Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Sorting and Searching

Quicksort

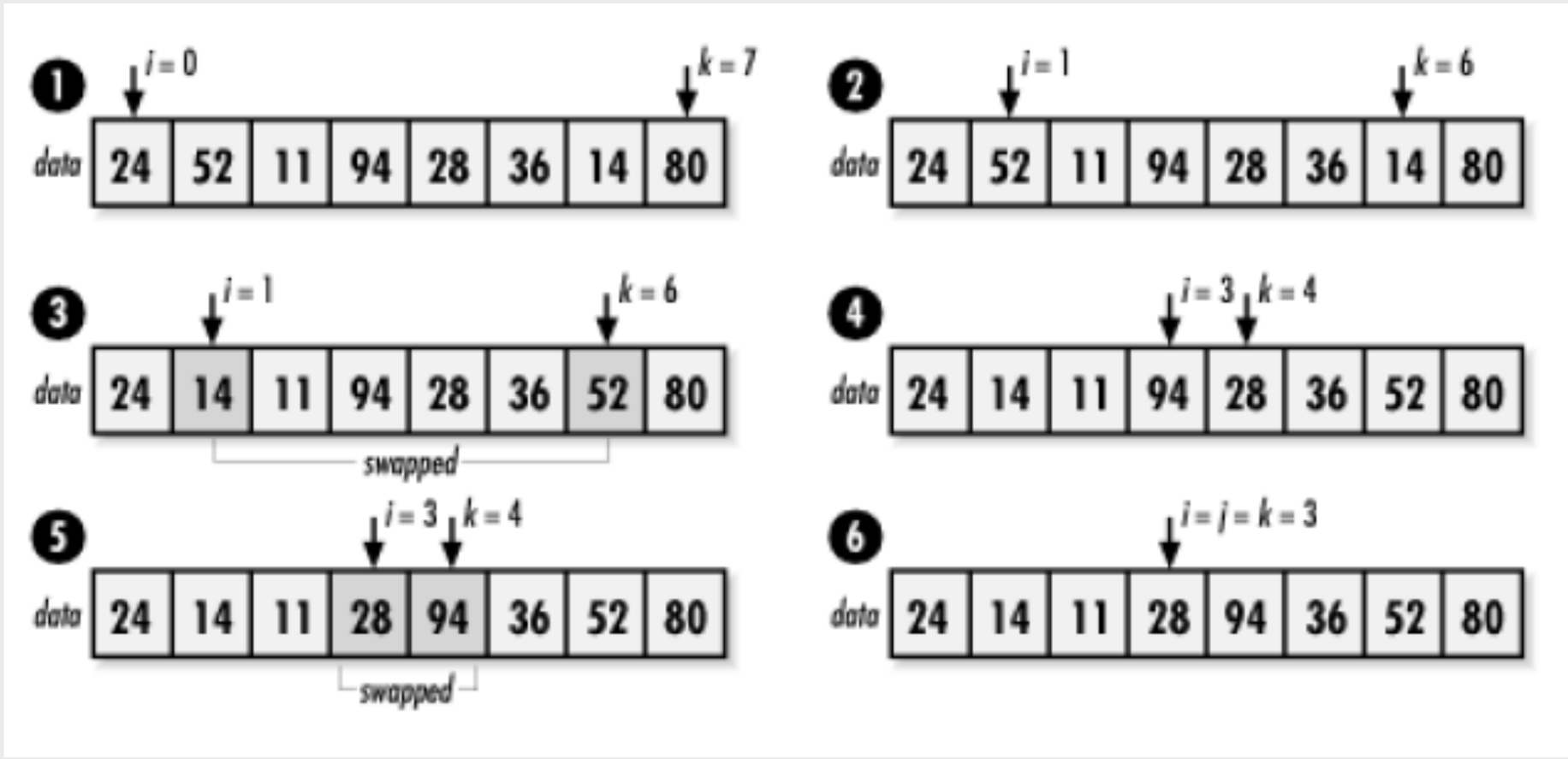
- Description
  - a divide-and-conquer sorting algorithm
  - it’s regarded as the best for general use
  - better choice for medium to large sets of data

Quicksort

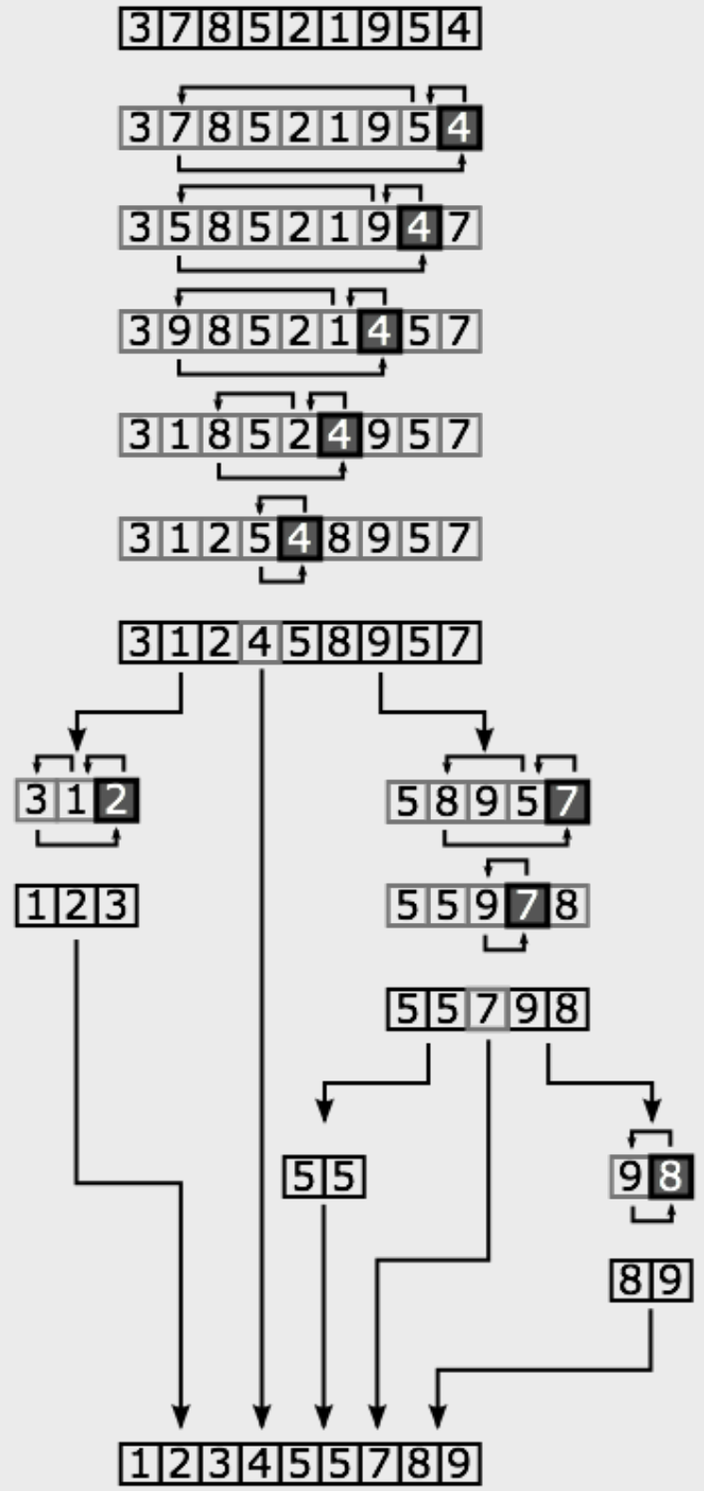
- Three main steps
  - Divide: partition the data into two partitions around a partition value (pivot value)
  - Conquer: sort the two partitions by recursively applying quicksort to them
  - Combine: do nothing since the partitions are sorted after the previous step

Quicksort

- Example: partition around 28



Quicksort

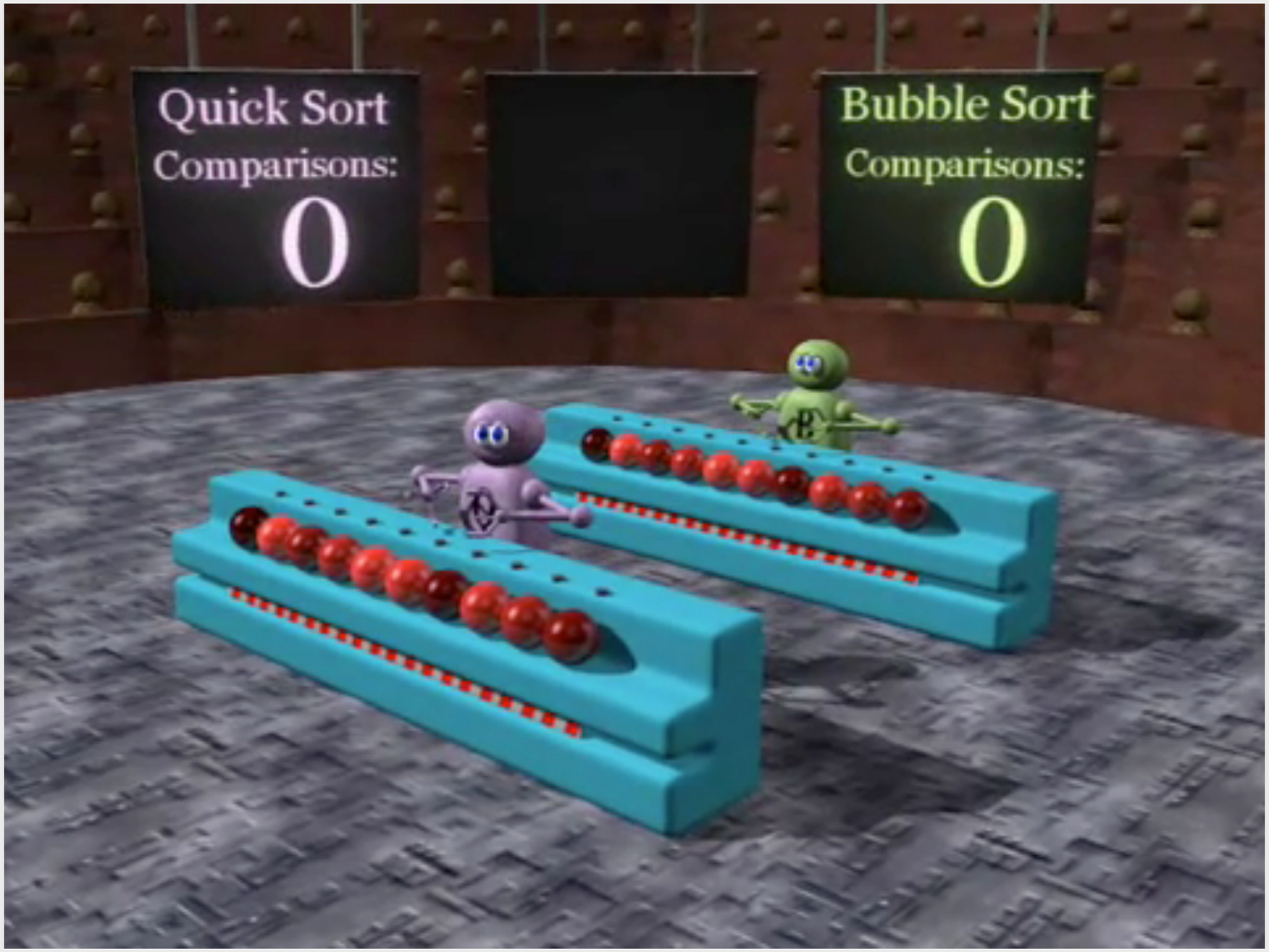


Quicksort

- Video Demo: Quicksort algorithm

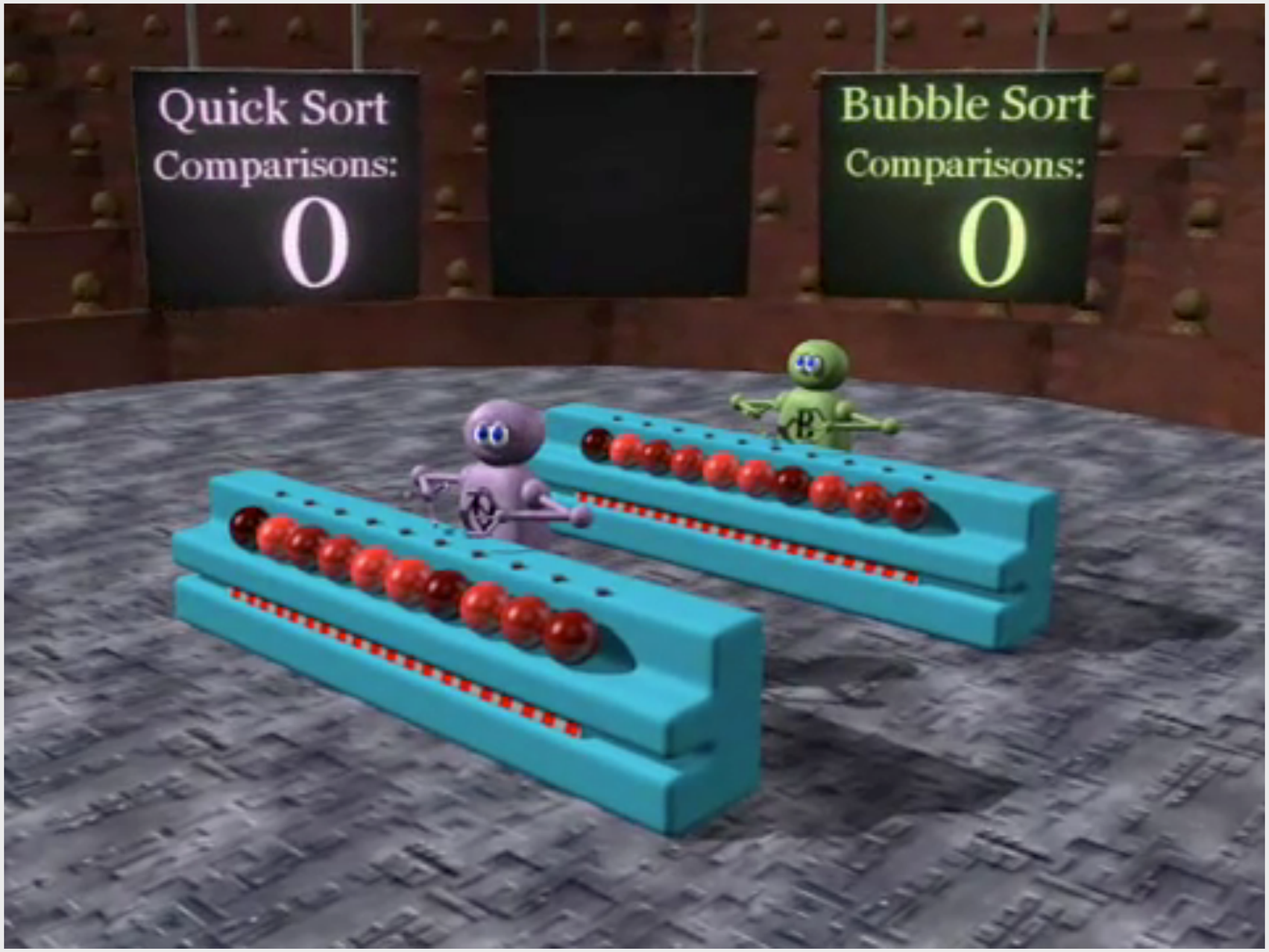
Quicksort

- Video Demo: Visualization of Quick sort



Quicksort

- Video Demo: Visualization of Quick sort



Quicksort

- Choice of pivot/partition value
  - In very early versions of quicksort, the leftmost element of the partition would often be chosen as the pivot element
  - Unfortunately, this causes worst-case behavior on already sorted arrays
  - The problem was easily solved by choosing either a random index for the pivot, or the median of the first, middle and last element of the partition for the pivot

Quicksort

- Example: sort/qksort.c

```
43      /*************************************************
44       *  Use the median-of-three method to find t
45       *************************************************
46      r[0] = (rand() % (k - i + 1)) + i;
47      r[1] = (rand() % (k - i + 1)) + i;
48      r[2] = (rand() % (k - i + 1)) + i;
49      isort(r, 3, sizeof(int), compare_int);
50      memcpy(pval, &a[r[1] * esize], esize);
```

```
58      while (1) {
59          /*************************************************************
60           *  Move left until an element is found in the wrong partition.
61           *************************************************************/
62          do {
63              k--;
64          } while (compare(&a[k * esize], pval) > 0);
65
66          /*************************************************************
67           *  Move right until an element is found in the wrong partition.
68           *************************************************************/
69          do {
70              i++;
71          } while (compare(&a[i * esize], pval) < 0);
72
73          if (i >= k) {
74              /*********************************************************
75               *  Stop partitioning when the left and right counters cross.
76               *********************************************************/
77              break;
78          } else {
79              /*********************************************************
80               *  Swap the elements now under the left and right counters.
81               *********************************************************/
82              memcpy(temp, &a[i * esize], esize);
83              memcpy(&a[i * esize], &a[k * esize], esize);
84              memcpy(&a[k * esize], temp, esize);
85          }
86      }
```

Quicksort

- Summary
  - Analysis of time complexity
    - worst case: $O(n^2)$
    - average-case running time: $O(n \log n)$
  - Not stable
  - in-place sort
  - Not online sort

Merge Sort

- Description
  - another example of divide-and-conquer sorting algorithm
  - Like quicksort, it relies on making comparisons between elements to sort them. However, it doesn’t sort in place

Merge Sort

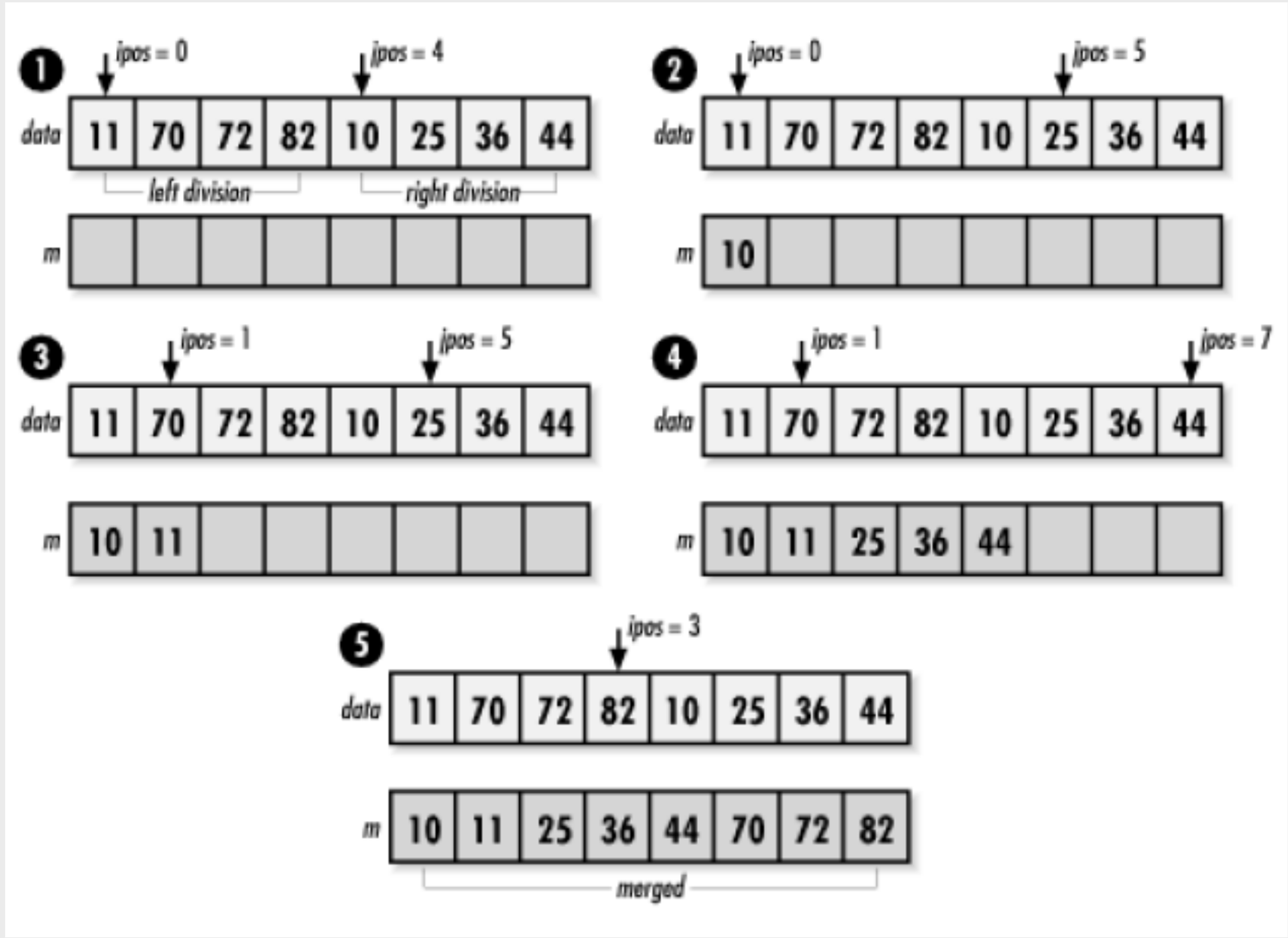
- Three main steps
  - Divide: we divide the data in half
  - Conquer: we sort the two divisions by recursively applying merge sort to them
  - Combine: we merge the two divisions into a single sorted set

Merge Sort

- Distinguishing component of merge sort is its merging process
  - takes two sorted sets and merges them into a single sorted one
- The space requirement of merge sort presents a drawback
  - twice the space of the unsorted data
- Valuable for very large sets of data
  - because it divides the data in manageable ways

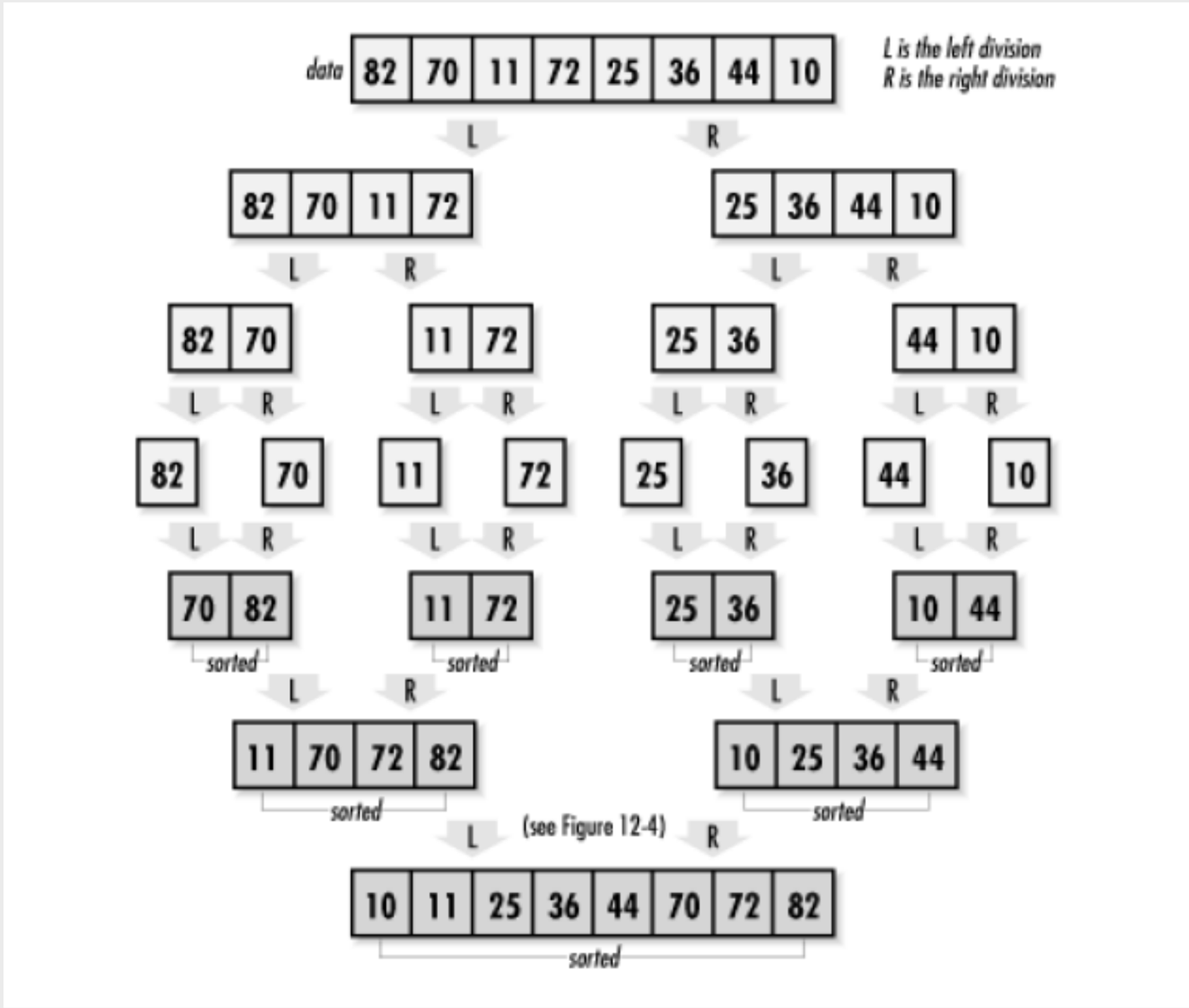
Merge Sort

- Example: Merging two sorted sets



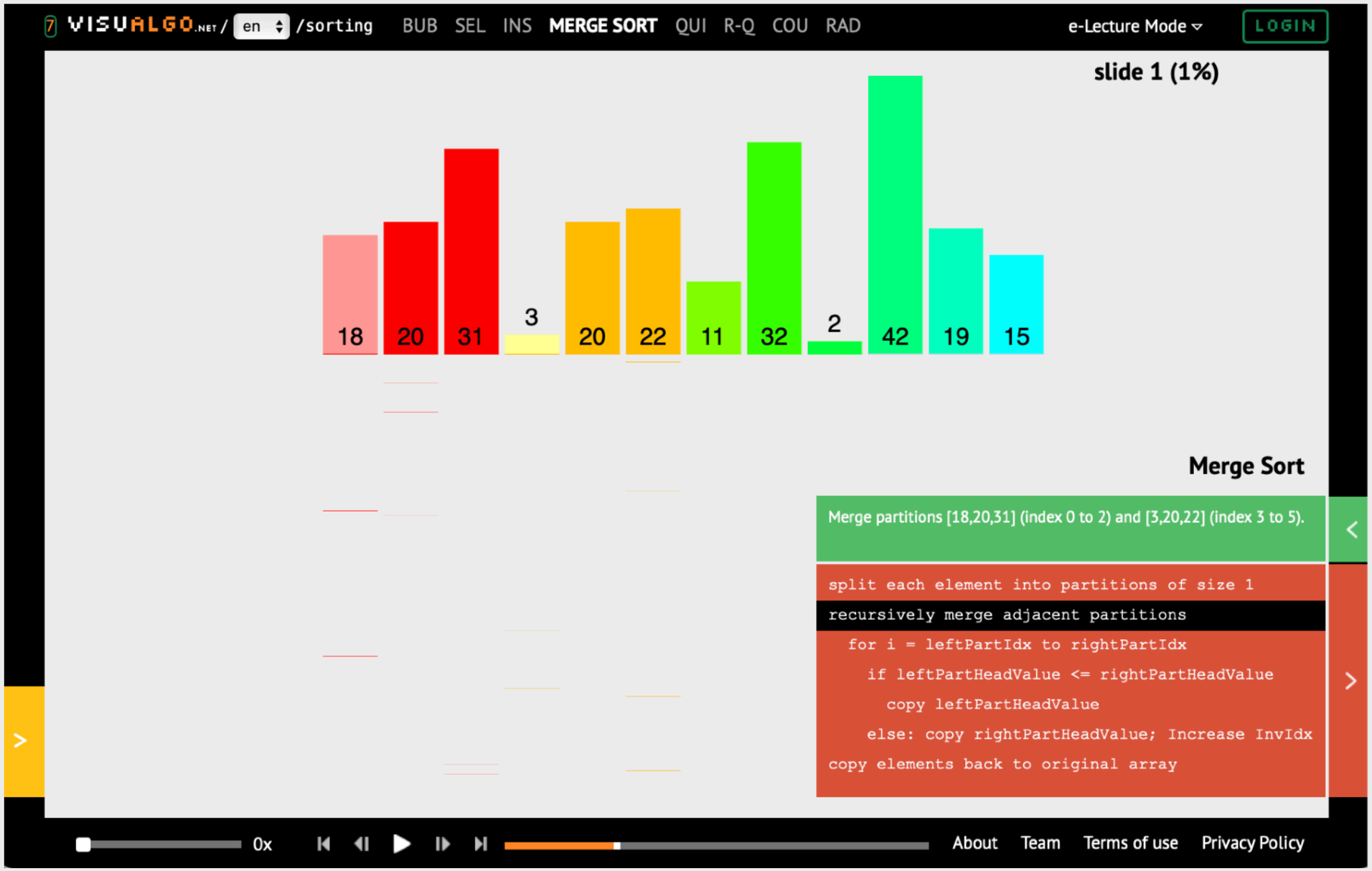
Merge Sort

- Sorting with Merge Sort



Merge Sort

- Video Demo: Merge Sort



Merge Sort

- Example: sort/mgsort.c

```c
static int merge(void *data, int esize, int i, int j, int k,
        int (*compare) (const void *key1, const void *key2)) {

    char                *a = data, *m;
    int                 ipos, jpos, mpos;

    /************************************************************
     *  Initialize the counters used in merging.
     ************************************************************/
    ipos = i;
    jpos = j + 1;
    mpos = 0;

    /************************************************************
     *  Allocate storage for the merged elements.
     ************************************************************/
    if ((m = (char *)malloc(esize * ((k - i) + 1))) == NULL)
        return -1;
```

```c
    /************************************************************
     *  Continue while either division has elements to merge.
     ************************************************************/
    while (ipos <= j || jpos <= k) {
        if (ipos > j) {
            /************************************************************
             *  The left division has no more elements to merge.
             ************************************************************/
            while (jpos <= k) {
                memcpy(&m[mpos * esize], &a[jpos * esize], esize);
                jpos++;
                mpos++;
            }
            continue;
        } else if (jpos > k) {
            /************************************************************
             *  The right division has no more elements to merge.
             ************************************************************/
            while (ipos <= j) {
                memcpy(&m[mpos * esize], &a[ipos * esize], esize);
                ipos++;
                mpos++;
            }
            continue;
        }

        /************************************************************
         *  Append the next ordered element to the merged elements.
         ************************************************************/
        if (compare(&a[ipos * esize], &a[jpos * esize]) < 0) {
            memcpy(&m[mpos * esize], &a[ipos * esize], esize);
            ipos++;
            mpos++;
        } else {
            memcpy(&m[mpos * esize], &a[jpos * esize], esize);
            jpos++;
            mpos++;
        }
    }
```

Merge Sort

- Example: sort/mgsort.c

```
86  int mgsort(void *data, int size, int esize, int i, int k,
87          int (*compare) (const void *key1, const void *key2)) {
88
89      int j;
90      /*************************************************************
91       *  Stop the recursion when no more divisions can be made.
92       *************************************************************
93      if (i < k) {
94          /*********************************************************
95           *  Determine where to divide the elements.
96           *********************************************************
97          j = (int)(((i + k - 1)) / 2);
98
99          /*********************************************************
100          *  Recursively sort the two divisions.
101          *********************************************************
102         if (mgsort(data, size, esize, i, j, compare) < 0)
103             return -1;
104
105         if (mgsort(data, size, esize, j + 1, k, compare) < 0)
106             return -1;
107
108         /*********************************************************
109          *  Merge the two sorted divisions into a single sorted set.
110          *********************************************************
111         if (merge(data, esize, i, j, k, compare) < 0)
112             return -1;
113     }
114     return 0;
```

Merge Sort

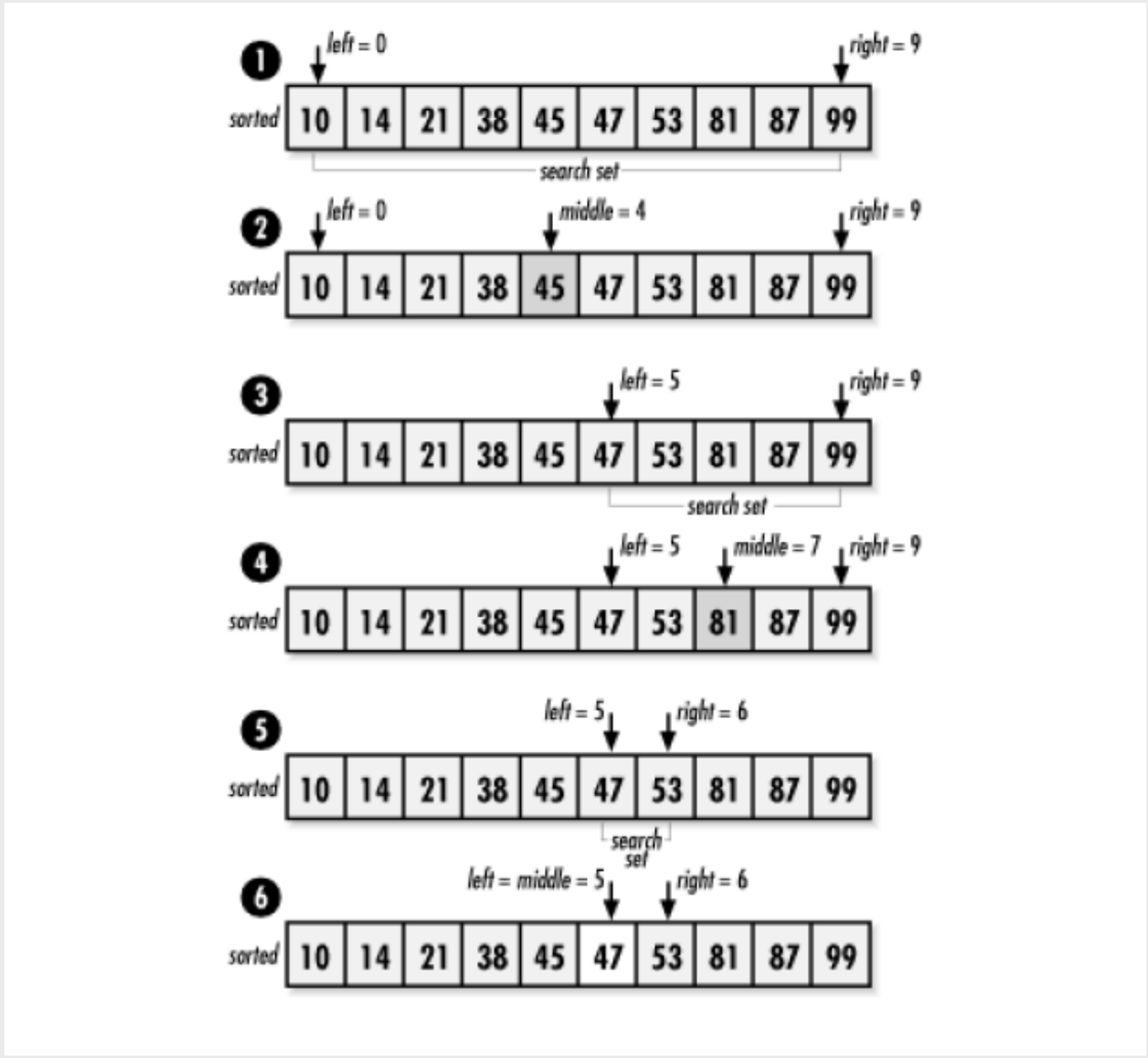
- Analysis of time complexity
  - *logn* levels of division are required
  - for two sorted sets of p and q,
    merging runs in *O(p+q)* is required
  - merge sort runs in time *O(n logn)*
- Not in-place sort: require twice space
- Stable sort
- Not online sort

Binary Search

- Description
  - a technique for searching that works similarly to how we might systematically guess numbers in a guessing game
  - Binary search begins with a set of data that is sorted
  - inspect the middle element of the sorted set

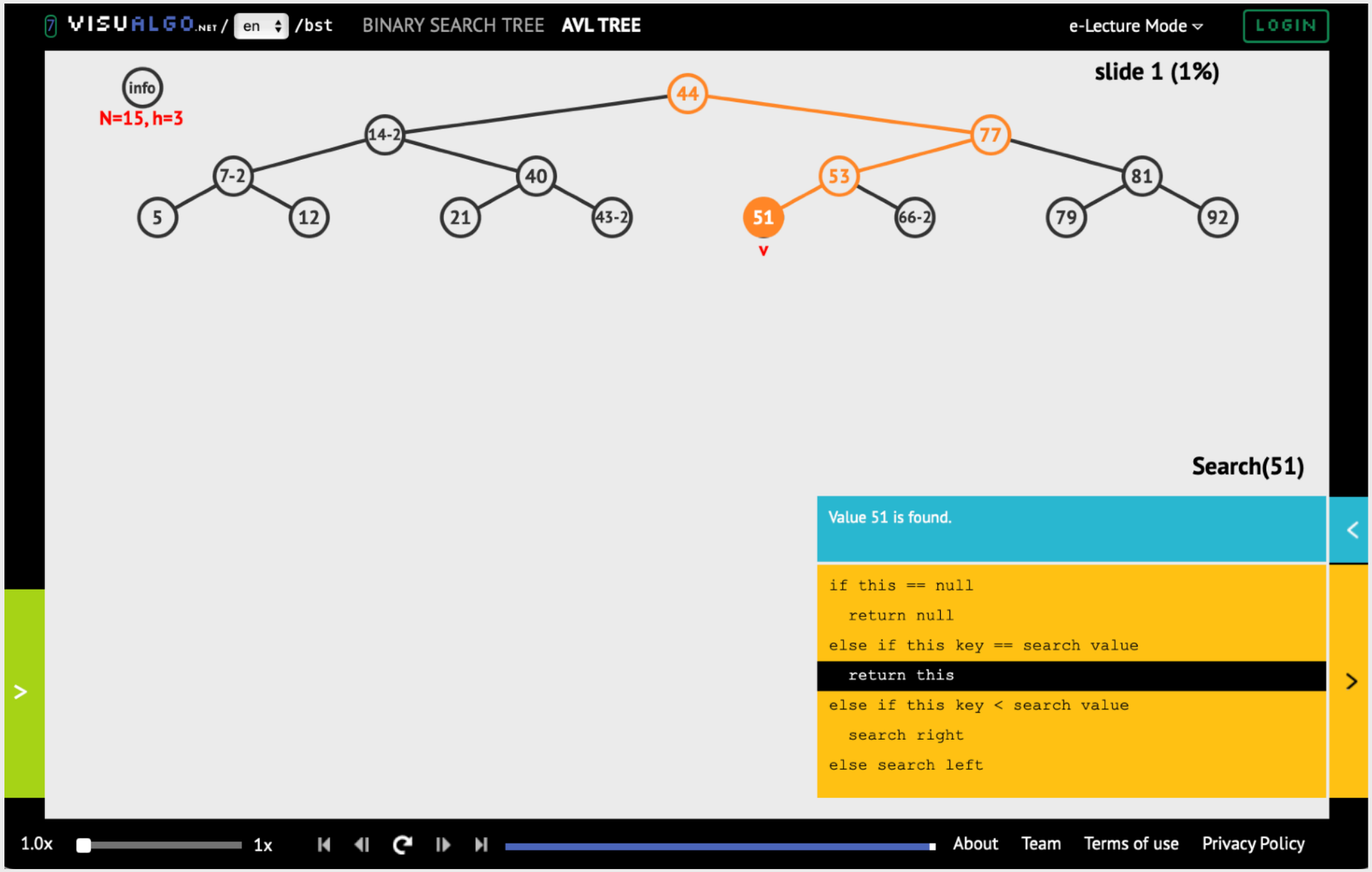
Binary Search

- Searching 47 using binary search



Binary Search

- Video Demo: Binary Search



Binary Search

- Example: search/bisearch.c

```
11 int bisearch(void *sorted, const void *target, int size, int esize, int
12         (*compare)(const void *key1, const void *key2)) {
13
14     int                 left, middle, right;
15     /*************************************************************
16      *  Continue searching until the left and right indices cross.
17      *************************************************************
18     left = 0;
19     right = size - 1;
20
21     while (left <= right) {
22         middle = (left + right) / 2;
23         switch (compare(((char *)sorted + (esize * middle)), target)) {
24             case -1:
25                 /*********************************************************
26                  *  Prepare to search to the right of the middle index.
27                  *********************************************************
28                 left = middle + 1;
29                 break;
30             case 1:
31                 /*********************************************************
32                  *  Prepare to search to the left of the middle index.
33                  *********************************************************
34                 right = middle - 1;
35                 break;
36             case 0:
37                 /*********************************************************
38                  *  Return the exact index where the data has been found.
39                  *********************************************************
40                 return middle;
41         }
42     }
```

Binary Search

- Analysis of time complexity
  - the time complexity of binary search depends on the maximum number of divisions possible during the searching process
    - for a set of n elements, up to *logn* divisions
    - worst case: when the target is not found
    - the time complexity of binary search is *O(logn)*

Advanced Bitwise Operations

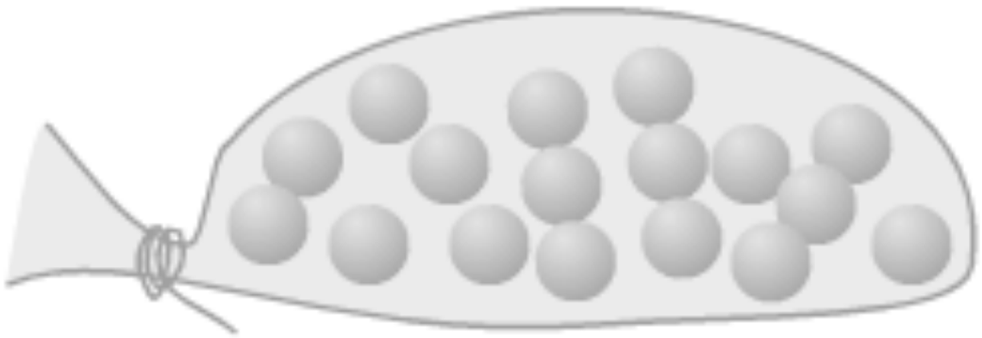
Bit

- A bit is the smallest unit of information
- Eight bits together form a byte, represented by the C data type char
- **01100100** ==> **0x64**; **10101111** ==> **0xAF**

Hexadecimal	Binary	Hexadecimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Bit

- The **`printf`** format
  - **`%x`** for hexadecimal
  - **`%o`** for octal



Representation	Number of marbles
Hexadecimal	0x11
Decimal	17
Octal	021
Binary	10001

Bit Operations

- Bit operators are usually efficient!!

Operator	Meaning
&	Bitwise and
|	Bitwise or
^	Bitwise exclusive or
~	Complement
<<	Shift left
>>	Shift right

The and Operator (&)

Bit1	Bit2	Bit1 & Bit2
0	0	0
0	1	0
1	0	0
1	1	1

- Example

```
   c1 = 0x45    binary 01000101
&  c2 = 0x71    binary 01110001

-------------------------------
=  0x41         binary 01000001
```

The and Operator (&)

- Example: bit/and.c

```
6     i1 = 4;
7     i2 = 2;
8
9     /* Nice way of writing the conditional */
10    if ((i1 != 0) && (i2 != 0))
11        printf("Both are not zero\n");
12
13    /* Shorthand way of doing the same thing */
14    /* Correct C code, but rotten style */
15    if (i1 && i2)
16        printf("Both are not zero\n");
17
18    /* Incorrect use of bitwise and resulting in an error */
19    if (i1 & i2)
20        printf("Both are not zero\n");
```

Note: the difference between **&** and **&&**

The and Operator (&)

- How to examine if a number is even or odd?
- Example: bit/even.c

```
10      i1 = 46;
11      i2 = 73;
12
13      if (even(i1)){
14          printf("%d is an even number.\n", i1);
15      } else {
16          printf("%d is an odd number.\n", i1);
17      }
18
19      if (even(i2)){
20          printf("%d is an even number.\n", i2);
21      } else {
22          printf("%d is an odd number.\n", i2);
23      }
```

The and Operator (&)

- How to examine if a number is even or odd?
- Example: bit/even.c

```
10      i1 = 46;
11      i2 = 73;
12
13      if (even(i1)){
14          printf("%d is an even number.\n", i1);
15      } else {
16          printf("%d is an odd number.\n", i1);
17      }
18
19      if (even(i2)){
20          printf("%d is an even number.\n", i2);
21      } else {
22          printf("%d is an odd number.\n", i2);
23      }
```

```
3  int even(const int value) {
4      return ((value & 1) == 0);
5  }
```

The and Operator (&)

- How to examine if a number is even or odd?
- Example: bit/even.c

```
10    i1 = 46;
11    i2 = 73;
12
13    if (even(i1)){
14        printf("%d is an even number.\n", i1);
15    } else {
16        printf("%d is an odd number.\n", i1);
17    }
18
19    if (even(i2)){
20        printf("%d is an even number.\n", i2);
21    } else {
22        printf("%d is an odd number.\n", i2);
23    }
```

```
3 int even(const int value) {
4     return ((value & 1) == 0);
5 }
```

The and Operator (&)

- How can you quickly determine whether a number is a power of 2?

The and Operator (&)

- How can you quickly determine whether a number is a power of 2?
  - Check whether x & (x-1) is 0

Bitwise or (|)

- Operation

0	0	0
0	1	1
1	0	1
1	1	1

```
    i1=0x47    01000111
|   i2=0x53    01010011
-------------------------
=   0x57       01010111
```

The Bitwise Exclusive or (^)

- Operation

Bit1	Bit2	Bit1 ^ Bit2
0	0	0
0	1	1
1	0	1
1	1	0

```
    i1=0x47    01000111
^   i2=0x53    01010011

-------------------------

=   0x14       00010100
```

The Ones Complement Operator (Not) (~)

- Operation

Bit	~Bit
0	1
1	0

```
c=    0x45      01000101

-------------------------------

~c=   0xBA      10111010
```

The Left- and Right-Shift Operators (<<, >>)

- Operation

	c=0x1C	00011100
c << 1	c=0x38	00111000
c >> 2	c=0x07	00000111

- Shifting left by one (**`x<<1`**)
  - same as multiplying by **2**
- Shifting right by one (**`x>>1`**)
  - same as dividing by **2**
- Shifting is faster than multiplication!!

Right-Shift Details

	signed char	signed char	unsigned char
Expression	9 >> 2	-8 >> 2	248 >> 2
Binary Value >> 2	0000 1001 >> 2	1111 1000 >> 2	1111 1000 >> 2
Result	??00 0010	??11 1110	??11 1110
Fill	Sign Bit (0)	Sign Bit (1)[3]	Zero
Final Result (Binary)	0000 0010	1111 1110	0011 1110
Final Result (short int)	2	-2	62

Right-Shift Details

	signed char	signed char	unsigned char
Expression	9 >> 2	-8 >> 2	248 >> 2
Binary Value >> 2	0000 1001 >> 2	1111 1000 >> 2	1111 1000 >> 2
Result	??00 0010	??11 1110	??11 1110
Fill	Sign Bit (0)	Sign Bit (1)[3]	Zero
Final Result (Binary)	0000 0010	1111 1110	0011 1110
Final Result (short int)	2	-2	62

Right-Shift Details

	signed char	signed char	unsigned char
Expression	9 >> 2	-8 >> 2	248 >> 2
Binary Value >> 2	0000 1001 >> 2	1111 1000 >> 2	1111 1000 >> 2
Result	??00 0010	??11 1110	??11 1110
Fill	Sign Bit (0)	Sign Bit (1)[3]	Zero
Final Result (Binary)	0000 0010	1111 1110	0011 1110
Final Result (short int)	2	-2	62

Right-Shift Details

	signed char	signed char	unsigned char
Expression	9 >> 2	-8 >> 2	248 >> 2
Binary Value >> 2	0000 1001 >> 2	1111 1000 >> 2	1111 1000 >> 2
Result	??00 0010	??11 1110	??11 1110
Fill	Sign Bit (0)	Sign Bit (1)[3]	Zero
Final Result (Binary)	0000 0010	1111 1110	0011 1110
Final Result (short int)	2	-2	62

Right-Shift Details

	signed char	signed char	unsigned char
Expression	9 >> 2	-8 >> 2	248 >> 2
Binary Value >> 2	0000 1001 >> 2	1111 1000 >> 2	1111 1000 >> 2
Result	??00 0010	??11 1110	??11 1110
Fill	Sign Bit (0)	Sign Bit (1)[3]	Zero
Final Result (Binary)	0000 0010	1111 1110	0011 1110
Final Result (short int)	2	-2	62

Note: only positive numbers need the 2's complement transformation!!

As to how to conduct the transformation, please refer to the slides of further information!!

Setting, Clearing, and Testing Bits

- A character (**char**) contains eight bits. Each of these can be treated as a separate flag.
- Take a low-level communication as an example
- Use an 8-bit status character instead of five bytes storage

Name	Description
ERROR	True if any error is set.
FRAMING_ERROR	A framing error occurred for this character.
PARITY_ERROR	Character had the wrong parity.
CARRIER_LOST	The carrier signal went down.
CHANNEL_DOWN	Power was lost on the communication device.

Setting, Clearing, and Testing Bits

- Assign the flags as follows

Bit	Name
0	ERROR
1	FRAMING_ERROR
2	PARITY_ERROR
3	CARRIER_LOST
4	CHANNEL_DOWN

- Bits are numbered 76543210 by convention

Bit numbers							
7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0

Setting, Clearing, and Testing Bits

- Flags

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

```
/* True if any error is set */
const int ERROR =      0x01;

/* A framing error occurred for this character */
const int FRAMING_ERROR =  0x02;

/* Character had the wrong parity */
const int PARITY_ERROR = 0x04;

/* The carrier signal went down */
const int CARRIER_LOST =   0x08;

/* Power was lost on the communication device */
const int CHANNEL_DOWN =   0x10;
```

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

```
/* True if any error is set */
const int ERROR =      0x01;

/* A framing error occurred for this character */
const int FRAMING_ERROR =  0x02;

/* Character had the wrong parity */
const int PARITY_ERROR = 0x04;

/* The carrier signal went down */
const int CARRIER_LOST =   0x08;

/* Power was lost on the communication device */
const int CHANNEL_DOWN =   0x10;
```

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

```
/* True if any error is set */
const int ERROR =      0x01;

/* A framing error occurred for this character */
const int FRAMING_ERROR =  0x02;

/* Character had the wrong parity */
const int PARITY_ERROR = 0x04;

/* The carrier signal went down */
const int CARRIER_LOST =   0x08;

/* Power was lost on the communication device */
const int CHANNEL_DOWN =   0x10;
```

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

```
/* True if any error is set */
const int ERROR =      0x01;

/* A framing error occurred for this character */
const int FRAMING_ERROR =  0x02;

/* Character had the wrong parity */
const int PARITY_ERROR = 0x04;

/* The carrier signal went down */
const int CARRIER_LOST =   0x08;

/* Power was lost on the communication device */
const int CHANNEL_DOWN =   0x10;
```

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

```
/* True if any error is set */
const int ERROR =      0x01;

/* A framing error occurred for this character */
const int FRAMING_ERROR =  0x02;

/* Character had the wrong parity */
const int PARITY_ERROR = 0x04;

/* The carrier signal went down */
const int CARRIER_LOST =   0x08;

/* Power was lost on the communication device */
const int CHANNEL_DOWN =   0x10;
```

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

```
/* True if any error is set */
const int ERROR =      0x01;

/* A framing error occurred for this character */
const int FRAMING_ERROR =  0x02;

/* Character had the wrong parity */
const int PARITY_ERROR = 0x04;

/* The carrier signal went down */
const int CARRIER_LOST =   0x08;

/* Power was lost on the communication device */
const int CHANNEL_DOWN =   0x10;
```

Setting, Clearing, and Testing Bits

- Flags

Bit	Binary value	Hexadecimal constant
7	10000000	0x80
6	01000000	0x40
5	00100000	0x20
4	00010000	0x10
3	00001000	0x08
2	00000100	0x04
1	00000010	0x02
0	00000001	0x01

```
/* True if any error is set */
const int ERROR =      0x01;

/* A framing error occurred for this character */
const int FRAMING_ERROR =  0x02;

/* Character had the wrong parity */
const int PARITY_ERROR = 0x04;

/* The carrier signal went down */
const int CARRIER_LOST =   0x08;

/* Power was lost on the communication device */
const int CHANNEL_DOWN =   0x10;
```

Confusing!!
use left-shift (<<) instead

Setting, Clearing, and Testing Bits

- Left-Shift Operator and Bit Definition

Setting, Clearing, and Testing Bits

- Left-Shift Operator and Bit Definition

C Representation	Base 2 Equivalent	Result (Base 2)	Bit Number
1<<0	00000001 << 0	00000001	Bit 0
1<<1	00000001 << 1	00000010	Bit 1
1<<2	00000001 << 2	00000100	Bit 2
1<<3	00000001 << 3	00001000	Bit 3
1<<4	00000001 << 4	00010000	Bit 4
1<<5	00000001 << 5	00100000	Bit 5
1<<6	00000001 << 6	01000000	Bit 6
1<<7	00000001 << 7	10000000	Bit 7

Setting, Clearing, and Testing Bits

- Left-Shift Operator and Bit Definition

C Representation	Base 2 Equivalent	Result (Base 2)	Bit Number
1<<0	00000001 << 0	00000001	Bit 0
1<<1	00000001 << 1	00000010	Bit 1
1<<2	00000001 << 2	00000100	Bit 2
1<<3	00000001 << 3	00001000	Bit 3
1<<4	00000001 << 4	00010000	Bit 4
1<<5	00000001 << 5	00100000	Bit 5
1<<6	00000001 << 6	01000000	Bit 6
1<<7	00000001 << 7	10000000	Bit 7

use **`1<<4`** to replace **`0x10`**

because you can easily tell what bit is set by **`1<<4`**

Setting, Clearing, and Testing Bits

- Define the flags

```
/* True if any error is set */
const int ERROR =           (1<<0);

/* A framing error occurred for this character  */
const int FRAMING_ERROR =   (1<<1);

/* Character had the wrong parity */
const int PARITY_ERROR =    (1<<2);

/* The carrier signal went down */
const int CARRIER_LOST =    (1<<3);

/* Power was lost on the communication device */
const int CHANNEL_DOWN =    (1<<4);
```

Setting, Clearing, and Testing Bits

- Define the flags

```
/* True if any error is set */
const int ERROR =           (1<<0);

/* A framing error occurred for this character  */
const int FRAMING_ERROR =   (1<<1);

/* Character had the wrong parity */
const int PARITY_ERROR =    (1<<2);

/* The carrier signal went down */
const int CARRIER_LOST =    (1<<3);

/* Power was lost on the communication device */
const int CHANNEL_DOWN =    (1<<4);
```

Setting, Clearing, and Testing Bits

- Define the flags

```
/* True if any error is set */
const int ERROR =           (1<<0);

/* A framing error occurred for this character  */
const int FRAMING_ERROR =   (1<<1);

/* Character had the wrong parity */
const int PARITY_ERROR =    (1<<2);

/* The carrier signal went down */
const int CARRIER_LOST =    (1<<3);

/* Power was lost on the communication device */
const int CHANNEL_DOWN =    (1<<4);
```

Setting, Clearing, and Testing Bits

- Define the flags

```
/* True if any error is set */
const int ERROR =           (1<<0);

/* A framing error occurred for this character  */
const int FRAMING_ERROR =   (1<<1);

/* Character had the wrong parity */
const int PARITY_ERROR =    (1<<2);

/* The carrier signal went down */
const int CARRIER_LOST =    (1<<3);

/* Power was lost on the communication device */
const int CHANNEL_DOWN =    (1<<4);
```

Setting, Clearing, and Testing Bits

- Define the flags

```
/* True if any error is set */
const int ERROR =           (1<<0);

/* A framing error occurred for this character  */
const int FRAMING_ERROR =   (1<<1);

/* Character had the wrong parity */
const int PARITY_ERROR =    (1<<2);

/* The carrier signal went down */
const int CARRIER_LOST =    (1<<3);

/* Power was lost on the communication device */
const int CHANNEL_DOWN =    (1<<4);
```

Setting, Clearing, and Testing Bits

- Define the flags

```
/* True if any error is set */
const int ERROR =           (1<<0);

/* A framing error occurred for this character  */
const int FRAMING_ERROR =   (1<<1);

/* Character had the wrong parity */
const int PARITY_ERROR =    (1<<2);

/* The carrier signal went down */
const int CARRIER_LOST =    (1<<3);

/* Power was lost on the communication device */
const int CHANNEL_DOWN =    (1<<4);
```

Setting, Clearing, and Testing Bits

- After defining the flags, we can manipulate them
- To set a bit, use the | operator

```
char    flags = 0;  /* start all flags at 0 */

flags |= CHANNEL_DOWN; /* Channel just died */
```

- To test a bit, use the **&** operator and mask out the bits

```
if ((flags & ERROR) != 0)
    printf("Error flag is set\n");
else
    printf("No error detected\n");
```

Setting, Clearing, and Testing Bits

- After defining the flags, we can manipulate them
- To set a bit, use the | operator

```
char    flags = 0;  /* start all flags at 0 */

    flags |= CHANNEL_DOWN; /* Channel just died */
```

- To test a bit, use the **&** operator and mask out the bits

```
if ((flags & ERROR) != 0)
    printf("Error flag is set\n");
else
    printf("No error detected\n");
```

Setting, Clearing, and Testing Bits

- After defining the flags, we can manipulate them
- To set a bit, use the | operator

```
char    flags = 0;  /* start all flags at 0 */

    flags |= CHANNEL_DOWN; /* Channel just died */
```

- To test a bit, use the **&** operator and mask out the bits

```
if ((flags & ERROR) != 0)
    printf("Error flag is set\n");
else
    printf("No error detected\n");
```

Setting, Clearing, and Testing Bits

- Clearing a bit is a harder task.
  - Suppose we want to clear the bit **`PARITY_ERROR (00000100)`**.
  - Create a mask **`(11111011)`**
  - The mask is then anded with the flag to clear the bit

```
PARITY_ERROR                    00000100
~PARITY_ERROR                   11111011
flags                           00000101

_____________________________________________

flags & ~PARITY_ERROR           00000001
```

```
flags &= ~PARITY_ERROR; /* Who cares about parity */
```

Setting, Clearing, and Testing Bits

- Clearing a bit is a harder task.
  - Suppose we want to clear the bit **`PARITY_ERROR (00000100)`**.
  - Create a mask **`(11111011)`**
  - The mask is then anded with the flag to clear the bit

```
PARITY_ERROR                    00000100
~PARITY_ERROR                   11111011
flags                           00000101

_____________________________________________

flags & ~PARITY_ERROR           00000001
```

```
flags &= ~PARITY_ERROR; /* Who cares about parity */
```

Setting, Clearing, and Testing Bits

- Example: bit/high.c

Setting, Clearing, and Testing Bits

- Example: bit/high.c

```
3 const int HIGH_SPEED = (1<<3);
4 const int DIRECT_CONNECT = (1<<5);
5
6 char flags = 0;
```

Setting, Clearing, and Testing Bits

- Example: bit/high.c

```
3 const int HIGH_SPEED = (1<<3);
4 const int DIRECT_CONNECT = (1<<5);
5
6 char flags = 0;
```

Setting, Clearing, and Testing Bits

- Example: bit/high.c

```
3  const int HIGH_SPEED = (1<<3);
4  const int DIRECT_CONNECT = (1<<5);
5
6  char flags = 0;
```

```
10     flags |= HIGH_SPEED;
11     flags |= DIRECT_CONNECT;
12
13     if ((flags & HIGH_SPEED) != 0)
14         printf("High speed is set\n");
15     else
16         printf("High speed is not set\n");
17
18     if ((flags & DIRECT_CONNECT) != 0)
19         printf("Direct connect is set\n");
20     else
21         printf("Direct connect is not set\n");
```

Setting, Clearing, and Testing Bits

- Example: bit/high.c

```
3  const int HIGH_SPEED = (1<<3);
4  const int DIRECT_CONNECT = (1<<5);
5
6  char flags = 0;
```

```
10     flags |= HIGH_SPEED;
11     flags |= DIRECT_CONNECT;
12
13     if ((flags & HIGH_SPEED) != 0)
14         printf("High speed is set\n");
15     else
16         printf("High speed is not set\n");
17
18     if ((flags & DIRECT_CONNECT) != 0)
19         printf("Direct connect is set\n");
20     else
21         printf("Direct connect is not set\n");
```

Setting, Clearing, and Testing Bits

- Example: bit/high.c

```
3  const int HIGH_SPEED = (1<<3);
4  const int DIRECT_CONNECT = (1<<5);
5
6  char flags = 0;
```

```
10     flags |= HIGH_SPEED;
11     flags |= DIRECT_CONNECT;
12
13     if ((flags & HIGH_SPEED) != 0)
14         printf("High speed is set\n");
15     else
16         printf("High speed is not set\n");
17
18     if ((flags & DIRECT_CONNECT) != 0)
19         printf("Direct connect is set\n");
20     else
21         printf("Direct connect is not set\n");
```

```
23     flags &= ~HIGH_SPEED;
24     flags &= ~DIRECT_CONNECT;
25
26     if ((flags & HIGH_SPEED) != 0)
27         printf("High speed is set\n");
28     else
29         printf("High speed is not set\n");
30
31     if ((flags & DIRECT_CONNECT) != 0)
32         printf("Direct connect is set\n");
33     else
34         printf("Direct connect is not set\n");
```

Setting, Clearing, and Testing Bits

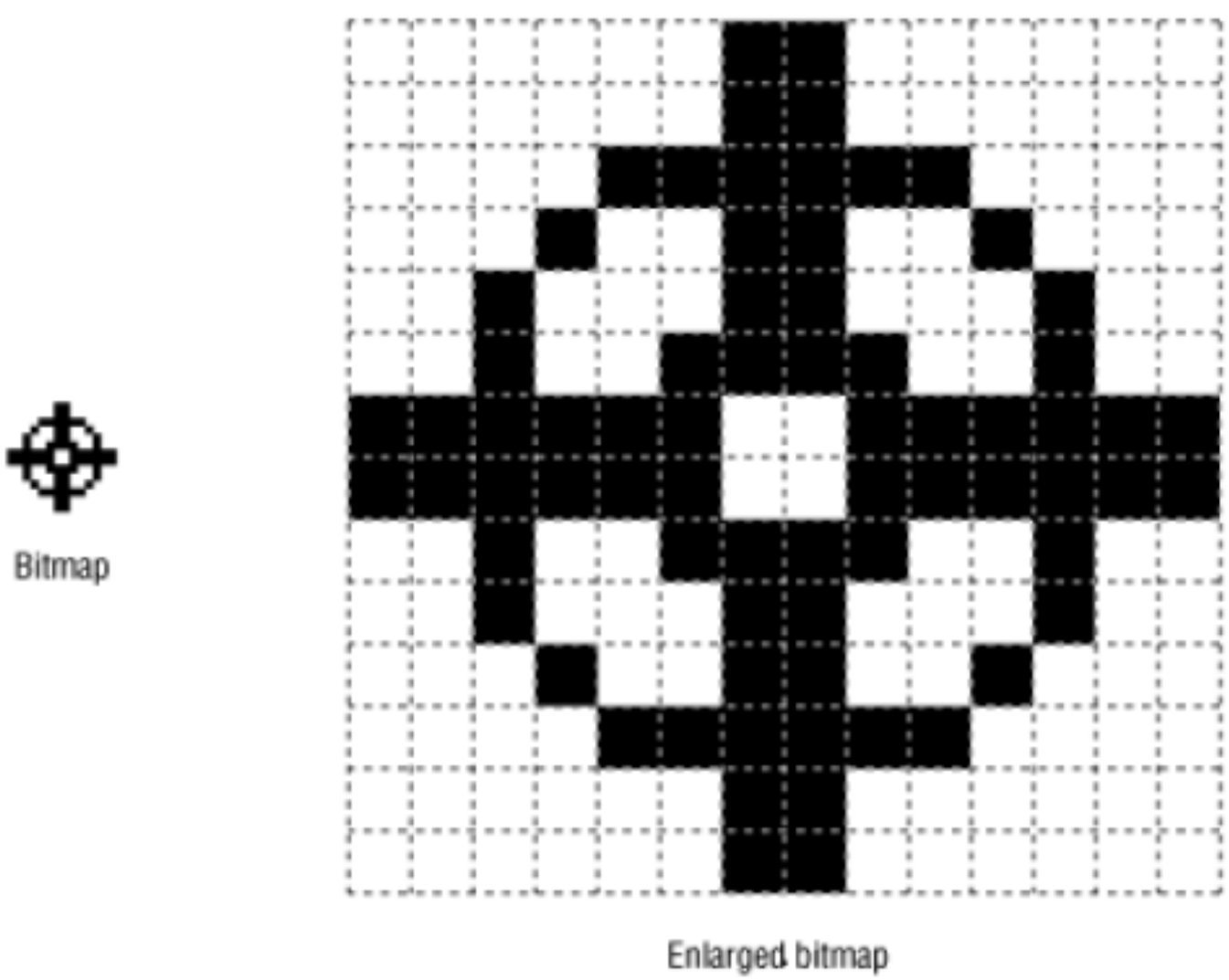
- Example: bit/high.c

```c
const int HIGH_SPEED = (1<<3);
const int DIRECT_CONNECT = (1<<5);

char flags = 0;
```

```c
    flags |= HIGH_SPEED;
    flags |= DIRECT_CONNECT;

    if ((flags & HIGH_SPEED) != 0)
        printf("High speed is set\n");
    else
        printf("High speed is not set\n");

    if ((flags & DIRECT_CONNECT) != 0)
        printf("Direct connect is set\n");
    else
        printf("Direct connect is not set\n");
```

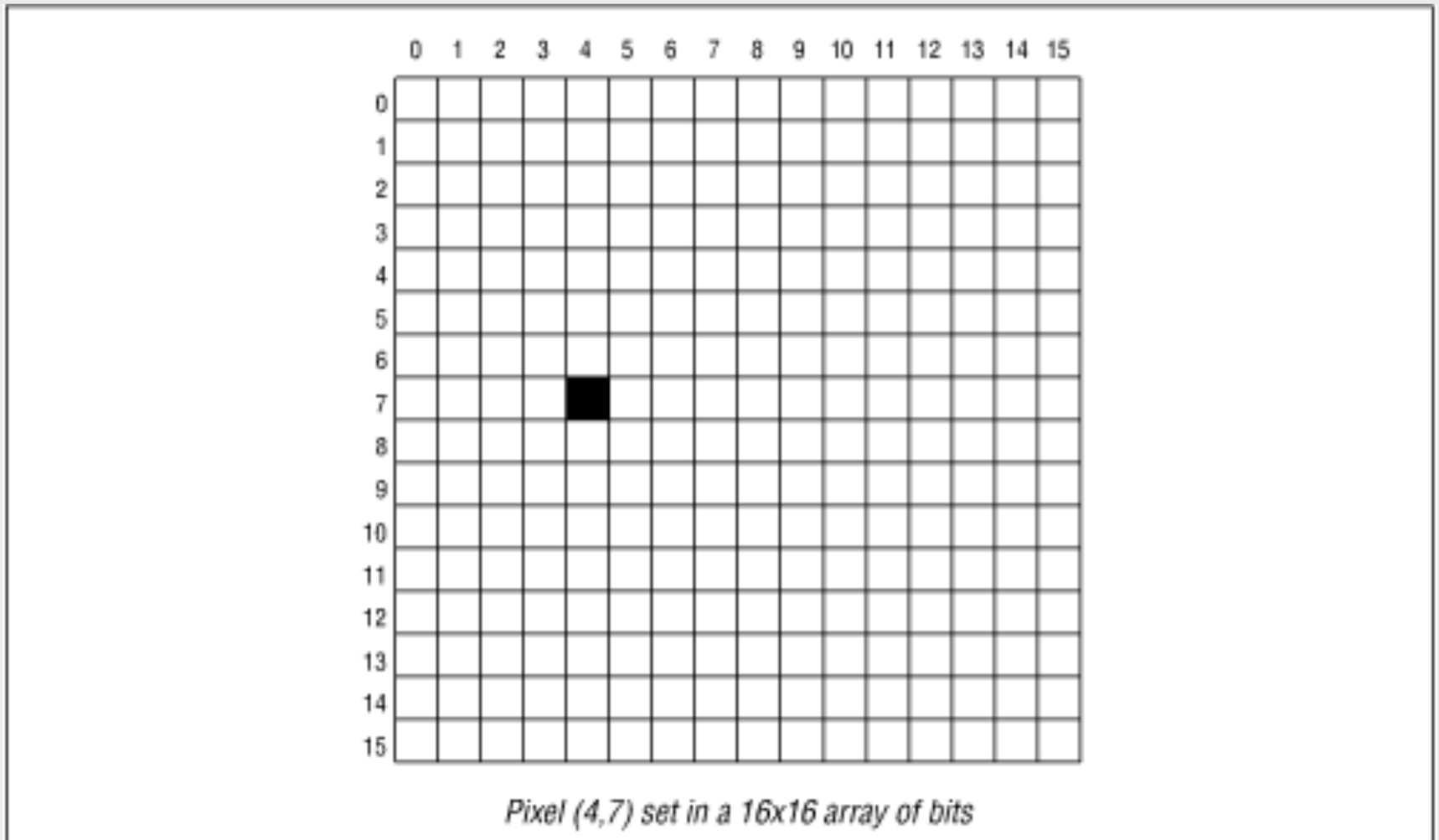
```c
    flags &= ~HIGH_SPEED;
    flags &= ~DIRECT_CONNECT;

    if ((flags & HIGH_SPEED) != 0)
        printf("High speed is set\n");
    else
        printf("High speed is not set\n");

    if ((flags & DIRECT_CONNECT) != 0)
        printf("Direct connect is set\n");
    else
        printf("Direct connect is not set\n");
```

Bitmapped Graphics



Bitmapped Graphics

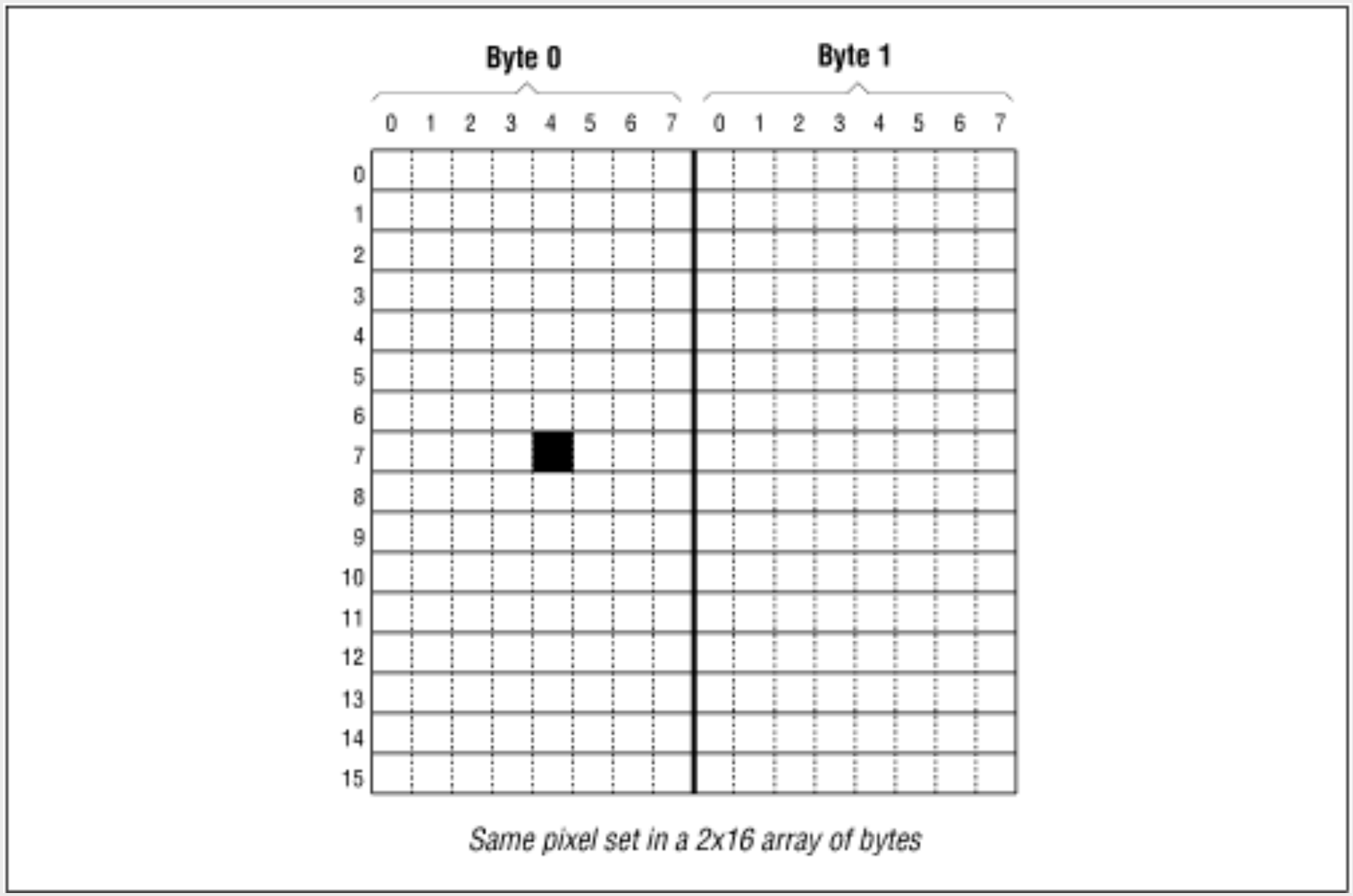
- Suppose we have a small graphic device -- a 16-by-16-pixel display. We want to set the bit at 4,7.



Pixel (4,7) set in a 16x16 array of bits

Bitmapped Graphics

- 16-by-16 array of bits is a 2-by-16 array of bytes



Same pixel set in a 2x16 array of bytes

Bitmapped Graphics

- Use the following algorithm to set a bit

```
byte_y = y;
byte_x = x / 8;
bit_index = x % 8;
bit = 0x80 >> bit_index;
graphics[byte_x][byte_y] |= bit;
```

- Use a single macro to accomplish it

```
#define set_bit(x, y) graphics[(x)/8][y] |= (0x80 >> ((x)%8))
```

  - to set the pixel at bit number 4, 7
  - set the fourth bit of byte 0, 7
  - **`bit_array[0][7] |= (0x80 >> 4);`**

Bitmapped Graphics

- Example: bit/graph.c

Bitmapped Graphics

- Example: bit/graph.c

```
3  #define X_SIZE 32  /* size of array in the X direction */
4  #define Y_SIZE 32  /* size of the array in Y direction */
5  /*
6   * We use X_SIZE/8 since we pack 8 bits per byte
7   */
8  char graphics[X_SIZE / 8][Y_SIZE];   /* the graphics data */
9
10 #define SET_BIT(x,y) graphics[(x)/8][y] |= (0x80 >>((x)%8))
```

Bitmapped Graphics

- Example: bit/graph.c

```
3  #define X_SIZE 32  /* size of array in the X direction */
4  #define Y_SIZE 32  /* size of the array in Y direction */
5  /*
6   * We use X_SIZE/8 since we pack 8 bits per byte
7   */
8  char graphics[X_SIZE / 8][Y_SIZE];   /* the graphics data */
9
10 #define SET_BIT(x,y) graphics[(x)/8][y] |= (0x80 >>((x)%8))
```

```
12 int main()
13 {
14     int   loc;          /* current location we are setting */
15     void  print_graphics(void); /* print the data */
16
17     for (loc = 0; loc < X_SIZE; ++loc)
18         SET_BIT(loc, loc);
19
20     print_graphics();
21     return (0);
22 }
```

Bitmapped Graphics

- Example: bit/graph.c

```
3  #define X_SIZE 32  /* size of array in the X direction */
4  #define Y_SIZE 32  /* size of the array in Y direction */
5  /*
6   * We use X_SIZE/8 since we pack 8 bits per byte
7   */
8  char graphics[X_SIZE / 8][Y_SIZE];   /* the graphics data */
9
10 #define SET_BIT(x,y) graphics[(x)/8][y] |= (0x80 >>((x)%8))
```

```
12 int main()
13 {
14     int   loc;          /* current location we are setting */
15     void  print_graphics(void); /* print the data */
16
17     for (loc = 0; loc < X_SIZE; ++loc)
18         SET_BIT(loc, loc);
19
20     print_graphics();
21     return (0);
22 }
```

Bitmapped Graphics

- Example: bit/graph.c

```
3  #define X_SIZE 32  /* size of array in the X direction */
4  #define Y_SIZE 32  /* size of the array in Y direction */
5  /*
6   * We use X_SIZE/8 since we pack 8 bits per byte
7   */
8  char graphics[X_SIZE / 8][Y_SIZE];   /* the graphics data */
9
10 #define SET_BIT(x,y) graphics[(x)/8][y] |= (0x80 >>((x)%8))
```

```
12 int main()
13 {
14     int   loc;          /* current location we are setting */
15     void  print_graphics(void); /* print the data */
16
17     for (loc = 0; loc < X_SIZE; ++loc)
18         SET_BIT(loc, loc);
19
20     print_graphics();
21     return (0);
22 }
```

Bitmapped Graphics

- Example: bit/graph.c

Bitmapped Graphics

- Example: bit/graph.c

```
24 void print_graphics(void)
25 {
26     int x;              /* current x BYTE */
27     int y;              /* current y location */
28     unsigned int bit;   /* bit we are testing in the current byte */
29
30     for (y = 0; y < Y_SIZE; ++y) {
31         /* Loop for each byte in the array */
32         for (x = 0; x < X_SIZE / 8; ++x) {
33             /* Handle each bit */
34             for (bit = 0x80; bit > 0; bit = (bit >> 1)) {
35                 if ((graphics[x][y] & bit) != 0)
36                     printf("X");
37                 else
38                     printf(".");
39             }
40         }
41         printf("\n");
42     }
43 }
```

Bitmapped Graphics

- Example: bit/graph.c

```
24 void print_graphics(void)
25 {
26     int x;              /* current x BYTE */
27     int y;              /* current y location */
28     unsigned int bit;   /* bit we are testing in the current byte */
29
30     for (y = 0; y < Y_SIZE; ++y) {
31         /* Loop for each byte in the array */
32         for (x = 0; x < X_SIZE / 8; ++x) {
33             /* Handle each bit */
34             for (bit = 0x80; bit > 0; bit = (bit >> 1)) {
35                 if ((graphics[x][y] & bit) != 0)
36                     printf("X");
37                 else
38                     printf(".");
39             }
40         }
41         printf("\n");
42     }
43 }
```