Arrays

Objectives

At the end of the meeting, students should be able to:

- explain the importance of arrays
- solve simple problems using arrays

What if…

“I need to store 1000 numbers”

Arrays

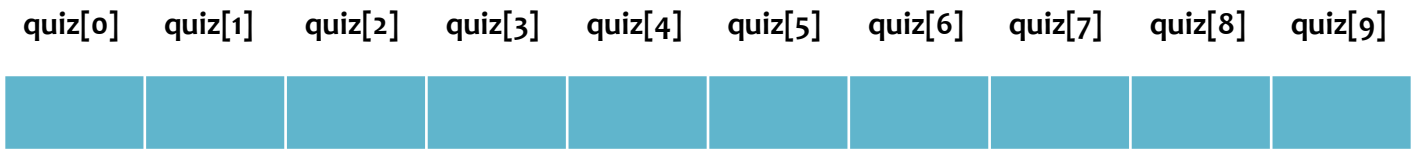
- An array is a list of adjacent memory locations whose components have a **uniform type**

- **Declaration:**

```
int quiz[10];
float long_exam[4];
```

Access of Array Components

```
int quiz[10];
```



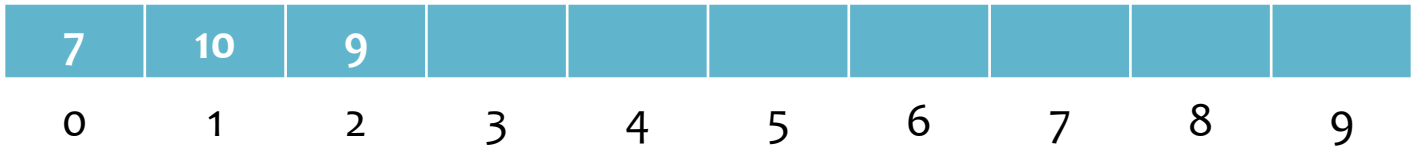
possible physical addresses

```
float long_exam[4];
```

Note: Array indices in C always start with 0

Example: Access of Array Components

int quiz[10];

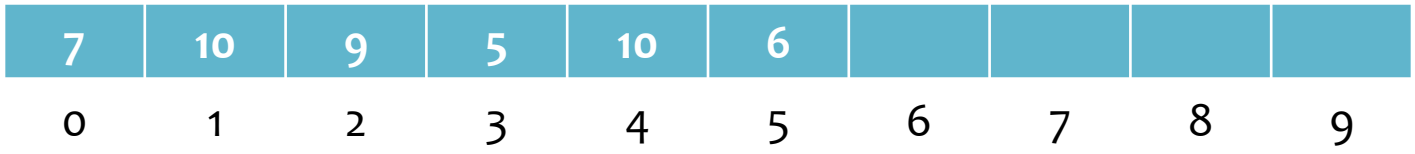


Sample code for filling the values of array ‘quiz’:

```
quiz[0] = 7;
quiz[1] = 10;
quiz[2] = 9;
```

Example: Access of Array Components

int quiz[10];



Sample code for filling the values of array 'quiz':

```
scanf("%d %d %d", &quiz[3], &quiz[4], &quiz[5]);
   // assume user entered 5, 10, 6
```

Example: Access of Array Components

int quiz[10];

7	10	9	5	10	6	10	7	8	9
0	1	2	3	4	5	6	7	8	9

Sample code for filling the values of array 'quiz':

```
int x = 8;
quiz[3*2] = 10;
quiz[7] = quiz[0];
quiz[x] = x;
scanf("%d", &quiz[x+1]);
   //assume user entered 9
```

Example: Access of Array Components

For larger arrays, a more practical way is by using a loop

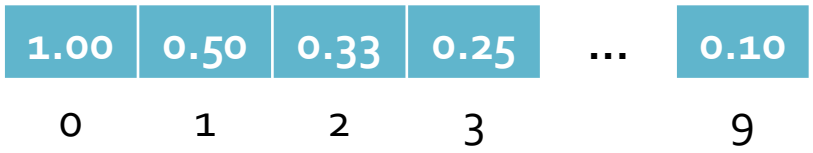
```
int a[500], j;

for ( j=0; j<500; j++ ) {    // input: a[0] ... a[499]
      printf("Enter item %d: ", j);
      scanf("%d", &a[j]);
}

for( j=0; j<500; j++ ){
      printf("Number %d = %d", j, a[j]);
}
```

Initializing the Array(with zeros)

```
// method #1
    int x[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };

// method #2
    int x[8];
    x[0] = x[1] = x[2] = x[3] = x[4] = x[5] = x[6] = x[7] = 0;

// method #3
    int x[8], j;
    for ( j=0; j<8; j++ ) {
          x[j] = 0;
    }
```

Array indices can be any integer expression within the valid range

```
float r[10]; // reciprocals 1, 1/2, 1/3, 1/4, ... , 1/10
int j;

// initialize and print array
for ( j=0; j<10; j++ ) {
    r[j] = 1.0 / (j+1);
    printf("%.2f ", r[j] );
}

// print array values in reverse order
for ( j=0; j<10; j++ ) {
    printf("%.2f ", r[9-j] );
}
```



When to Use Arrays?

- When we have large data sets
  - Game data (hours spent playing dota, and scores in the first exam of 100 students)
    - float hours[100], score[100];
  - Diet data (number of siomai consumed per week, and weight of 100 UPLB students)
    - float numSiomai[100], students[100];
- When the data are entered, various statistical procedures can be programmed on the data sets
  - averages, standard deviations, regression and correlation analysis

Example

- Write a program to compute the **average** (arithmetic mean), **min** and **max**

Solution

```
int j;
float sum, min, max;
sum = min = max = x[0];       //assume x has 10 elements
for ( j=1; j<10; j++ ) {
    sum = sum + x[j];
    if (x[j] < min) min = x[j];
    else if (x[j] > max) max = x[j];
}
printf ("avg is %f, min is %f, max is %f\n", sum/n, min, max);
```

Strings

- A string is an array of characters.
- **Declaration**
  char name[10];

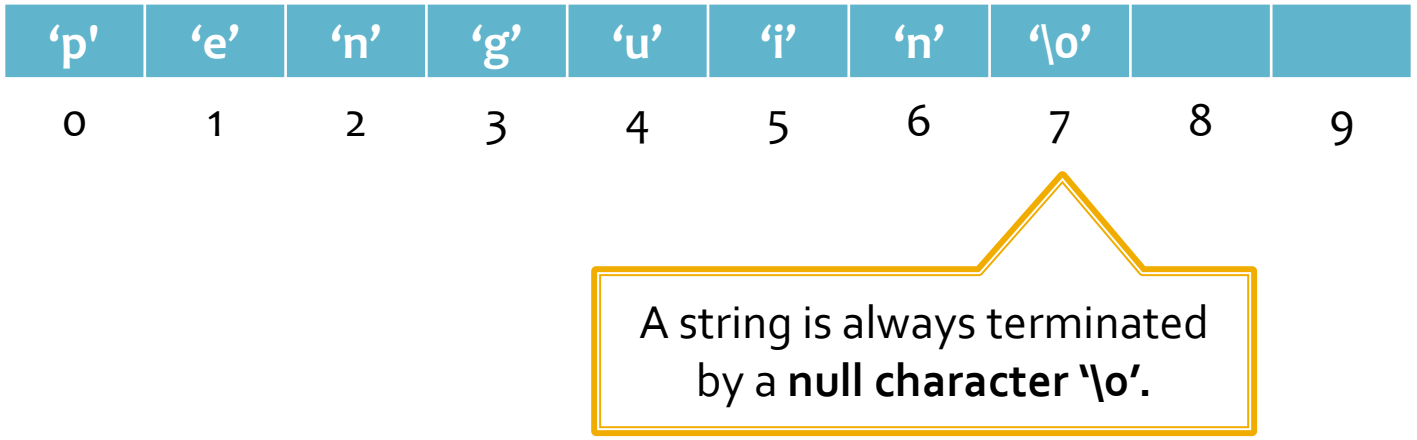
Strings

- **Assignment**

  strcpy(name, “penguin”);

  scanf(“%s”, name);

  // assuming user entered “penguin”



String Functions

- strcpy(char [] destination , char [] source);
  //copies the source string to the destination

- int strlen(char [] s);
  //returns the length of the string

- int strcmp(char *s, char *t) or
- int strncmp(char *s, char *t, int n)
  // returns 0, -1, 1, if s is identical to, before, or after t
  // strncmp checks only the first n characters

Exercises

- Classic word games provide a rich set of exercises for array processing
  - Reverse a string,

    e.g., “hello” --> “olleh”
  - Check if a string is a palindrome,

    e.g., “pop”, “radar”, ...
  - Check if 2 strings are anagrams,

    e.g., (“stop”, “post”), (“algorithm”, “logarithm”), ...
  - Test for palindromes/anagrams where strings are case-insensitive and can have blanks/punctuation,

    e.g., “Campus motto: Bottoms up, Mac!”

Working on an array of strings

- A string is an array of chars, so how do we represent an array of strings?

```
main(){
        char *a[4] = {“Alan”, “Alex”, “Ana”, “Alice”};
        int j;
        for ( j=0; j<4; j++ ) {
                // print only those that start with “Al”
                if ( strncmp( a[j], ”Al”, 2) == 0 )
                        printf( “%s has %d chars\n”, a[j], strlen(a[j]) );
        }
}
```

Objectives

At the end of the meeting, students should be able to

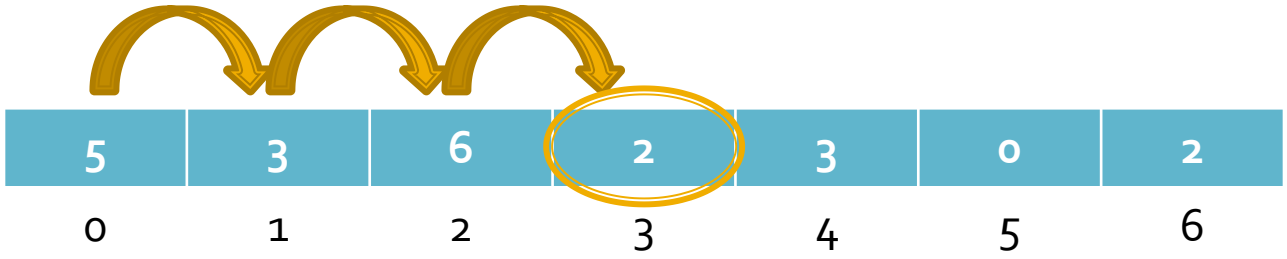
- identify and explain some searching and sorting algorithms
- enumerate some applications of 2D arrays and beyond

Searching for an item in an array

- Searching is a fundamental problem in computer science, e.g., the Google family of search tools
- We study a simpler version:
  - **Input:** an integer n, an array a[] of n items, and a target item x
  - **Output:** the **lowest index p** where x first occurs in a[]; if x is absent, return the special value -1 indicating absence
  - **Example:** searching for **x = 2** in a[7] = { 5,3,6,**2**,3,0,2 } should give the index 3

Linear Search

- The simplest algorithm for this problem is called sequential search – check all positions in the array



Linear Search

```
// search for a target x in a[0..n-1] and return its position
// (lowest index), or return -1 if x is absent
int search( int n, int a[], int x ){
    int j;
    for ( j=0; j<n; j++ ) {
        if ( a[j] == x ) return j; // found x, return immediately
    }
    return -1; // not found, x is absent in the array
}
```

Algorithm analysis:
How good/bad is sequential search?

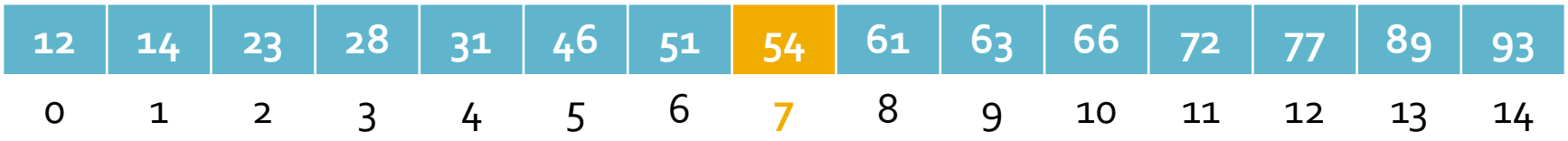
- If we are lucky, the target x is found quickly in a[0..n-1] such as when its index is close to 0
- But in the worst case, it takes n iterations to complete the algorithm (when x is in the last position or when x is absent)
- This can be very slow if n is large – imagine searching a telephone directory for the name "Zorro" using sequential search!
- Sometimes we have no choice but to use sequential search, such as when the array is in random order (e.g., finding MIN, MAX)

Better ways to search

- Dictionaries (as well as telephone directories, book indexes, etc) are arranged in alphabetical order for a good reason – they allow quick searches
- Some algorithms for quick searches
  - Binary search – start the search at the middle index and discard either the left half or the right half, repeat …
  - Interpolation search – start the search at a good estimate of where the target x might lie (start the search near the end if looking for "Zorro", etc)
- These algorithms assume that the array is arranged inorder (sorted), else they won't work

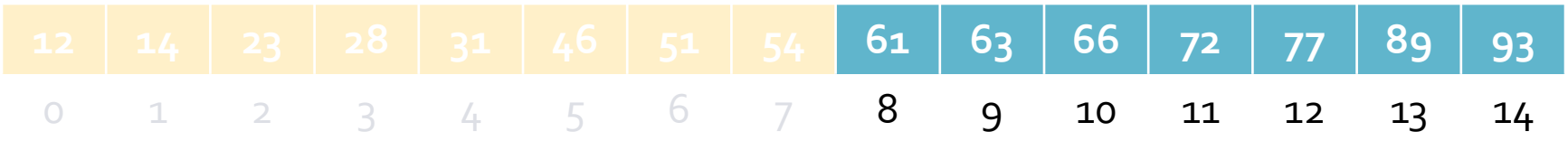
Binary Search Example

Search for x = 77 in an array



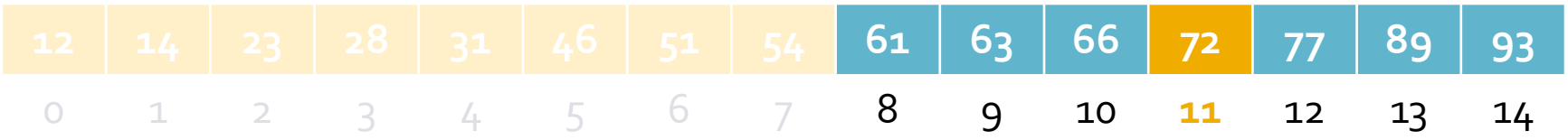
Binary Search Example

Search for x = 77 in an array



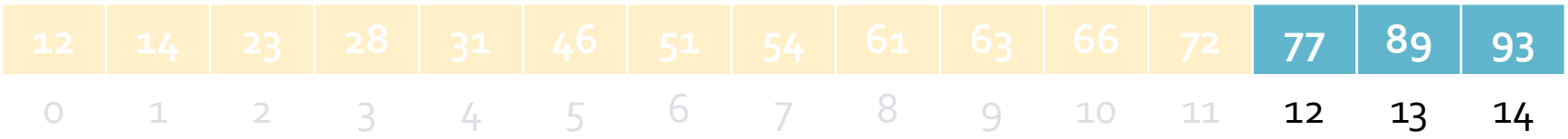
Binary Search Example

Search for x = 77 in an array



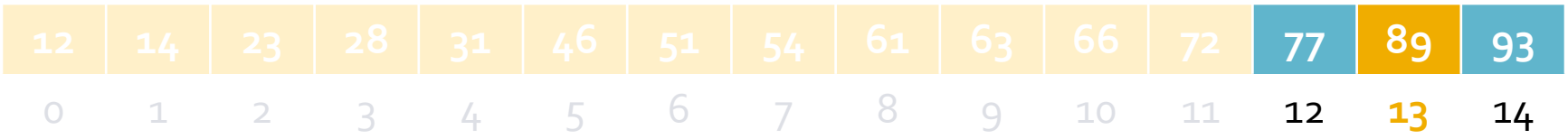
Binary Search Example

Search for x = 77 in an array



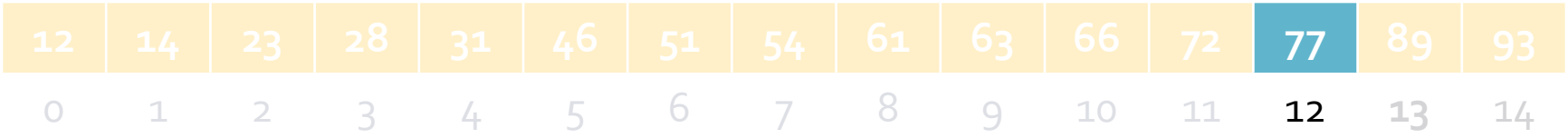
Binary Search Example

Search for x = 77 in an array



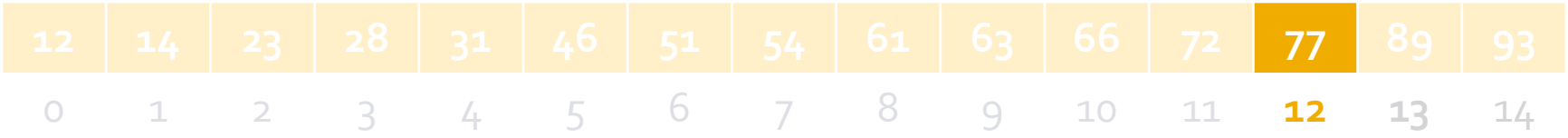
Binary Search Example

Search for x = 77 in an array



Binary Search Example

Search for x = 77 in an array



Binary Search Code

```
// search for a target x in sorted a[0..n-1] and return its position
int binarysearch( int n, int a[], int x ){
    int low = 0, mid, high = n-1; // indices in the array
    while ( low <= high ) {
        mid = (low + high) / 2;
        if ( a[mid] == x ) return mid; // found x, return immediately
        else if ( a[mid] < x ) low = mid+1; // ignore left half
        else high = mid-1; // ignore right half
    }
    return -1; // not found, x is absent in the array
}
```

Algorithm analysis:
How good/bad is binary search?

- How many iterations are necessary when searching for an item in an array with n items?
  - An array of size n=1 requires 1 iteration
  - An array of size n=3 requires at most 2 iterations
  - An array of size n=7 requires at most 3 iterations
  - An array of size n=15 requires at most 4 iterations
  - An array of size $n=2^k-1$ requires at most k iterations
  - An array of size n requires only about $\log_2 n$ iterations
- **Doubling the array size only adds 1 more iteration**
- Faster than sequential search, but we have to sort the array first – an excellent investment if several searches have to be made

Sorting

- Sorting is one of the most popular problems in computer science
- Dozens of sorting algorithms exist, some simple one are generally slow for large arrays, more complex ones sort faster

Sorting

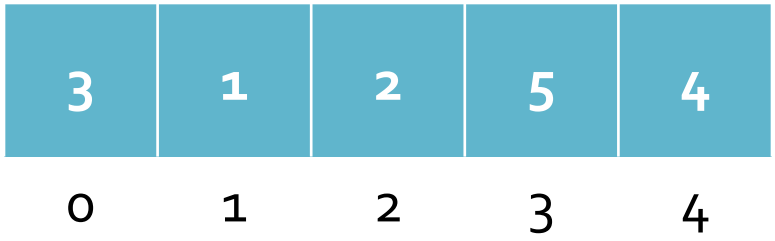
- We develop one of the simplest sorting algorithms called **selection sort**
- Standard C library provides a faster predefined sorting function called **qsort();** Linux also provides a command called **sort** for sorting lines in a text file; many applications such as spreadsheets also have sorting utilities

Selection Sort Algorithm

- Main idea is to find the largest, then the second largest, then the third largest, etc

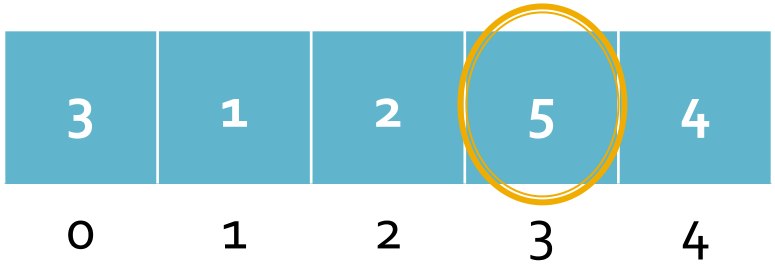
Selection Sort Algorithm

- Sort the array in ascending order



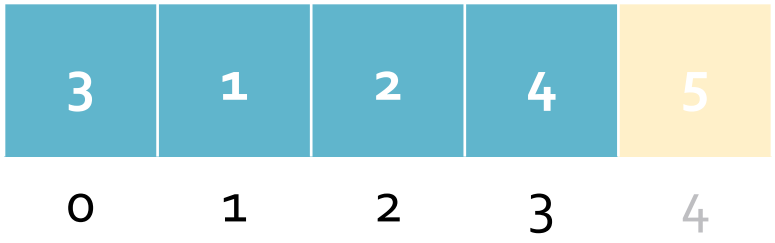
Selection Sort Algorithm

- Sort the array in ascending order



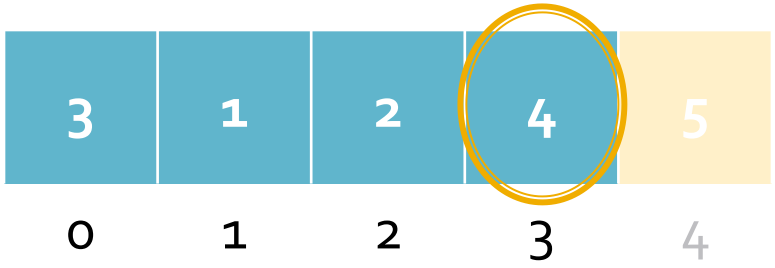
Selection Sort Algorithm

- Sort the array in ascending order



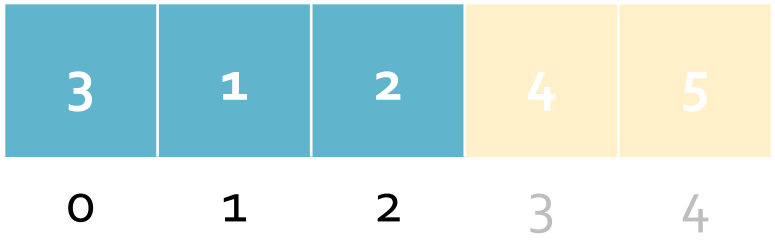
Selection Sort Algorithm

- Sort the array in ascending order



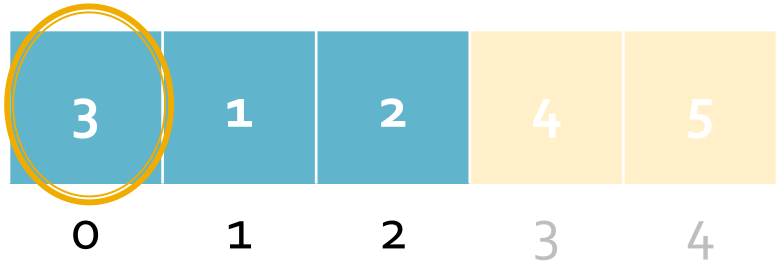
Selection Sort Algorithm

- Sort the array in ascending order



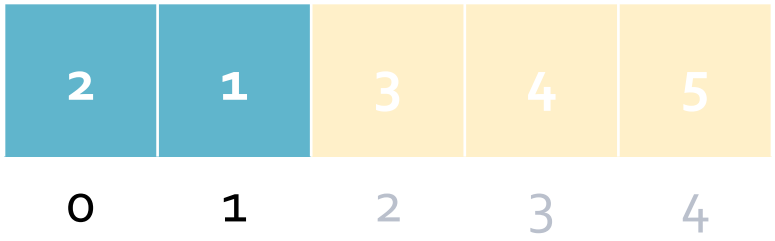
Selection Sort Algorithm

- Sort the array in ascending order



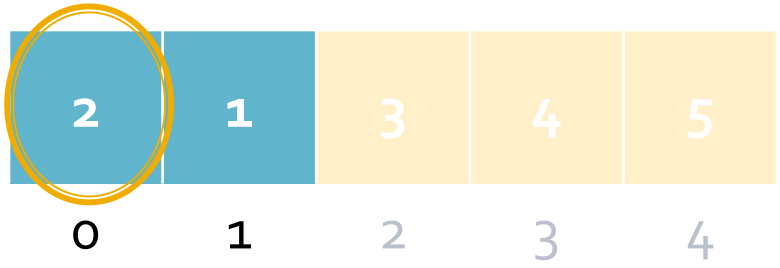
Selection Sort Algorithm

- Sort the array in ascending order



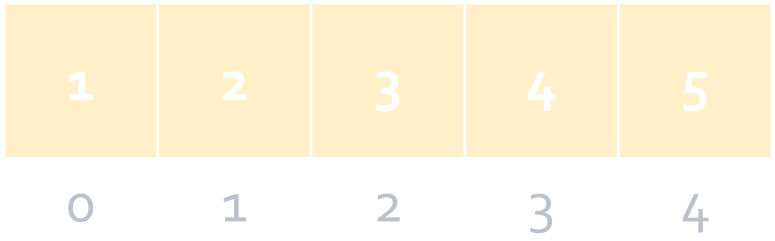
Selection Sort Algorithm

- Sort the array in ascending order



Selection Sort Algorithm

- Sort the array in ascending order



Selection Sort Code

```
sort( int n, int a[] ){
    int j, k, maxpos;
    for ( j=n-1; j>0; j-- ) {
        // find max in a[0..j] using sequential search
        maxpos = 0;
        for ( k=1; k<=j; k++ ){
            if (a[k] > a[maxpos] ) maxpos = k;
        }
        // now swap the max in a[0..j] with a[j]
        swap( &a[maxpos], &a[j] );
    }
}
```

Exercises

- Implement fully the sequential sort algorithm, including the swap() function, and a main program that generates a random array of integers, and prints both the unsorted and the sorted versions
- Modify the code so it will sort in descending order
- Modify the code so we can sort arrays of chars, floats, and strings – use strcmp() for comparing strings in the if statement

int strcmp(char x[], char y[])

- strcmp() is a predefined function for comparing two string x and y using alphabetical ordering based on the ASCII codes
- Returns:
  - <0, if x is before y
  - 0, if x is the same string as y
  - >0, if x is after y

int strcmp(char x[], char y[])

- Examples:

  strcmp(“hello”,”hello”) returns 0

  strcmp(“apple”, “zebra”) returns <0

  strcmp(“dog”, “cat”) returns >0

2D arrays and beyond

- Used for tables, matrices, and other multidimensional structures with a common data type
- Example: crossword and sudoku puzzles, tictactoe, chess, checkers, and many other puzzles and board games

char tictactoe[3][3];

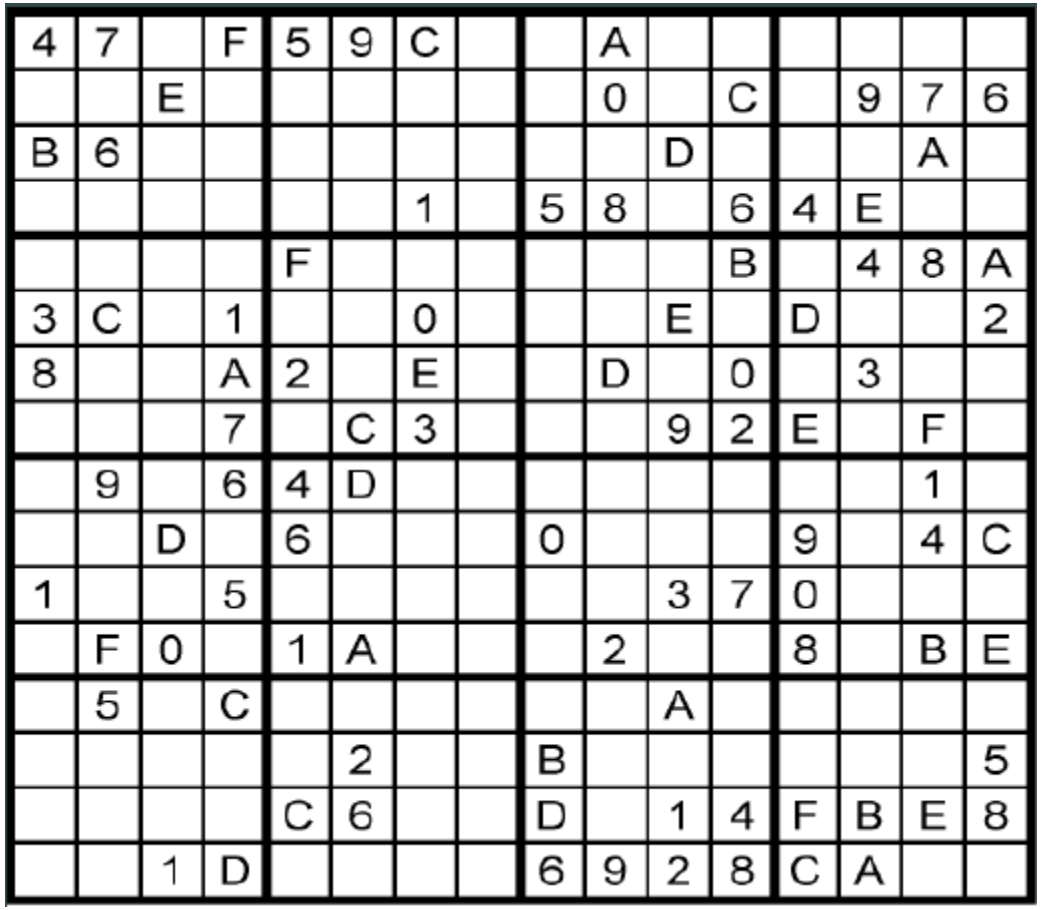
	0	1	2
0	x	o	-
1	-	x	-
2	o	x	o

Other 2D arrays

```
char sudoku[9][9];
char chess[8][8];
```

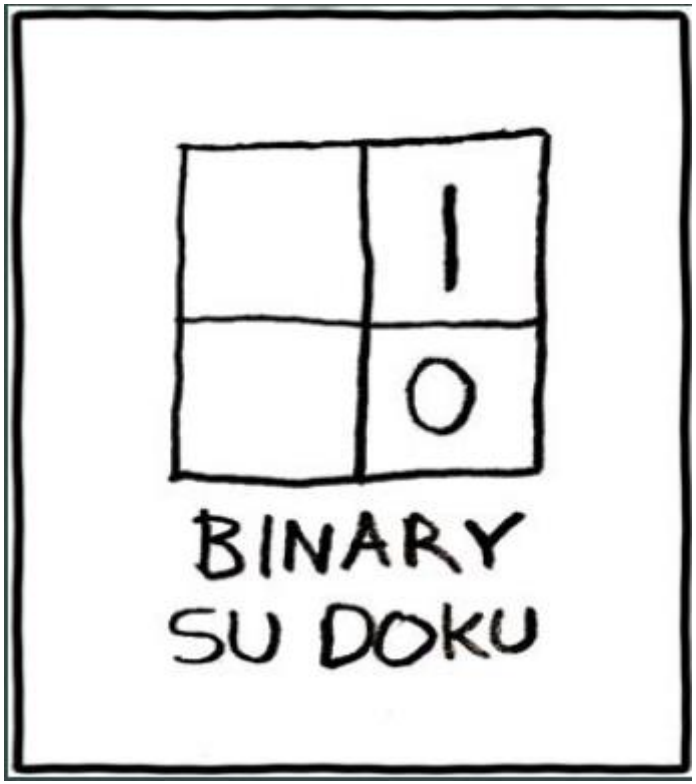
8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5

A hex Sudoku for those who need a challenge



4	7		F	5	9	C			A						
		E							0		C		9	7	6
B	6									D				A	
						1		5	8		6	4	E		
				F							B		4	8	A
3	C		1			0				E		D			2
8			A	2		E			D		0		3		
			7		C	3				9	2	E		F	
	9		6	4	D									1	
		D		6				0				9		4	C
1			5							3	7	0			
	F	0		1	A				2			8		B	E
	5		C							A					
					2			B							5
				C	6			D		1	4	F	B	E	8
		1	D					6	9	2	8	C	A		

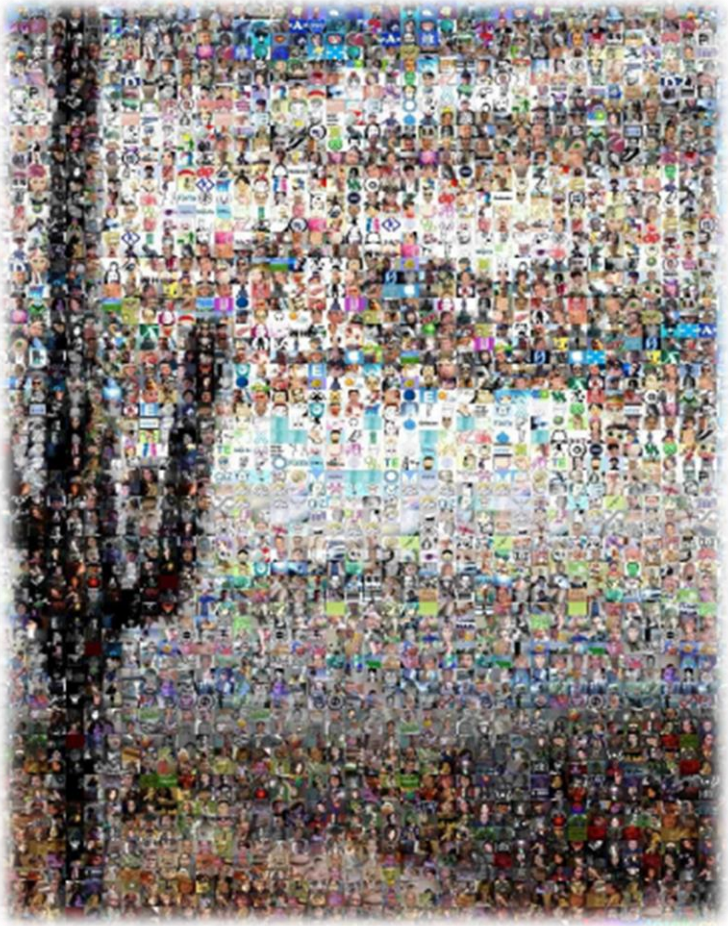
and a Sudoku for your pet cat..☺



Matrices are often used for financial applications…

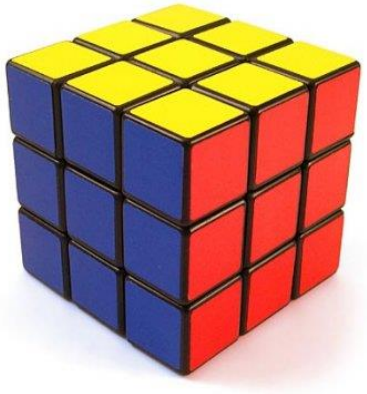
Sizes	Blue	Black	Brown	Gray	Tan
38	8.00	79.00	40.00	26.00	346.00
38-T	42.00	59.00	42.00	48.00	100.00
40	92.00	25.00	66.00	75.00	90.00
40-T	65.00	25.00	78.00	35.00	273.00
42	15.00	13.00	11.00	64.00	371.00
42-T	5.00	22.00	6.00	2.00	425.00
44	6.00	61.00	329.00	202.00	374.00
44-T	20.00	19.00	127.00	769.00	476.00

...and all types of images



A matrix of pixels...

Sometimes you may need 3D



A data structure for a word hunt

Easter Word Hunt

E G G S H U N T
N D L F S L L C
E Y A R F R I H
S E M B I K L I
T G B U N N Y C
I C H I D E F K
E A S T E R L W
D J G R A S S Q

BUNNY
CHICK
DYE
EASTER
EGGS
FIND
GRASS
HIDE
HUNT
LAMB
LILY
NEST


```
char grid[8][8];
char *words[12] = {
   "BUNNY",
   "CHICK",
   "DYE",
   "EASTER",
   ...
}
```