Regular Data Structures

1, 2, 3, 4, N-D Arrays

Data Structures

- Store and organize data on computers
- Facilitate data processing
  - Fast retrieval of data and of related data
- Similar to furniture with shelves and drawers
  - Quick access
  - Quick selection
- Data structure is designed according to data and data processing characteristics
  - A big part of the data processing solution

Regular data structures: *arrays*

- Identical data elements tightly packed
- Direct access through indexing
  - Index must be within array bounds
- Structure is implicit
  - Neighbors are found through indexing
  - No need to waste storage space for structure

Regular data structures: *arrays*

- 1-D array
  - A row

Example: houses on street

- The houses form a 1-D array
- House number serves as index
- You can refer to a house directly using its number
- An urban modeling SW application could store the houses on a street in a 1-D array

Example: today's hourly temperatures at given location

- There are 24 hours
- 1 temperature reading for every hour

Example: today's hourly temperatures at given location

- There are 24 hours
- 1 temperature reading for every hour
- A 1-D array with 24 elements
- Each element is a number

55	54	53	50	49	49	55	60	65	68	70	72	75	77	80	83	85	85	82	79	77	70	60	57

Example: hourly temperatures in West Lafayette on given day

- There are 24 hours
- 1 temperature reading for every hour
- A 1-D array with 24 elements
- Each element is a number
- We usually show indices in diagrams, but indices are NOT stored in the array

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
55	54	53	50	49	49	55	60	65	68	70	72	75	77	80	83	85	85	82	79	77	70	60	57

Example: hourly temperatures in West Lafayette on given day

- Let's call the array *T*, from temperature
- To find the temperature at 8am
  - Index the 9th element of the array T[8] (i.e. index 8 in 0-based indexing)
  - Read the value at T[8] to obtain 65

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
55	54	53	50	49	49	55	60	65	68	70	72	75	77	80	83	85	85	82	79	77	70	60	57

Example: hourly temperatures in West Lafayette on given day

- Change the temperature value at 7pm to 80
  - Assign 80 to T[19]
  - We will denote assignment w/ equal sign

    T[19] = 80

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
55	54	53	50	49	49	55	60	65	68	70	72	75	77	80	83	85	85	82	79	77	70	60	57

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
55	54	53	50	49	49	55	60	65	68	70	72	75	77	80	83	85	85	82	80	77	70	60	57

Example: text in a paragraph

- Consider the following paragraph
  - "In the midday breeze, where the willow is swaying, flowers are blooming."
- Stored in 1-D array of 72 bytes (1 byte / char)

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
I	n		t	h	e		m	i	d	d	a	y		b	r	e	e	z	e	,		w	h
24	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*
e	r	e		t	h	e		w	i	l	l	o	w		i	s		s	w	a	y	i	n
48	*49*	*50*	*51*	*52*	*53*	*54*	*55*	*56*	*57*	*58*	*59*	*60*	*61*	*62*	*63*	*64*	*65*	*66*	*67*	*68*	*69*	*70*	*71*
g	,		f	l	o	w	e	r	s		a	r	e		b	l	o	o	m	i	n	g	.

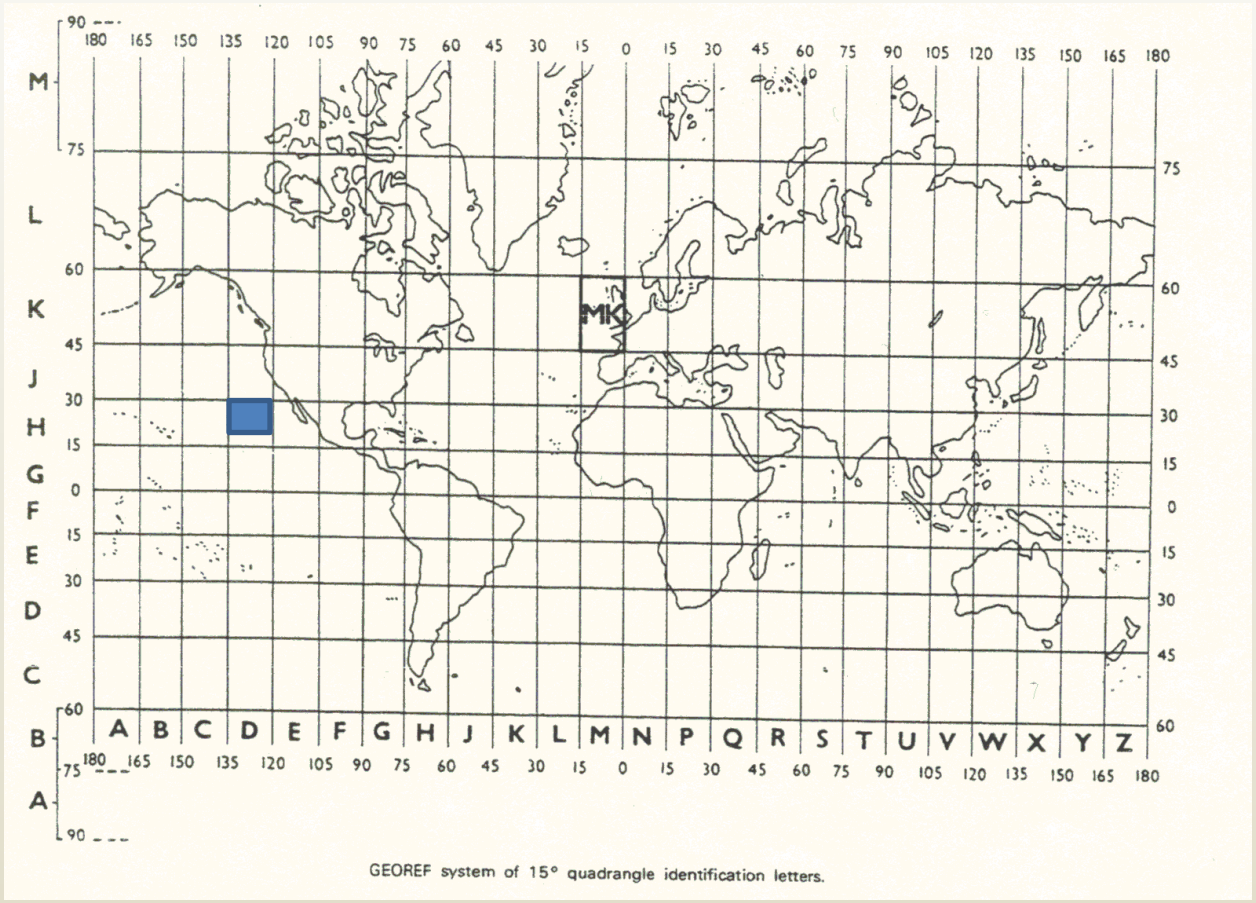
Regular data structures: *arrays*

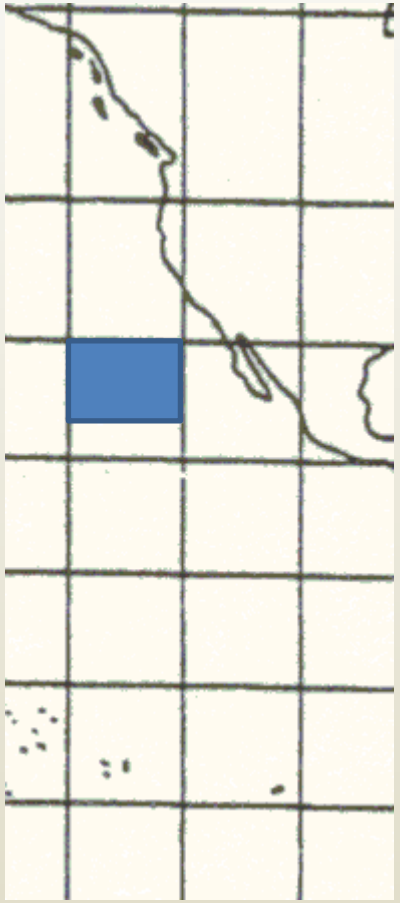
- 1-D array
  - A row
- 2-D array
  - Rows and columns, rows of rows

2-D array example: an ocean surface water temperature map

- Data structure to store
  - Water temperatures in the Pacific
  - Between 135^{o} and 120^{o} long W and between 30^{o} and 20^{o} lat N
  - For every 1^{o} long x 1^{o} lat ocean patch
- Solution
  - Regularly spaced data
  - Two dimensions: latitude and longitude
  - 2-D array would work great
  - Row: corresponds to same latitude
  - Column: corresponds to same longitude

2-D array example: an ocean surface water temperature map





2-D array example: an ocean surface water temperature map

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
0	75	75	75	75	75	76	75	75	75	75	75	75	75	75	75
1	75	75	75	75	75	75	75	75	75	75	75	75	75	75	75
2	75	75	75	75	75	75	75	75	75	75	76	76	75	75	75
3	75	75	75	75	75	75	75	75	76	75	76	76	76	75	75
4	76	76	76	75	76	76	75	76	75	76	77	76	76	75	75
5	76	76	76	76	76	77	75	76	75	76	77	77	78	75	75
6	76	76	76	77	76	78	75	76	75	76	79	76	79	78	77
7	78	79	82	83	82	80	82	80	80	82	80	82	80	79	80
8	80	81	80	80	82	81	82	82	83	82	81	82	82	81	83
9	80	81	82	82	82	82	82	82	83	82	83	82	83	83	83

2-D array example: an ocean surface water temperature map

- Let's call the 2-D array M, from map
- Find ocean temperature at 128° W and 27° N
  - Map covers 135° to 120° W and 30° to 20° N
  - Row index: 30-27=3
  - Col. index: 135-128=7
  - M[3][7] is 76
  - "Row major" order
  - M[3] is a 1-D array

  storing the temperatures at 27° N

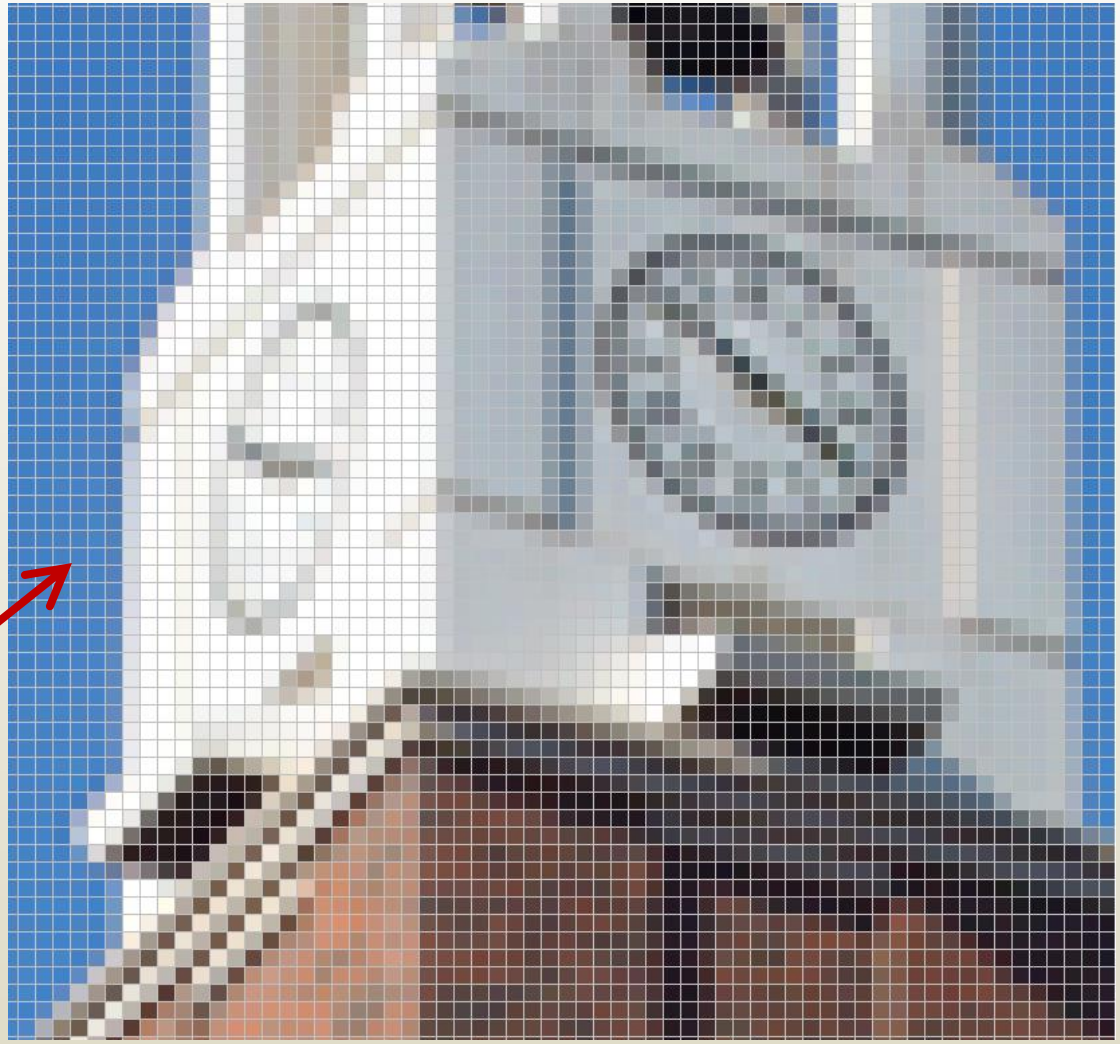
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
0	75	75	75	75	75	76	75	75	75
1	75	75	75	75	75	75	75	75	75
2	75	75	75	75	75	75	75	75	75
3	75	75	75	75	75	75	75	76	75
4	76	76	76	75	76	76	75	76	75

2-D array example: a digital image

- An image is a 2-D array of pixels
  - Color is constant within pixel
- A pixel is a triplet of numbers
  - A red, a green, and a blue intensity value (R, G, B)
  - Red, green, and blue are called channels
  - Usually 8 bit or 1 byte per channel
    - White (255, 255, 255), or, in hex (FF, FF, FF)
    - Red (255, 0, 0), or (FF, 0, 0)
    - Blue (0, 0, 255), or (0, 0, FF)
    - Black (0, 0, 0), or (0, 0, 0)

2-D array example: a digital image

Digital image with 176x288 pixels



Magnified fragment with pixel grid visualization

Pixel row 59, column 125
- Red intensity is 73
- Green intensity is 130
- Blue intensity is 198
I image 2-D array
I[59][125] is (73, 130, 198)

2-D array example: a floor plan

- A dining room
  - Walls (grey)
  - Dining table (brown)
  - 4 chairs (blue)
- 2-D array F
  - 10x10 elements
  - 0 if empty
  - 1 if occupied
  - (colors just for illustr. purposes, not stored in 2-D array)

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

2-D array example: a floor plan

- Robot navigation
  - Move to F[2][2]?
  - *Yes, F[2][2] is 0*
  - Robot at F[8][3], can exit at next step?
  - *Yes, F[9][3] is immediate neighbor is empty, and is exit*

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

2-D array example: a floor plan

- Rearrange room
  - Move chair

  from F[2][4] to F[3][4]
  - Assign 0 to old loc.

  F[2][4] = 0
  - Assign 1 to new loc.

  F[3][4] = 1

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

iClicker question

- How many immediate neighbors does an internal element of a 2-D array have?
  - A[i][j] is an immediate neighbor of B[k][l] if i and k, and j and l differ by at most 1.
  - An internal element of a 2-D array is an element that is not in the first row, the first column, the last row, or the last column.

A. 4 B. 8 C. 9 D. 16

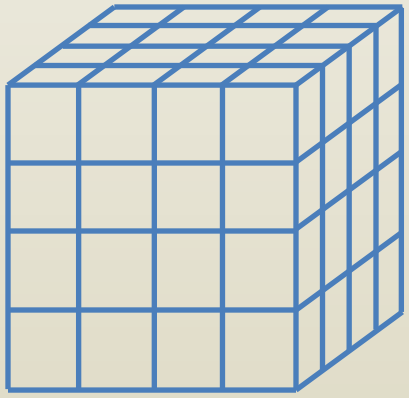
E. None of the above

Regular data structures: *arrays*

- 1-D array
  - A row
- 2-D array
  - Rows and columns, rows of rows
- 3-D array
  - Stack of 2-D arrays

3-D array example: temperatures in volume of water

- Temperatures in every $1m^3$ of $1km^3$ of ocean water
  - 3-D array with 1,000 x 1,000 x 1,000 elements
  - Each element is a number (i.e. a temperature reading)
  - 1 billion numbers



3-D array example: temperatures in volume of water

- Temperatures in every $1m^3$ of $1km^3$ of ocean water
- Let T be the 3-D array
  - T[0][500][500] is the temperature at the surface, at the center of the patch
  - T[0] is a 1,000 x 1,000 2-D array storing the temperatures at the surface
  - T[100] is a 1,000 x 1,000 2-D array storing the temperatures at depth 100m

3-D array example: stack of CT scans

- Engine block scanned to search for defects
  - A stack of 100 CT scans (images)
  - 256x256 resolution each
  - Volume size 50cm x 30cm x 30cm
- Let V be the array
  - 100x256x256 elements
  - Element stores 8 bit opacity value
    - 255: completely opaque (e.g. steel)
    - 0: not opaque (e.g. air)
  - Element corresponds to 50/100cm x 30/256cm x 30/256cm volume

Volume rendering of block engine CT scan

iClicker question

- A 3-D array stores a CT scan in 1GB. The slices are 2mm apart, and each slice has 1mmx1mm pixels. What is the new array size in GB if the CT scan resolution is changed to slices 1mm apart and 0.5mmx0.5mm pix.?

A. The same size, 1GB

B. 2/1*1/0.5*1/0.5*1GB = 8GB

C. 1/2*0.5/1*0.5/1*1GB = 1/8GB

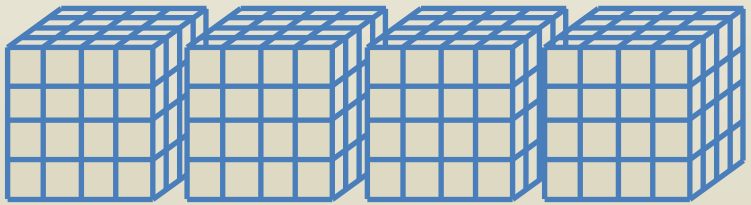
D. 1*0.5*0.5*1GB = 0.25GB

Regular data structures: *arrays*

- 1-D array
  - A row
- 2-D array
  - Rows and columns, rows of rows
- 3-D array
  - Stack of 2-D arrays
- 4-D arrays?

Regular data structures: *arrays*

- 4-D arrays
  - Rows of cuboids
  - D[1] means second cuboid
  - D[1][0] bottom 2-D array in second cuboid
  - D[3][3][1][2] means element in cuboid 3, slice 3, row 1, column 2



Regular data structures: *arrays*

- 4-D arrays alternative visualization
  - A 2x2x2x3 4-D array with 24 elements
  - Let's call the array C
  - C[1][0][1][2] is 7
- N-D arrays are possible

0												*1*											
0						*1*						*0*						*1*					
0			*1*			*0*			*1*			*0*			*1*			*0*			*1*		
0	*1*	*2*	*0*	*1*	*2*	*0*	*1*	*2*	*0*	*1*	*2*	*0*	*1*	*2*	*0*	*1*	*2*	*0*	*1*	*2*	*0*	*1*	*2*
2	3	3	3	2	2	3	4	5	4	2	3	4	5	6	5	3	7	4	3	5	6	4	5

Regular data structures

- Advantages
  - Direct access to elements
  - Implicit structure, no storage wasted for structure
  - Simplicity

Regular data structures

- Disadvantages
  - Data size needs to be known a priori
    - Increasing array size is possible but expensive
  - Inserting / deleting elements is expensive
  - Does not model well irregular, non-uniform data
    - Huge room with mostly empty floor plan?
    - Airline route map?
    - Genealogical tree?

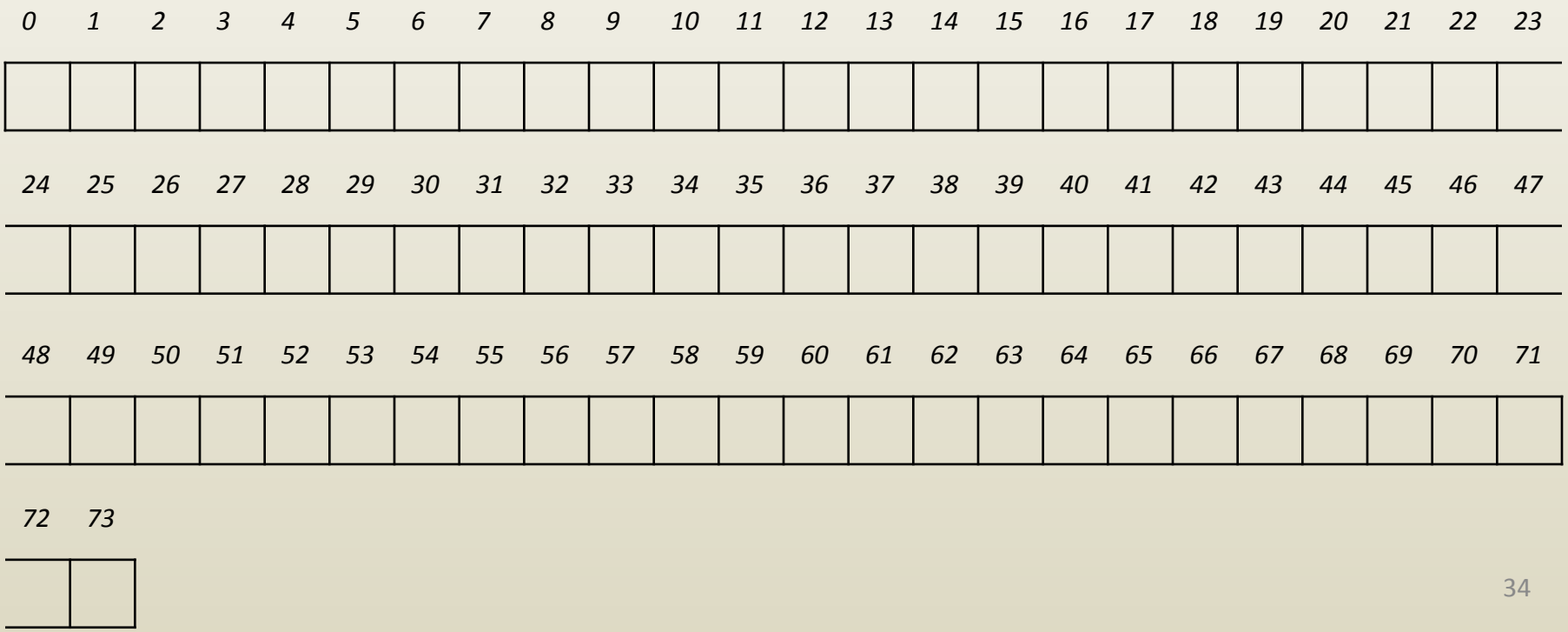
Array disadvantages

- Given the text

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
I	n		t	h	e		m	i	d	d	a	y		b	r	e	e	z	e	,		w	h
24	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*
e	r	e		t	h	e		w	i	l	l	o	w		i	s		s	w	a	y	i	n
48	*49*	*50*	*51*	*52*	*53*	*54*	*55*	*56*	*57*	*58*	*59*	*60*	*61*	*62*	*63*	*64*	*65*	*66*	*67*	*68*	*69*	*70*	*71*
g	,		f	l	o	w	e	r	s		a	r	e		b	l	o	o	m	i	n	g	.

Array disadvantages

- Change the text “where the willow is swaying” to “where the willows are swaying”
  - (1) Reallocate array of larger size (74)



Array disadvantages

- Change the text “where the willow is swaying” to “where the willows are swaying”
  - (2) Copy from old array up to (including) “willow”

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
I	n		t	h	e		m	i	d	d	a	y		b	r	e	e	z	e	,		w	h

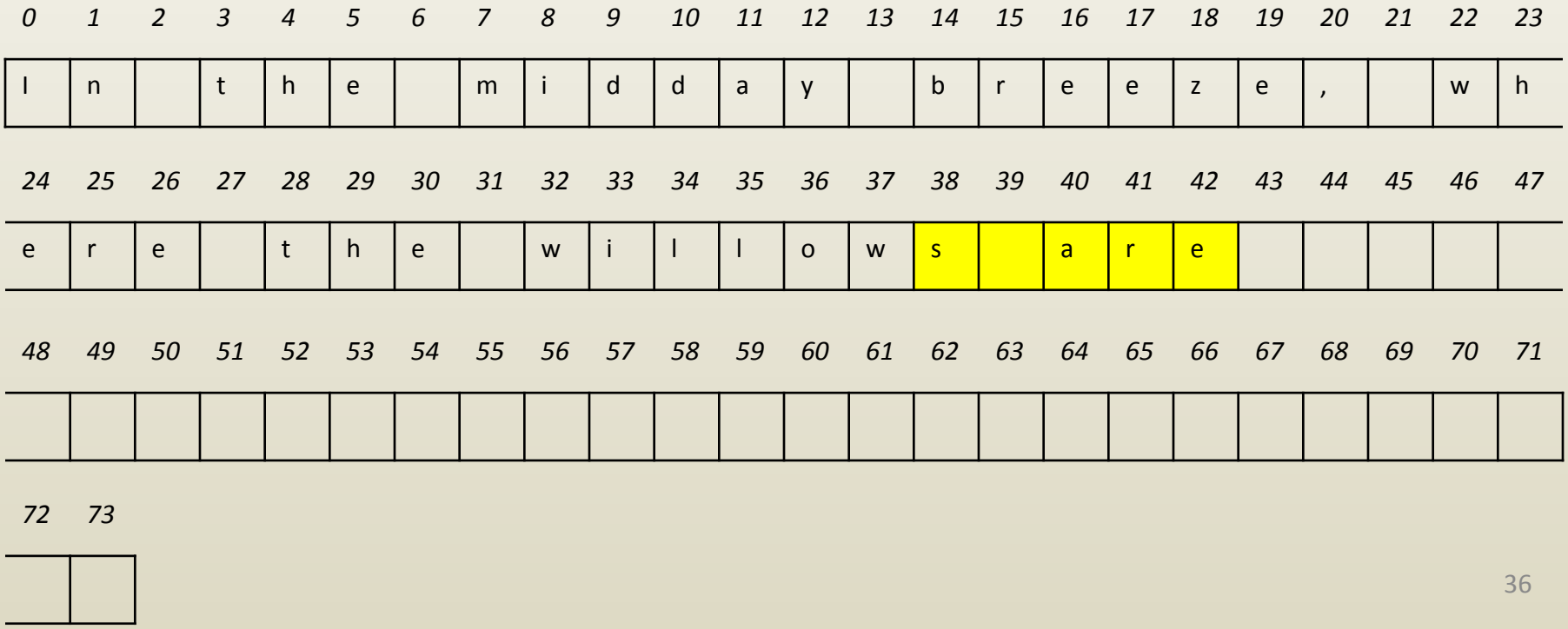
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
e	r	e		t	h	e		w	i	l	l	o	w										

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71

72	73

Array disadvantages

- Change the text “where the willow is swaying” to “where the willows are swaying”
  - (3) Write new text “s are”



Array disadvantages

- Change the text “where the willow is swaying” to “where the willows are swaying”
  - (4) Copy from old array from “ swaying” to end.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
I	n		t	h	e		m	i	d	d	a	y		b	r	e	e	z	e	,		w	h

24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
e	r	e		t	h	e		w	i	l	l	o	w	s		a	r	e		s	w	a	y

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
i	n	g	,		f	l	o	w	e	r	s		a	r	e		b	l	o	o	m	i	n

72	73
g	.

Array modification

- Allocate array of new length
- Copy data from old array as needed
- Copy new data as needed
- Release old array
- Example: given an array A with 50 elements, insert 10 elements after element with index 15
  - Allocate array B with 60 elements
  - Copy elements 0-14 from A to B at 0-14
  - Copy new elements 0-9 to B at 15-24
  - Copy elements 15-49 from A to B at 25-59
  - Release old array A