C How to Program

Arrays

Chapter 6 - Arrays

Outline

6.1 Introduction

- Arrays
  - Structures of related data items
    - An array is a set of data values – not just one
  - Static entity – same size throughout program
  - Dynamic data structures discussed in Chapter 12

6.2 Arrays

- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

  arrayname **[** *position number* **]**
  - First element at position **0**
  - **n** element array named **c**:
    - **c[0], c[1]...c[n – 1]**

Name of array (Note that all elements of this array have the same name, **c**)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array **c**

Arrays

Think of an array as a row of boxes

Each box is an *element*

The elements have an *index* – a label for each element of the array that have 10 integers

7	13	42	4	17	22	7	66	3	9
0	1	2	3	4	5	6	7	8	9

The 10 elements

index of each element

6.2 Arrays

- Array elements are like normal variables

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
```

  – Perform operations on subscript. If **x** equals **3**

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

6.3 Declaring Arrays

- When declaring arrays, specify
  - Name
  - Type of array
  - Number of elements

```
arrayType arrayName[ numberOfElements ];
```

  - Examples:

```
int c[ 10 ];
float myArray[ 3284 ];
```

- Declaring multiple arrays of same type
  - Format similar to regular variables
  - Example:

```
int b[ 100 ], x[ 27 ];
```

6.4 Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If not enough initializers, rightmost elements become **0**

```
int n[ 5 ] = { 0 }
```

    - All elements 0
  - If too many a syntax error is produced syntax error
  - C arrays have no bounds checking
- If size omitted, initializers determine it

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore 5 element array

```c
/* Fig. 6.8: fig06_08.c
   Histogram printing program */
#include <stdio.h>
#define SIZE 10

int main()
{
   int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
   int i, j;

   printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

   for ( i = 0; i <= SIZE - 1; i++ ) {
      printf( "%7d%13d        ", i, n[ i ]) ;

      for ( j = 1; j <= n[ i ]; j++ )   /* print one bar */
         printf( "%c", '*' );

      printf( "\n" );
   }

   return 0;
}
```

1. Initialize array

2. Loop

3. Print

Program Output

```
Element        Value        Histogram
      0           19        *******************
      1            3        ***
      2           15        ***************
      3            7        *******
      4           11        ***********
      5            9        *********
      6           13        *************
      7            5        *****
      8           17        *****************
      9            1        *
```

6.4 Examples Using Arrays

- Character arrays
  - String "**`first`**" is really a static array of characters
  - Character arrays can be initialized using string literals

```
char string1[] = "first";
```

    - Null character `'\0'` terminates strings
    - **`string1`** actually has 6 elements
      - It is equivalent to

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

  - Can access individual characters

```
string1[ 3 ] is character 's'
```

  - Array name is address of array, so "&" is not needed for scanf

```
scanf( "%s", string2 );
```

    - Reads characters until whitespace encountered
    - Can write beyond end of array, be careful

```c
/* Fig. 6.10: fig06_10.c
   Treating character arrays as strings */
#include <stdio.h>

int main()
{
   char string1[ 20 ], string2[] = "string literal";
   int i;

   printf(" Enter a string: ");
   scanf( "%s", string1 );
   printf( "string1 is: %s\nstring2: is %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

   for ( i = 0; string1[ i ] != '\0'; i++ )
      printf( "%c ", string1[ i ] );

   printf( "\n" );
   return 0;
}
```

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

1. Initialize strings

2. Print strings

2.1 Define loop

2.2 Print characters individually

2.3 Input string

3. Print string

Program Output

6.5 Sorting Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat
- Example:
  - original: 3 4 2 6 7
  - pass 1: 3 2 4 6 7
  - pass 2: 2 3 4 6 7
  - Small elements "bubble" to the top

Swapping (Temporary Variables)

It works for all data types

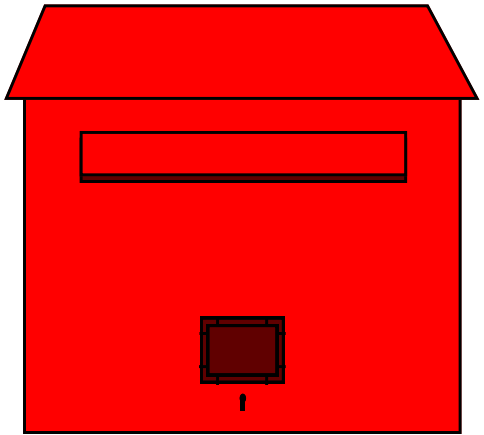
```
int Value1 = 3;
int Value2 = 7;

int Hold = Value1;
```

Value1	**3**
Value2	**7**
Hold	**3**

```
Value1 = Value2;
```

Value1	**7**
Value2	**7**
Hold	**3**

```
Value2 = Hold;
```



Value1	**7**
Value2	**3**
Hold	**3**

Swapping (Mathematic way)

It works for numbers only

```
int Value1 = 3;
int Value2 = 7;
```

Value1	**3**
Value2	**7**

```
Value1 = Value2 + Value1;
```

Value1	**10**
Value2	**7**

```
Value2 = Value1 – Value2;
```

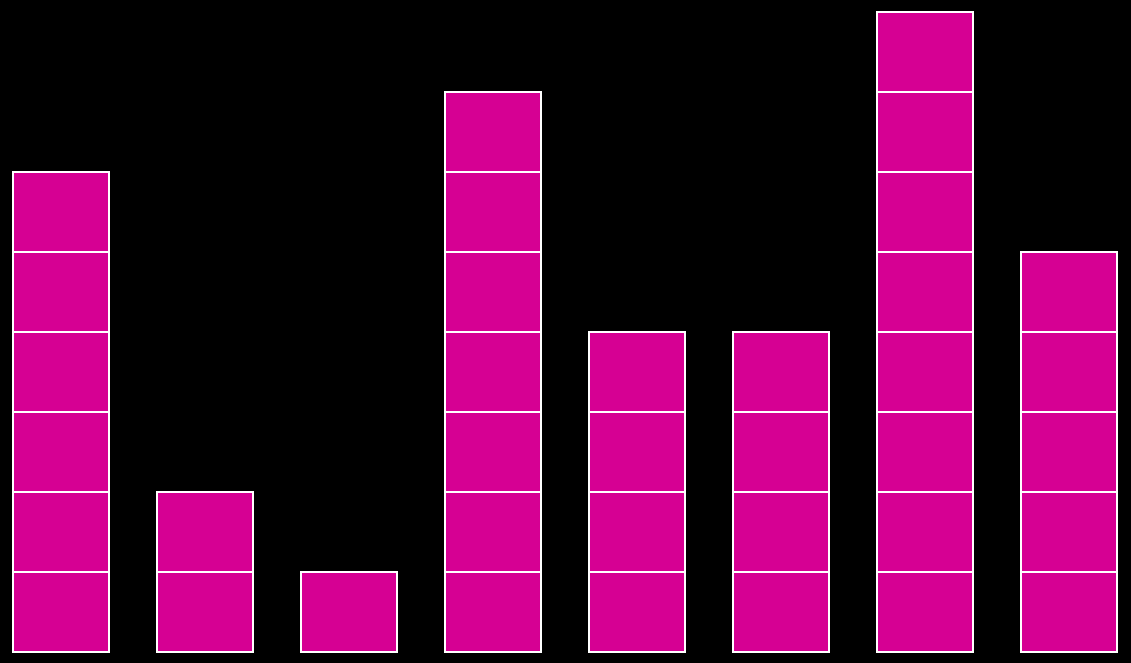
Value1	**10**
Value2	**3**

```
Value1 = Value1 - Value2;
```

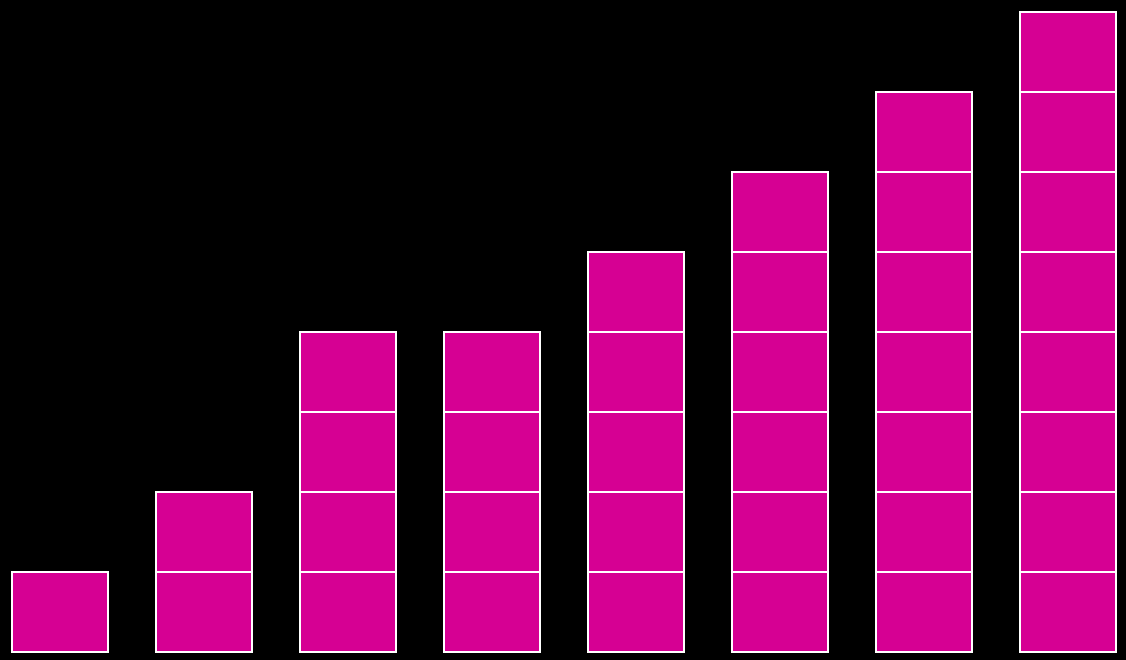
Value1	**7**
Value2	**3**

Another Example: Sorting

Sort the following objects according to their heights



Expected Result



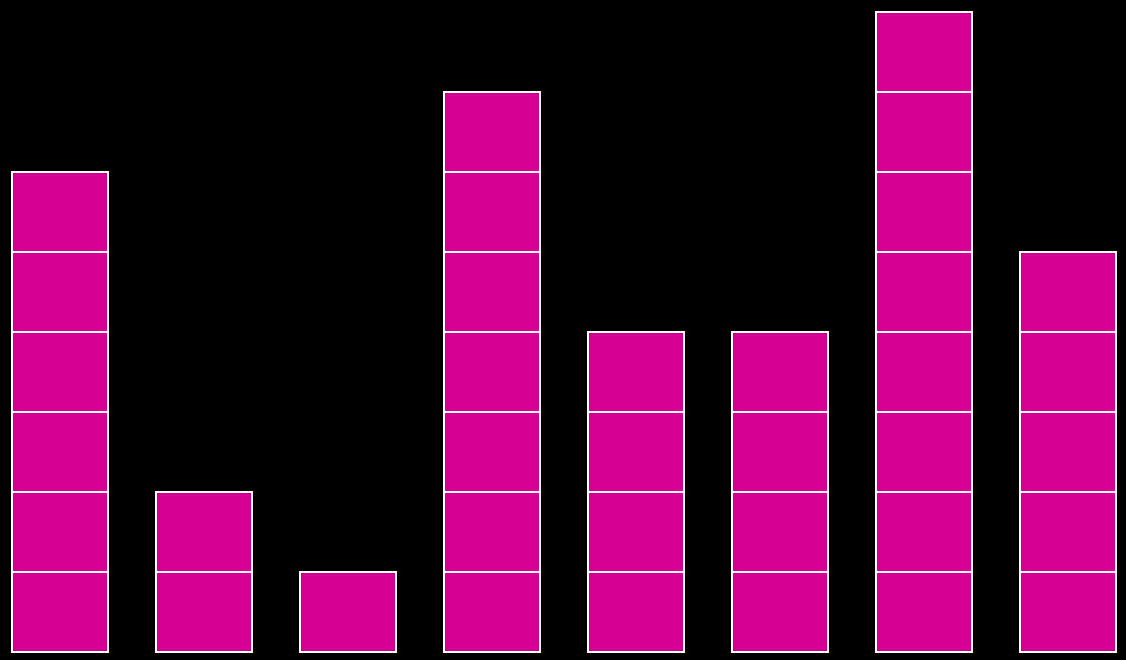
Strategy

There are many strategies for solving this problem. We demonstrate a simple one:

Repeat the following steps while the array is un-sorted:

- Start with the first object in the array
- Swap it with the one next to it if they are in the wrong order
- Repeat the same with the next to the first object
- Keep on repeating until you reach the last object in the list

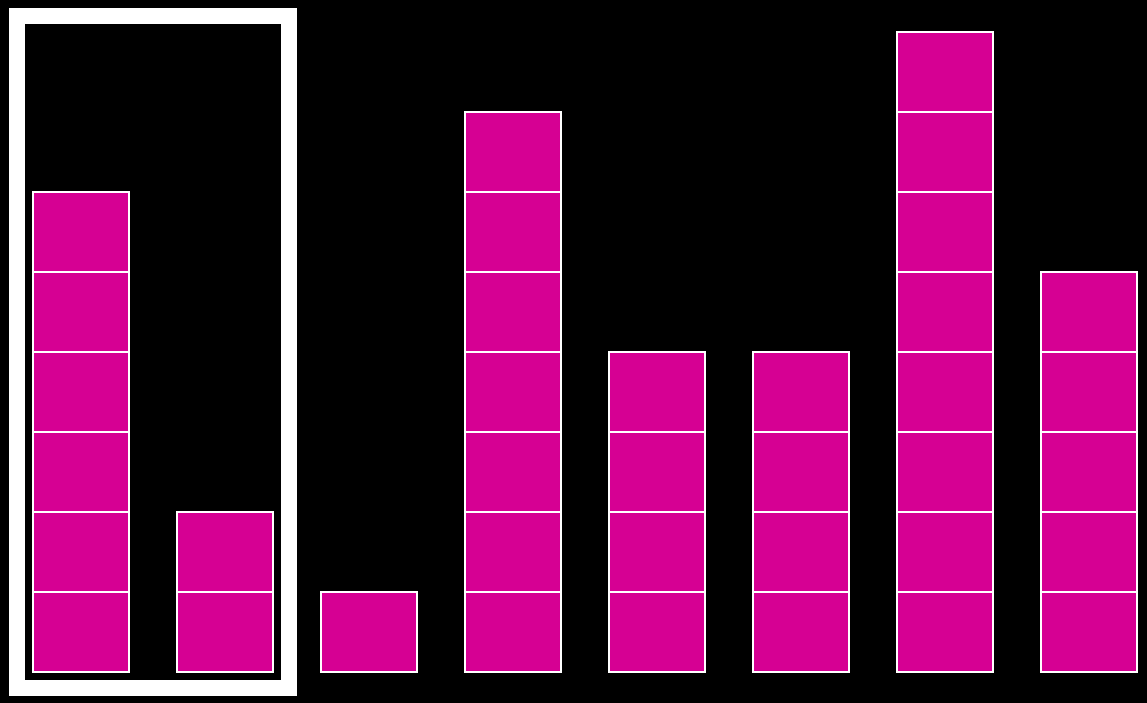
Back to the Objects to be Sorted



Q: Is the array sorted?

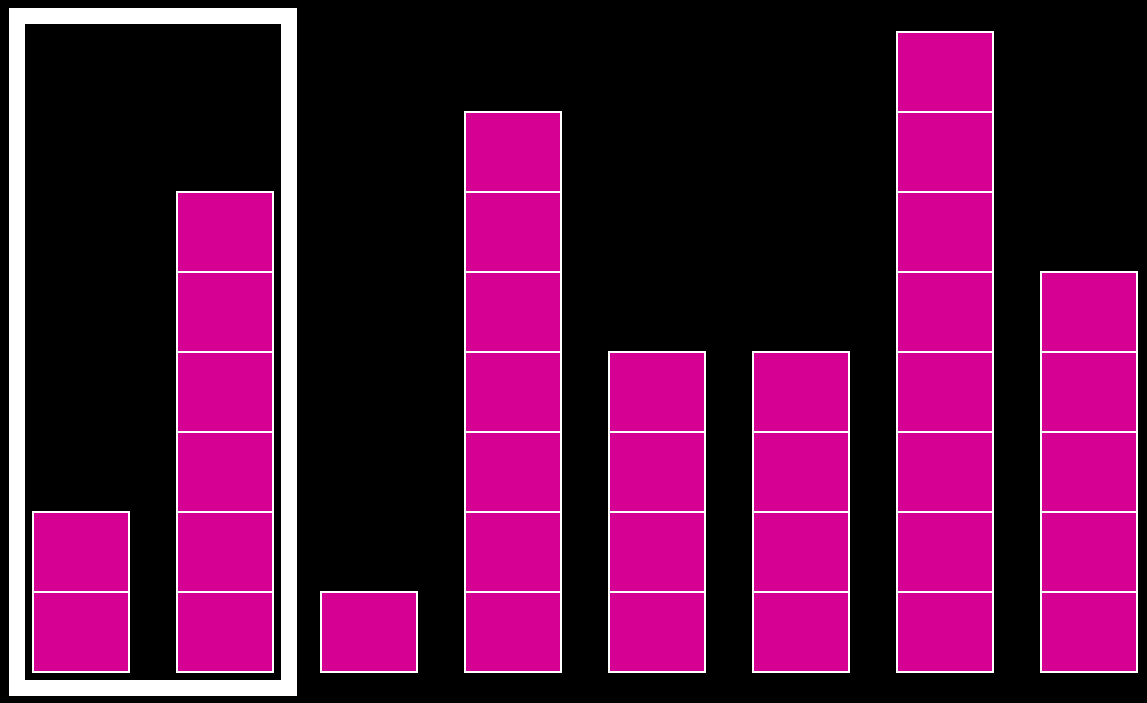
A: No

Sorting: Step A1

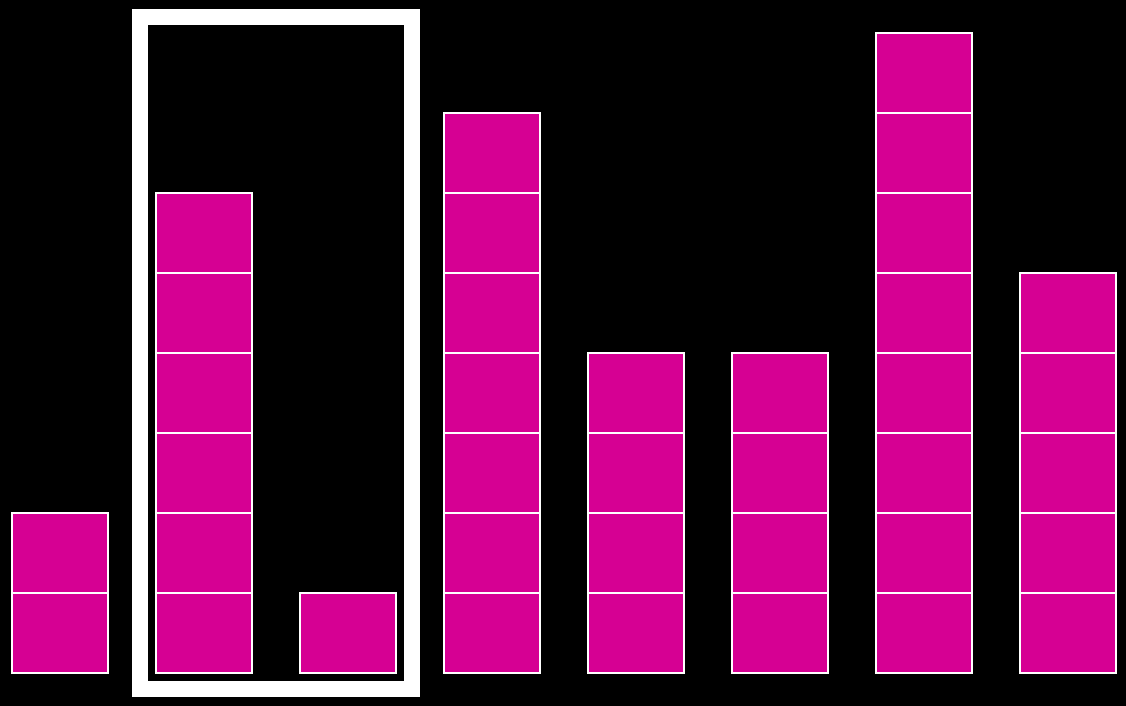


Sorting: Step A1

Swap? Yes

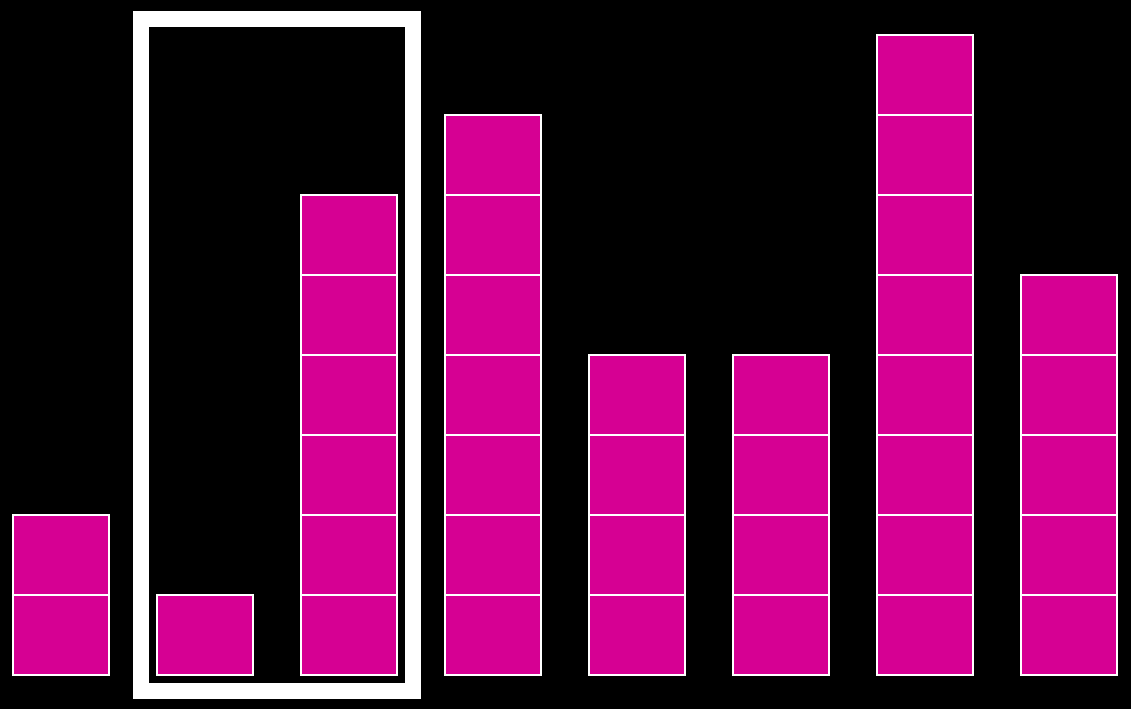


Sorting: Step A2

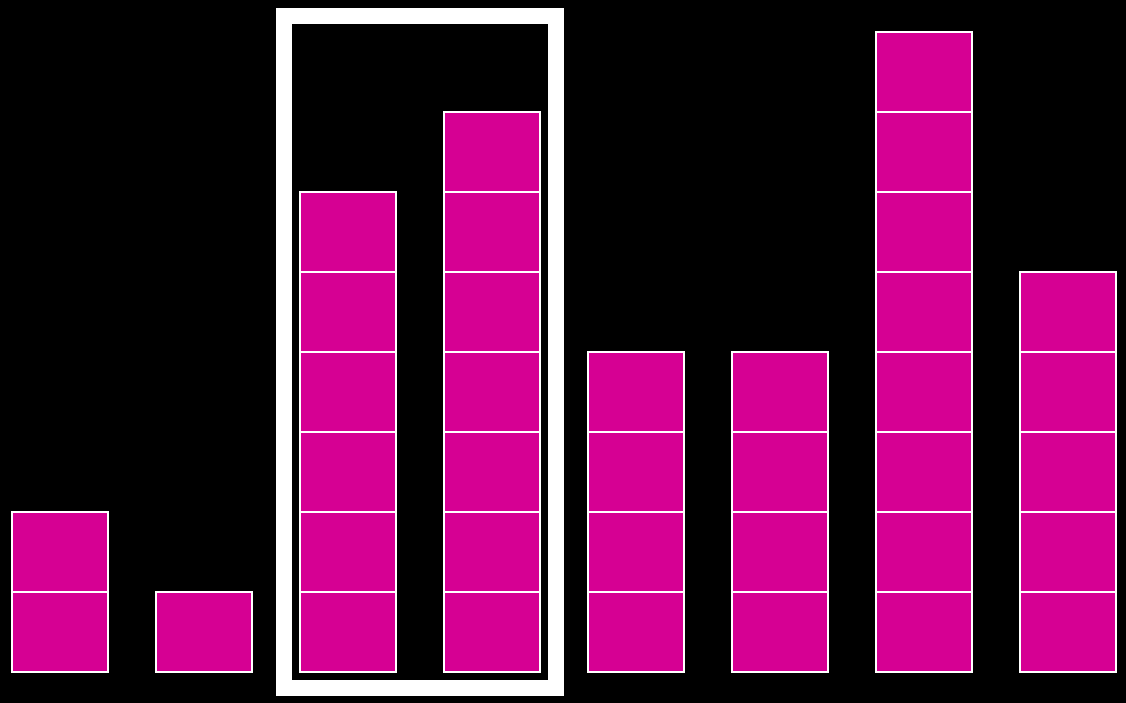


Sorting: Step A2

Swap? Yes

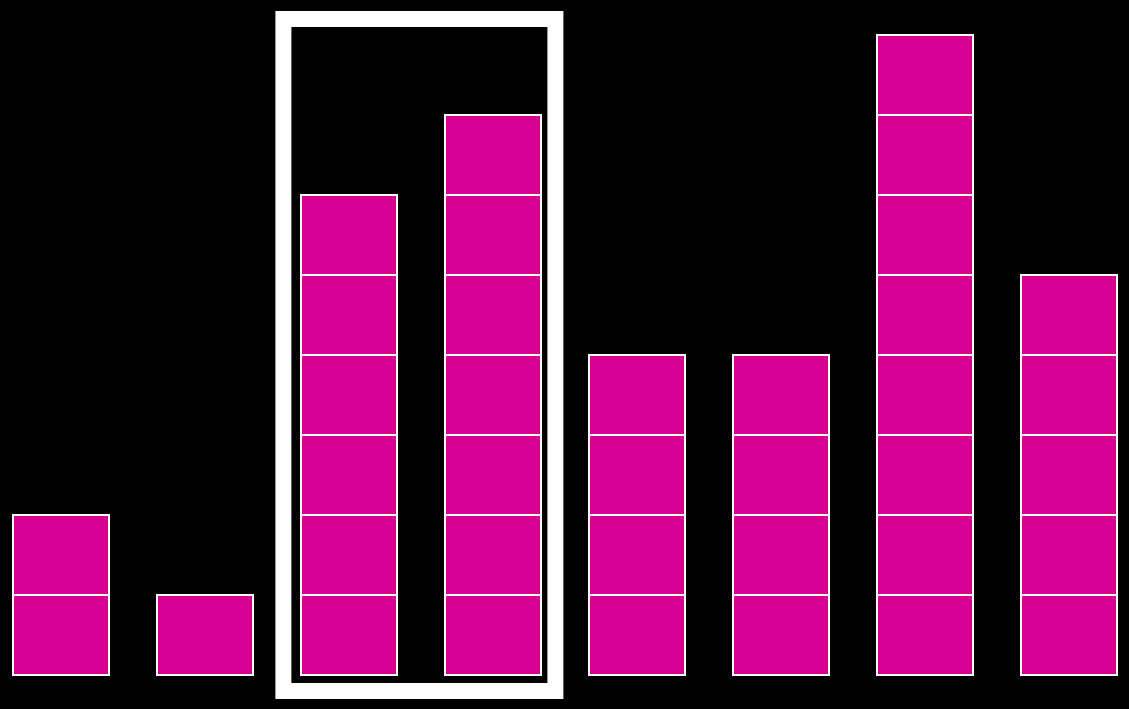


Sorting: Step A3

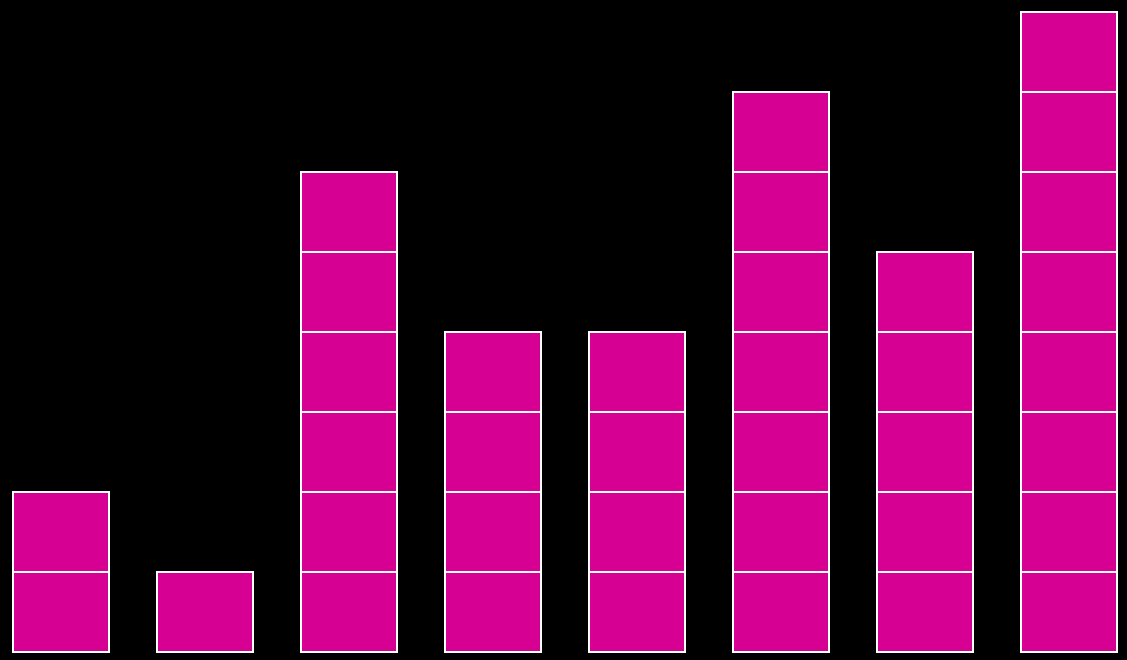


Sorting: Step A3

Swap? No



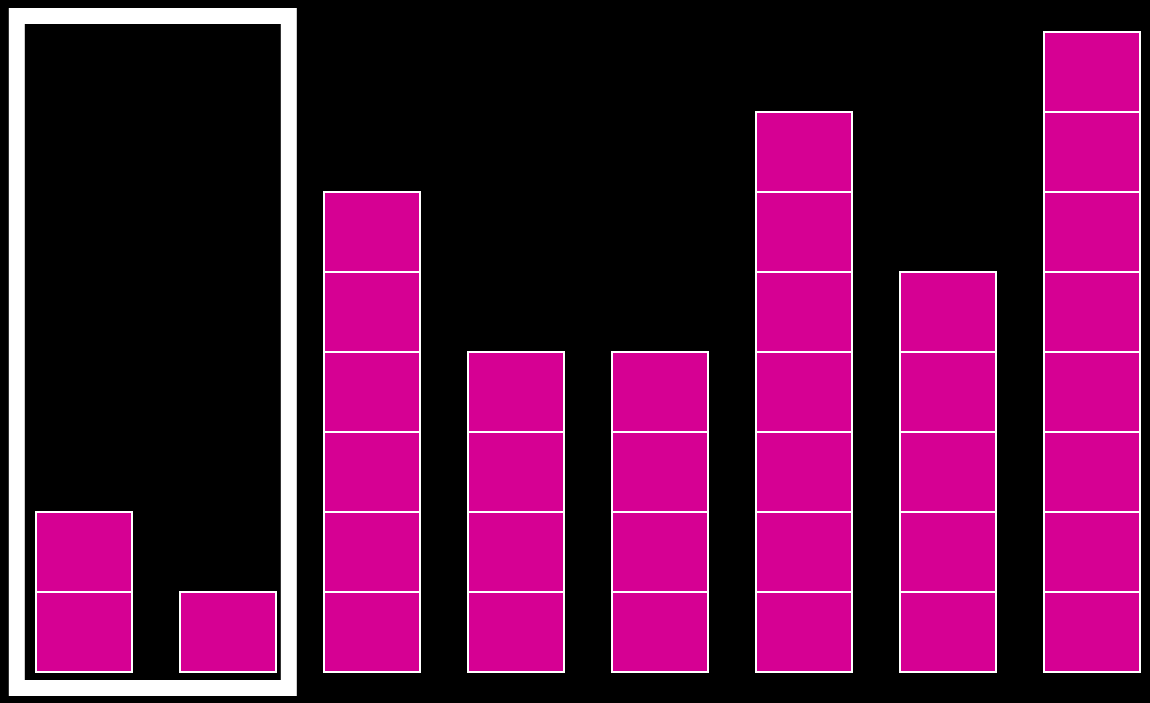
Sorting: After Step A7



Q: Is the array sorted?

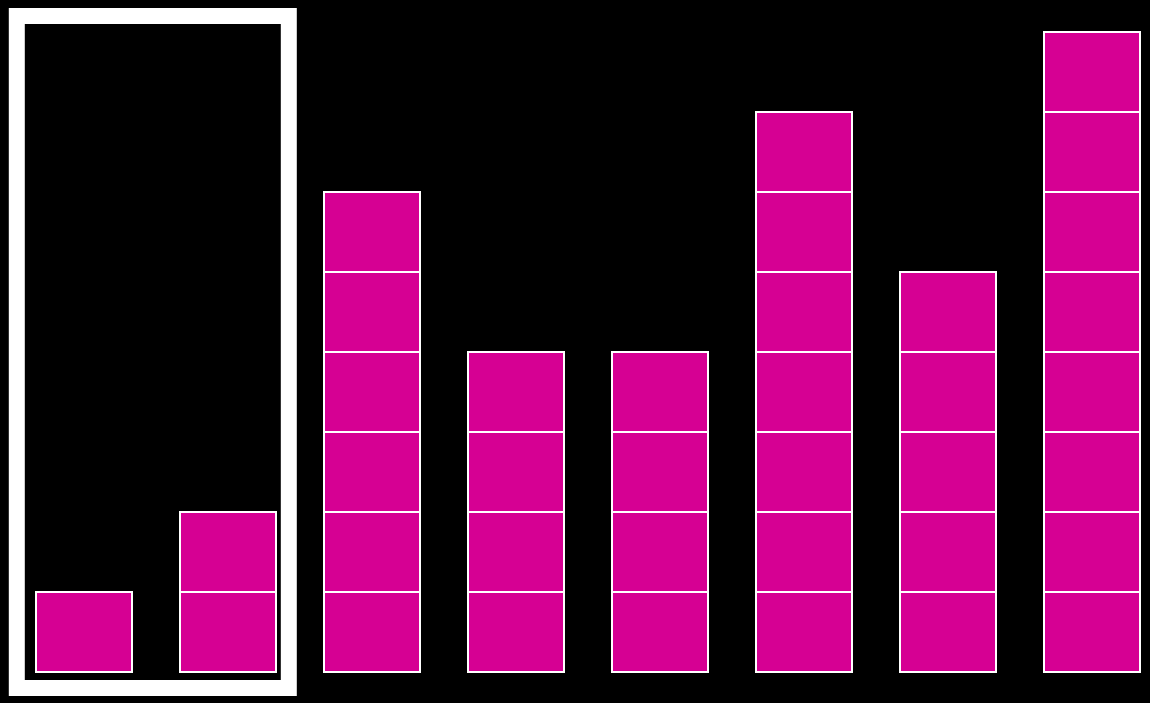
A: No

Sorting: Step B1

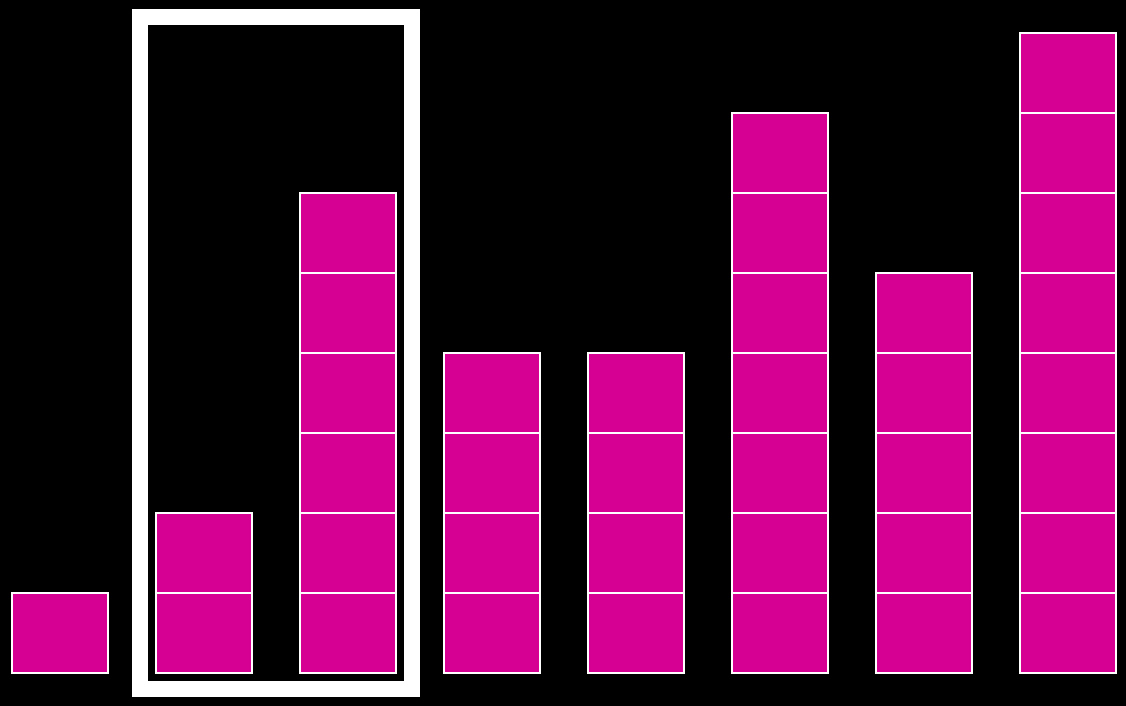


Sorting: Step B1

Swap? Yes

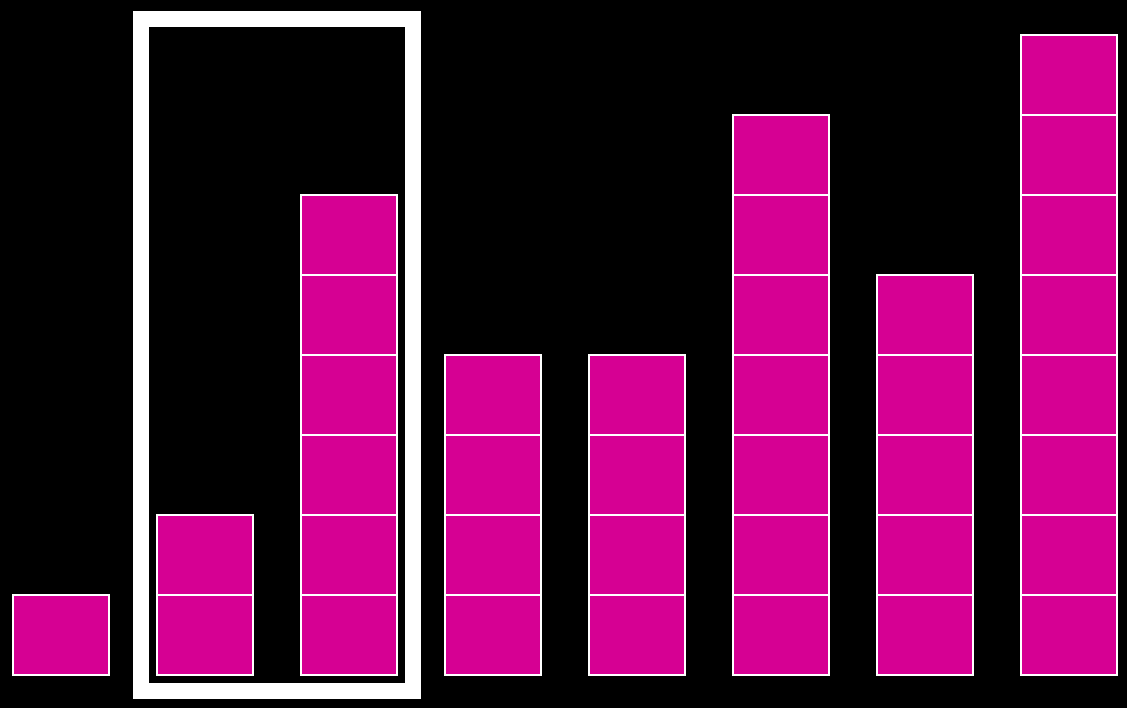


Sorting: Step B2

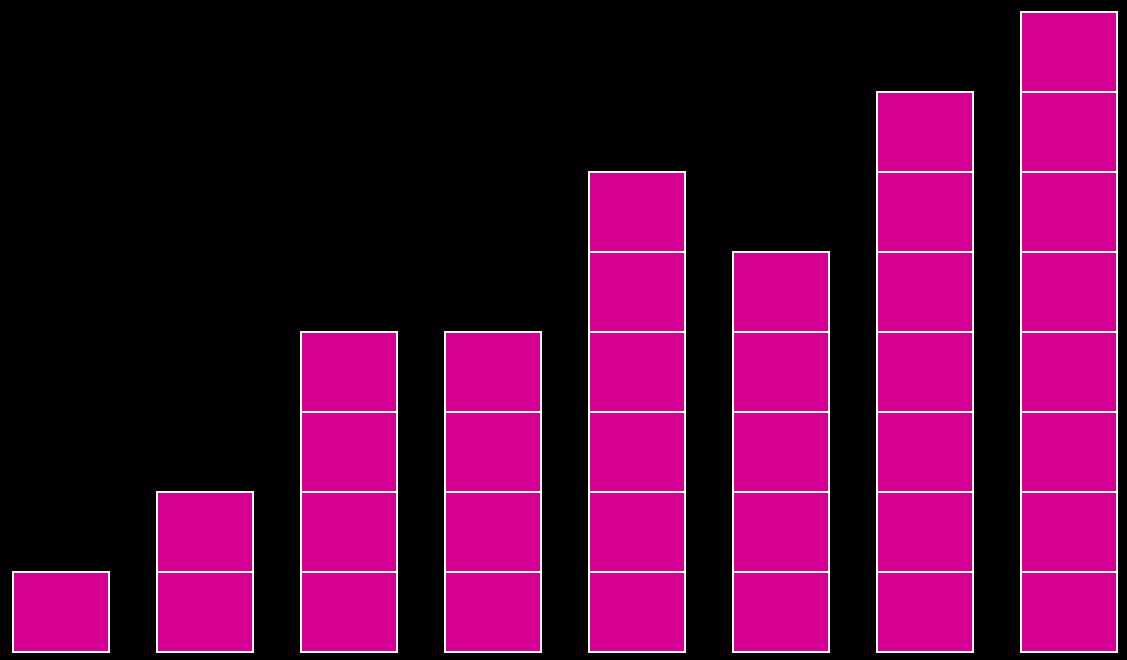


Sorting: Step B2

Swap? No



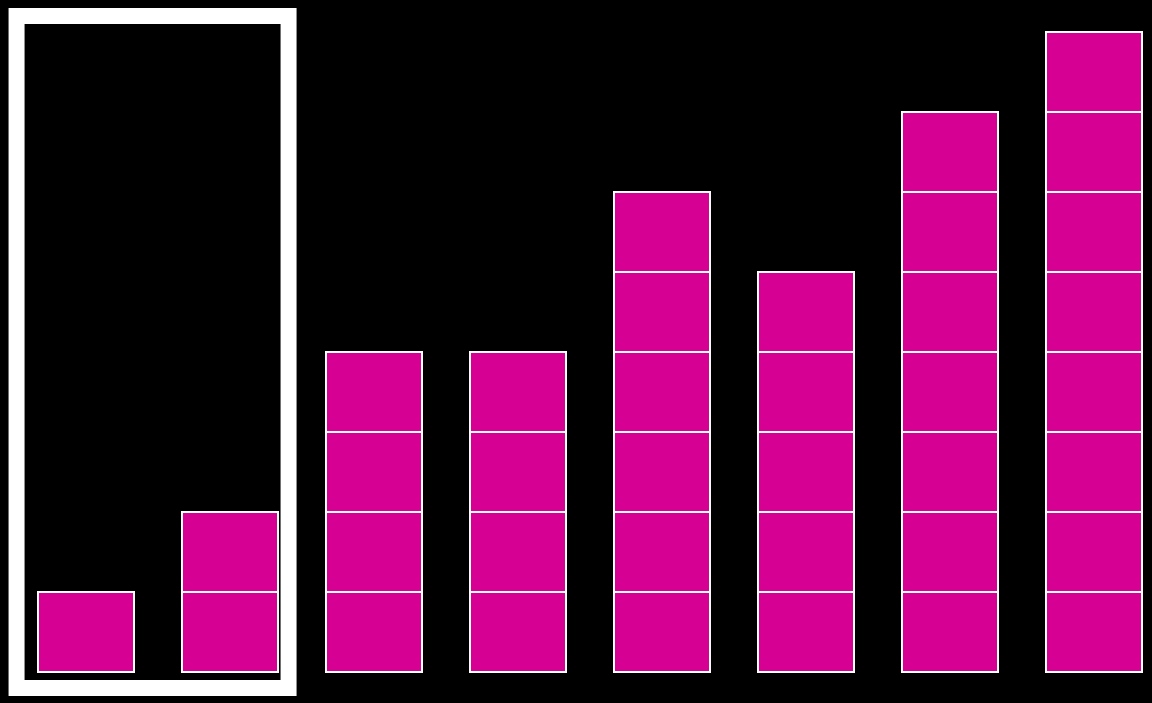
Sorting: After Step B7



Q: Is the array sorted?

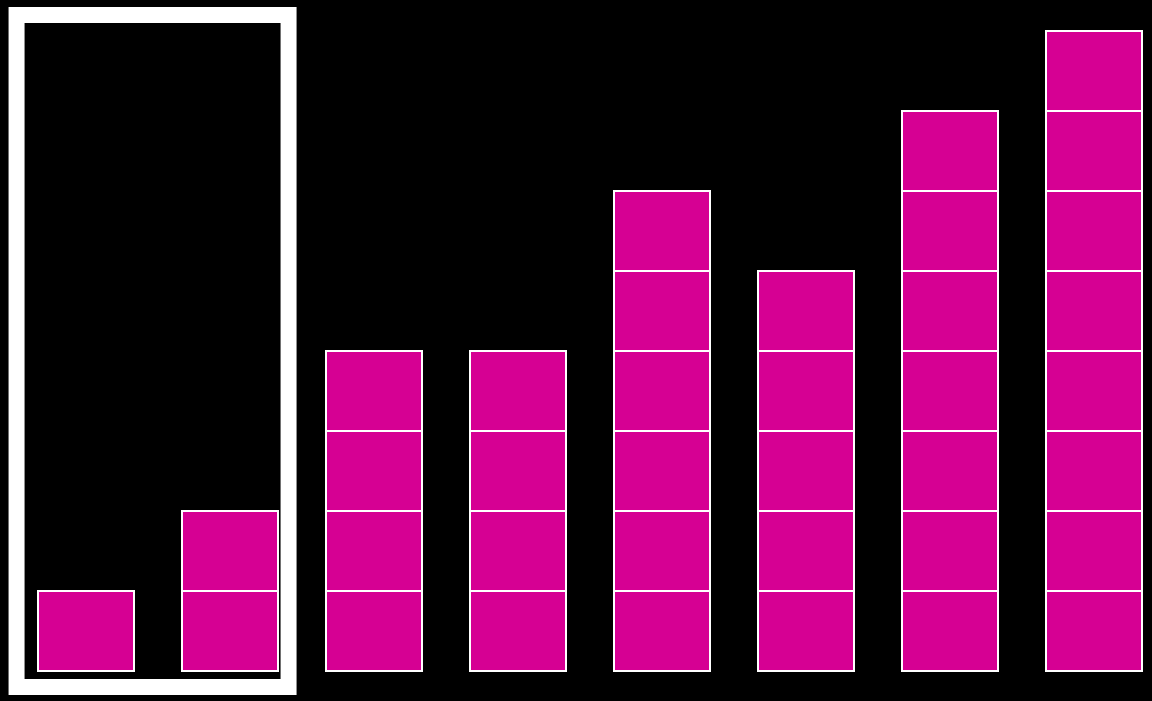
A: No

Sorting: Step C1

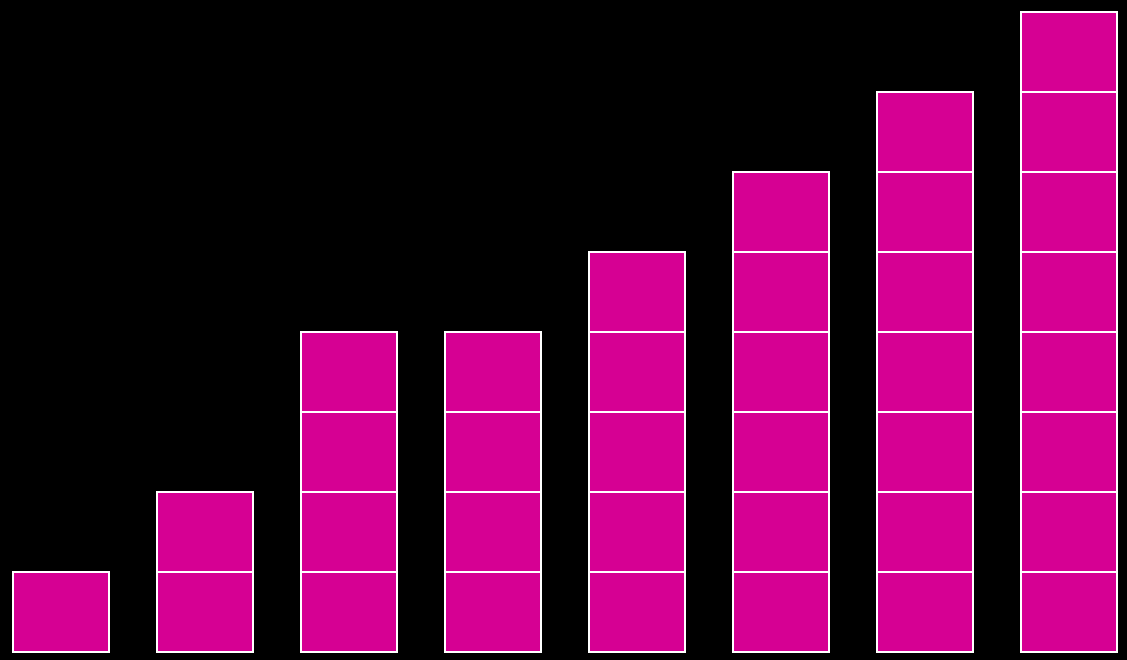


Sorting: Step C1

Swap? No



Sorting: After Step C7



Q: Is the array sorted?

A: Yes

STOP

The Traditional Method

```
int list[]={12,2,8,13,19,2,20,3};
int temp;
for (int n = 0; n< 7;n++)
for (int i = 0; i< 7;i++)
            if (list[i] > list[i +1] )
            { temp = list[i];
              list[i] = list [i + 1];
              list[i + 1] = temp;

            }
```

6.6 Searching Arrays: Linear Search and Binary Search

- Search an array for a *key value*
- Linear search
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays

Linear Search

```
int arr[10]={100,5,3,24,35,46,52,61,71,91};

int key, i=0;
printf("enter key :\n ");
scanf("%d",&key);

while( i <= 9)
}if(key==arr[ i ]){
    printf ("Key found in array[%d]\n",i);
    return 0;}
    i++;
    }
   printf ("Key not found");
```

6.7 Multiple-Subscripted Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (**m** by **n** array)
  - Like matrices: specify row, then column

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Column subscript

Array name

Row subscript

Multiple-Subscripted Arrays

1	2
3	4

- Initialization
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero

    `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`

1	0
3	4

- Referencing elements
  - Specify row, then column

    `printf( "%d", b[ 0 ][ 1 ] );`

Example: strings list

```
char b[10][50];
for (int I = 0; i<=9;i++)
{printf ("Enter the name\n");
scanf("%s",&b[i]);}
for (int i=0; i<=9;i++)
    printf("%s\n",b[i]);
```

6.8 working with Multiple-Subscripted Arrays

- To work with two dimensional array elements we use two for
- For example to print the b array we write
- for(int i=0;i<2;i++)

```
{
    for( int j=0;j<2; j++)
        printf("%d\t", b[i][j]);
    printf("\n");
}
```