Algorithms and Data Structures

Andrzej Pisarski

Plan of the lecture

- Arrays
- Ordered Arrays & Binary Searches
- Pointers
- Big O Notation

Arrays

Exampe: Kids-league baseball

- Which players are present at the practice field?
  1. A simple data structure to hold data
  2. Actions to perform on this data structure:
     - ❑ Insert a player into when the player arrives,
     - ❑ Searching for a particular player (by a number),
     - ❑ Delete a player.

Arrays

Exampe: Kids-league baseball

- Insert a player (32)

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53			

Arrays

Exampe: Kids-league baseball

- Insert a player (32)

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53			

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53	32		

Arrays

Exampe: Kids-league baseball

- Insert a player (32)

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53			

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53	32		

What will happen if we insert a player (32) once again?

Arrays

Exampe: Kids-league baseball

- Insert a player (32)

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53			

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53	32		

What will happen if we insert a player (32) once again?

Assumption: no duplication

Arrays

Exampe: Kids-league baseball

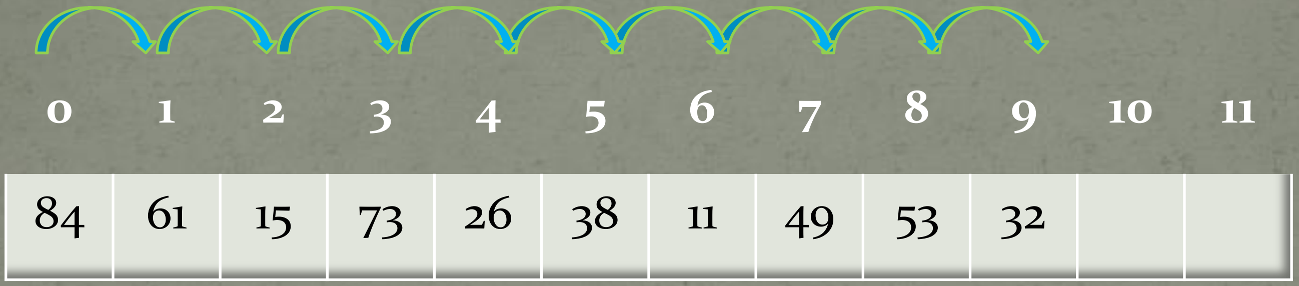
- Searching for a particular player (32)

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53	32		

Arrays

Exampe: Kids-league baseball

- ❑ Searching for a particular player (32)



0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53	32		

Arrays

Exampe: Kids-league baseball

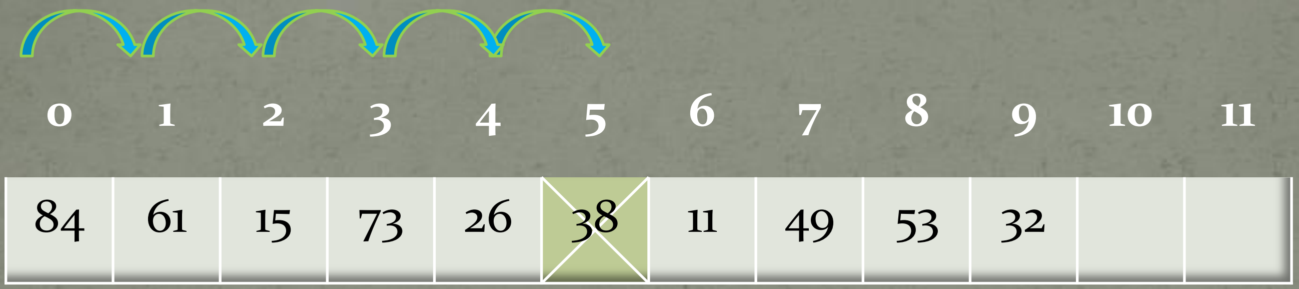
- Delete a player (38)

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53	32		

Arrays

Exampe: Kids-league baseball

- Delete a player (38)

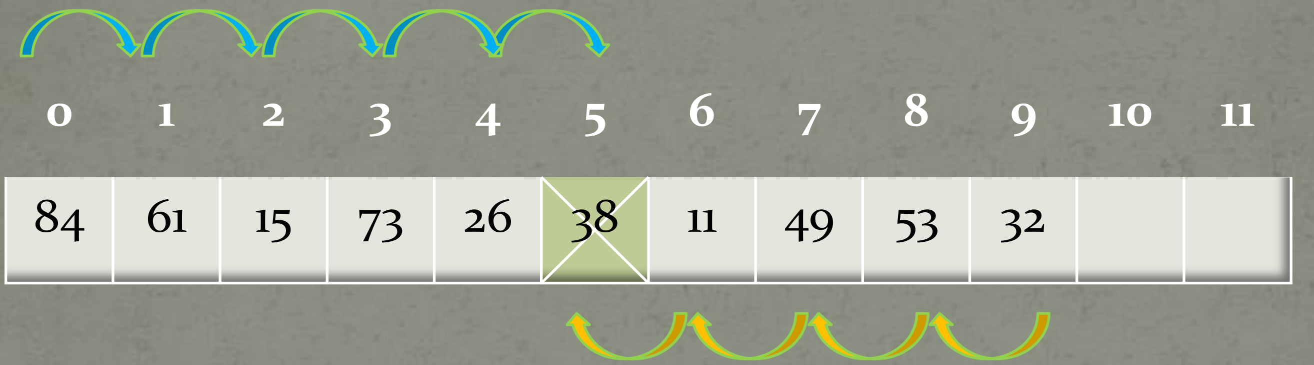


0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	~~38~~	11	49	53	32		

Arrays

Exampe: Kids-league baseball

- Delete a player (38)



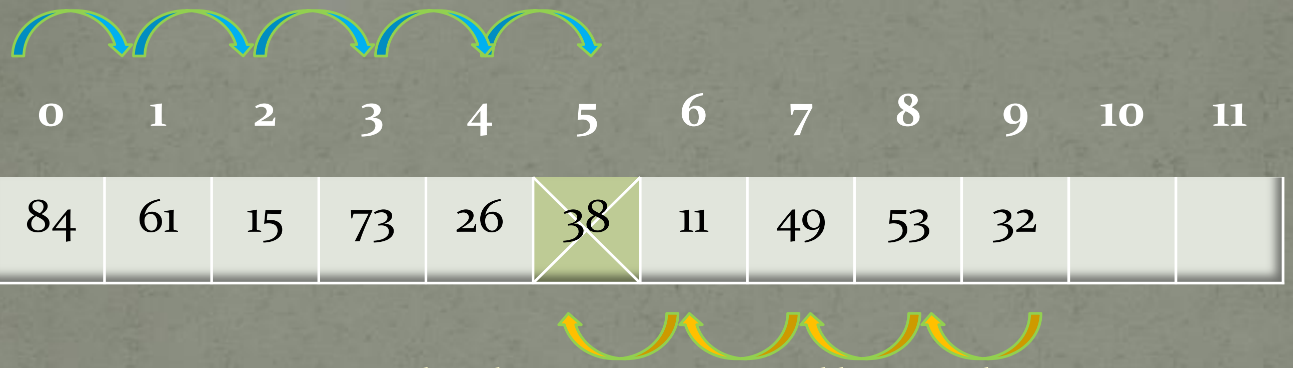
0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	38	11	49	53	32		

Assumption: holes are not allowed

Arrays

Exampe: Kids-league baseball

- Delete a player (38)



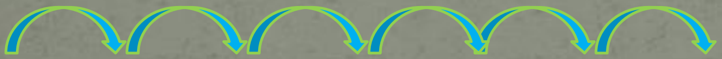
0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	~~38~~	11	49	53	32		

Assumption: holes are not allowed

0	1	2	3	4	5	6	7	8	9	10	11
84	61	15	73	26	11	49	53	32			

Ordered Arrays & Binary Searches

- Data items are arranged on ascending (or descending) key values (number of player, high of player, surname, etc.)
  - ❑ Insert a player (57)



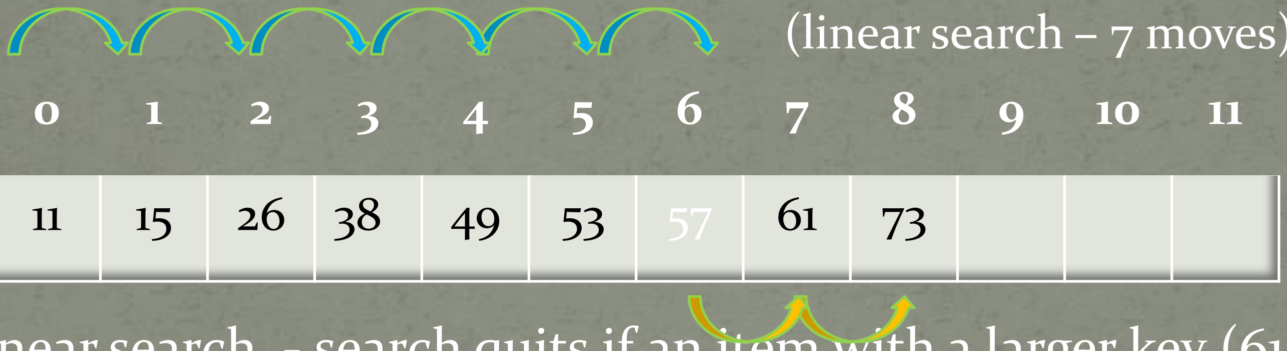
(linear search – 7 moves)

0	1	2	3	4	5	6	7	8	9	10	11
11	15	26	38	49	53	61	73				

Linear search - search quits if an item with a larger key (61) is found

Ordered Arrays & Binary Searches

- Data items are arranged on ascending (or descending) key values (number of player, high of player, surname, etc.)
  - ❑ Insert a player (57)



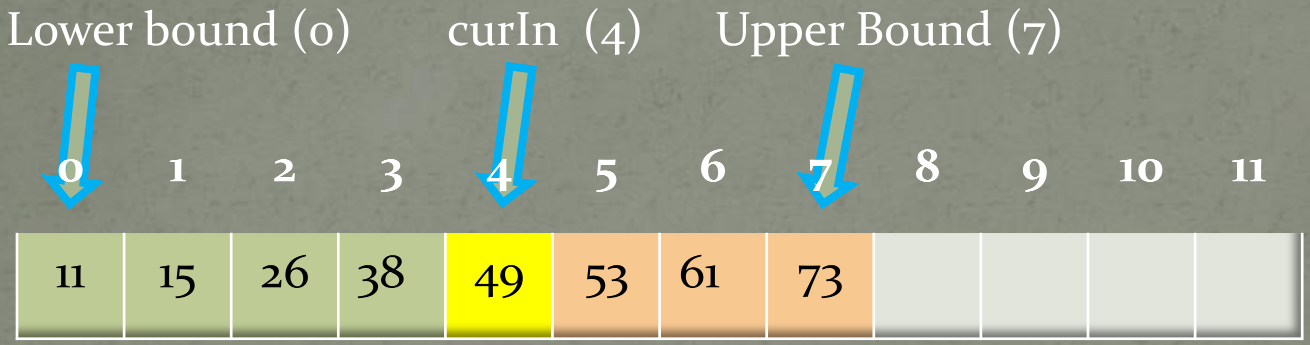
(linear search – 7 moves)

0	1	2	3	4	5	6	7	8	9	10	11
11	15	26	38	49	53	57	61	73			

Linear search - search quits if an item with a larger key (61) is found

Ordered Arrays & Binary Searches

- Insert a player (57) – binary search



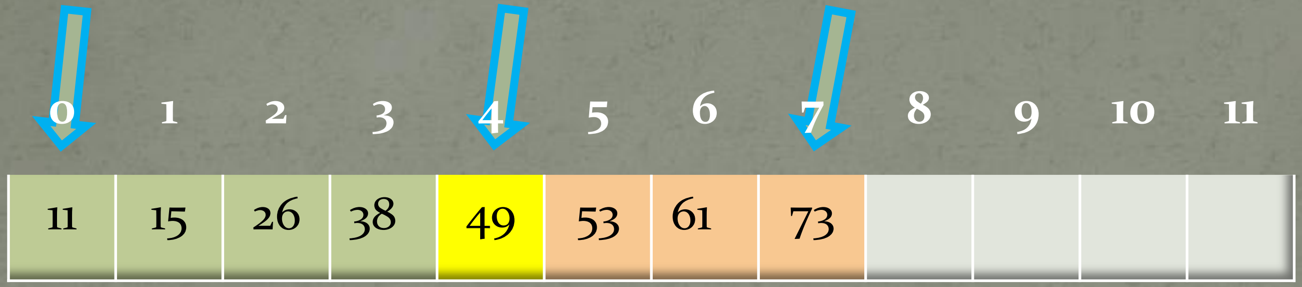
Lower bound (0) curIn (4) Upper Bound (7)

0	1	2	3	4	5	6	7	8	9	10	11
11	15	26	38	49	53	61	73				

Ordered Arrays & Binary Searches

- Insert a player (57) – binary search

Lower bound (0) curIn (4) Upper Bound (7)



0	1	2	3	4	5	6	7	8	9	10	11
11	15	26	38	49	53	61	73				

0	1	2	3
11	15	26	38

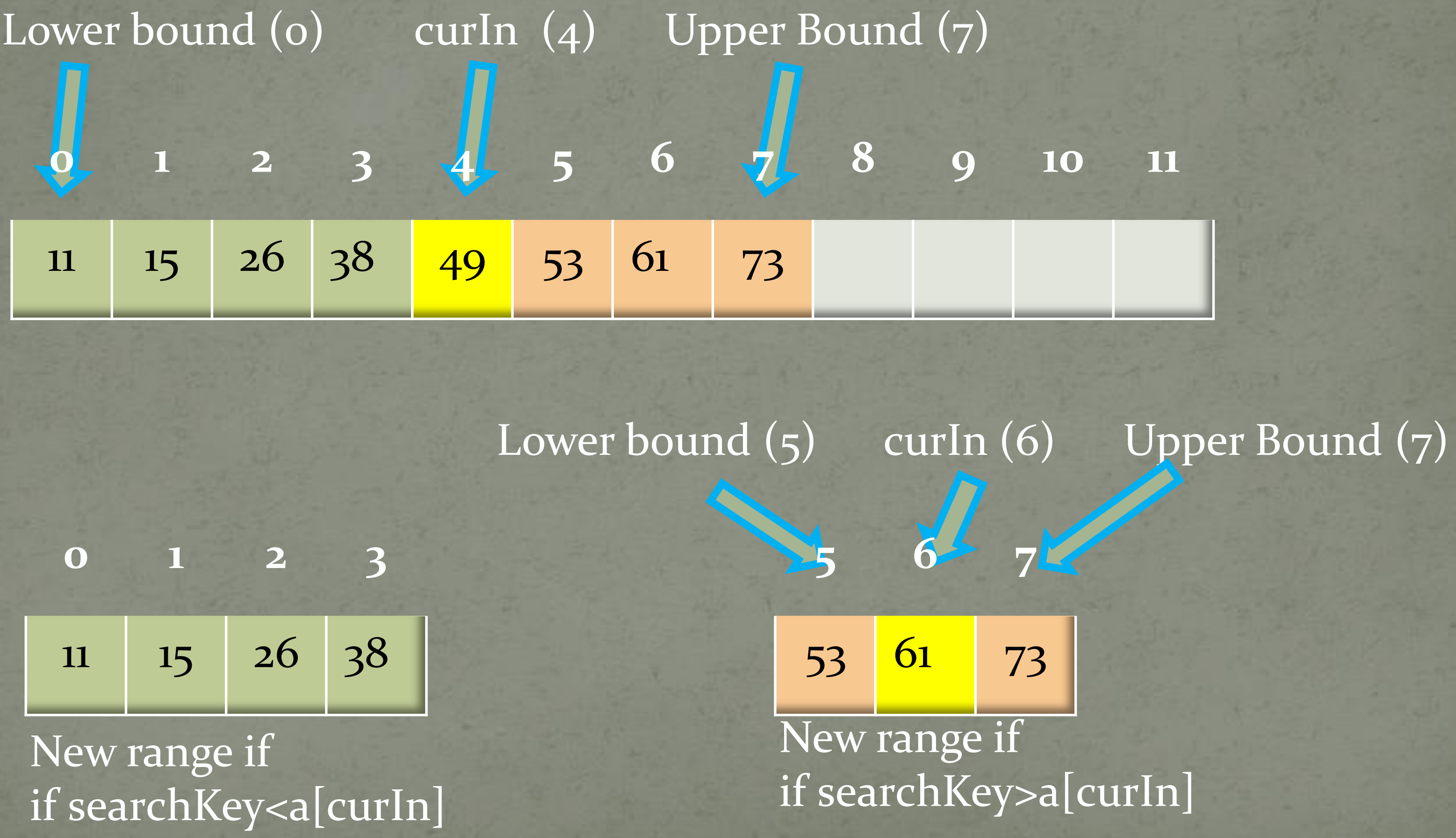
New range if
if searchKey<a[curIn]

5	6	7
53	61	73

New range if
if searchKey>a[curIn]

Ordered Arrays & Binary Searches

- Insert a player (57) – binary search



Lower bound (0) curIn (4) Upper Bound (7)

0	1	2	3	4	5	6	7	8	9	10	11
11	15	26	38	49	53	61	73				

Lower bound (5) curIn (6) Upper Bound (7)

0	1	2	3
11	15	26	38

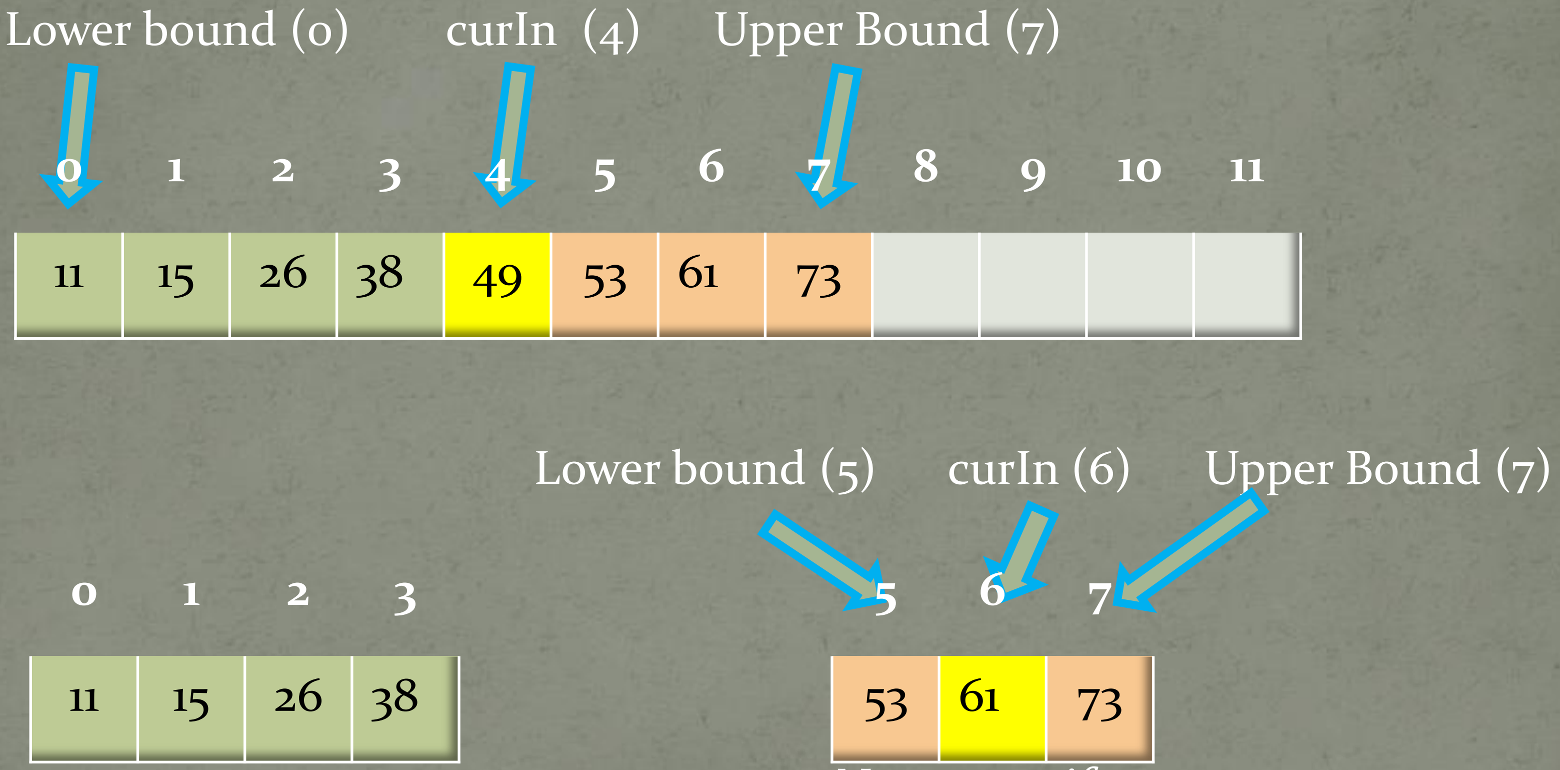
New range if
if searchKey<a[curIn]

5	6	7
53	61	73

New range if
if searchKey>a[curIn]

Ordered Arrays & Binary Searches

- Insert a player (57) – binary search



Lower bound (0) curIn (4) Upper Bound (7)

0	1	2	3	4	5	6	7	8	9	10	11
11	15	26	38	49	53	61	73				

Lower bound (5) curIn (6) Upper Bound (7)

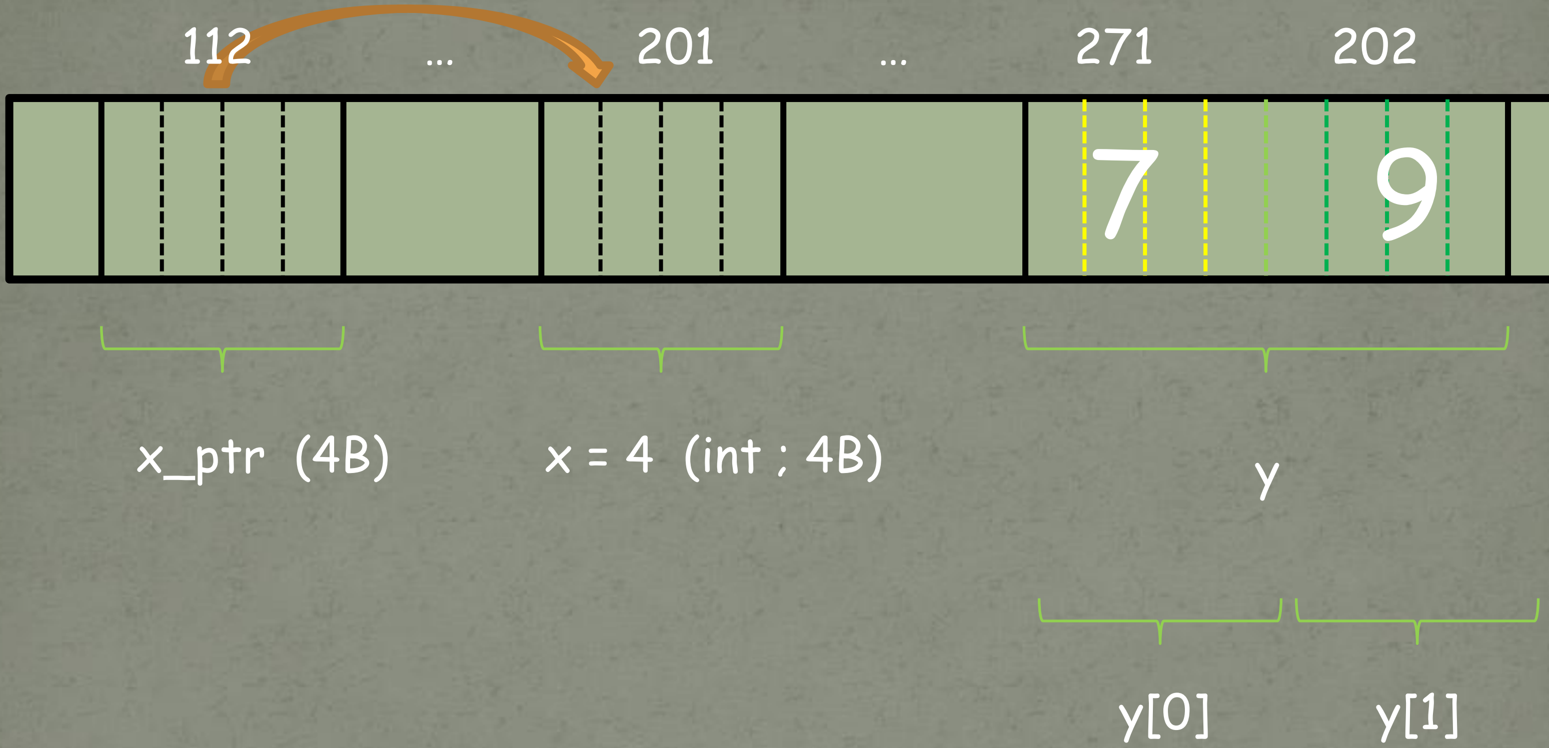
0	1	2	3
11	15	26	38

New range if
if searchKey<a[curIn]

5	6	7
53	61	73

New range if
if searchKey>a[curIn]

Pointers



Pointers

```
int x;
int *x_ptr; /// pointer to an integer

x=4;
x_ptr=&x;    /// & - 'address-of operator'

int y[2]={7,9};
int *y_ptr;
y_ptr=&y[0];

cout << "sizeof(x)= " << sizeof(x) << endl;
cout << "sizeof(x_ptr)= " << sizeof(x_ptr) << endl;

cout << "x= " << x << " *x_ptr= " << *x_ptr << endl;
cout << "address of 'x'= " << x_ptr << ", " << &x << endl;

cout << "address of 'y[o]'= " << y_ptr << ", " << y << endl;
cout << "y[1]= " << *(y_ptr+1) << endl;

int z = 3;
z = *x_ptr + 1;  /// * - dereference operator (value pointed to by ...)
cout << "z= " << z << endl;
```

Big O Notation

Someone could sey: „Algorithm A is twice as fast as algorithm B".

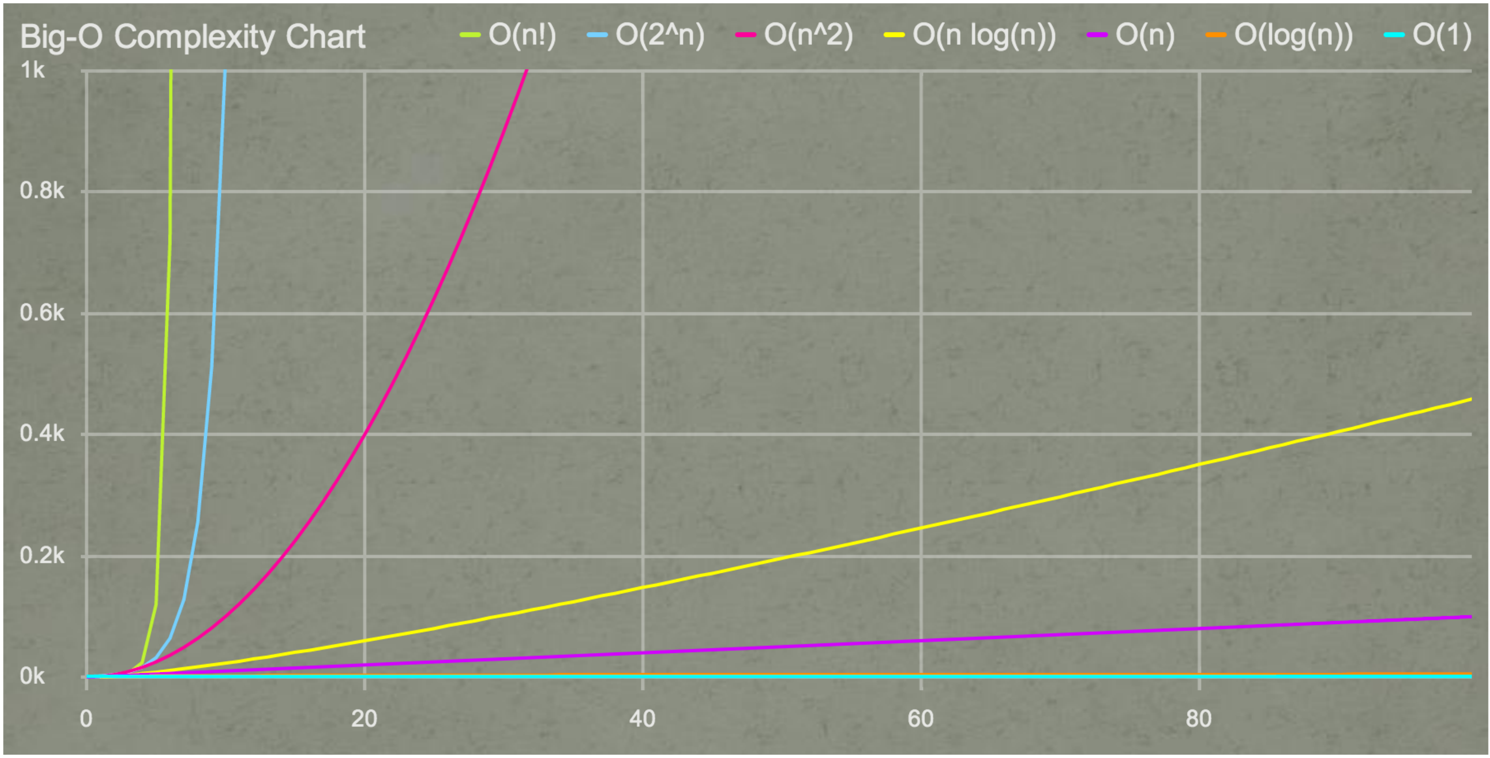
But what's that mean?

What will happen if amount of data change twice, triple, 2^N ?

Big O Notation – lets to set category of efficiency for particular algorithm.

Algoritm	Running Time in Big O Notation
Linear search	O(N)
Binary search	O(log N)
Insertion in unordered array	O(1)
Insertion in ordered array	O(N)

Big O Notation



https://github.com/donbeave/interview