]**.
Solution:
The given values are: B = 1020, LB = 1300, W = 2, I = 1700
Address of A [I] = B + W * (I – LB)
= 1020 + 2 * (1700 – 1300)
= 1020 + 2 * 400
= 1020 + 800
= 1820 **[Ans]** **Example:**