Algorithms and Data Structures

Module 1

Lecture 2

Sorting algorithms, part 1. List data structures.

Adigeev Mikhail Georgievich

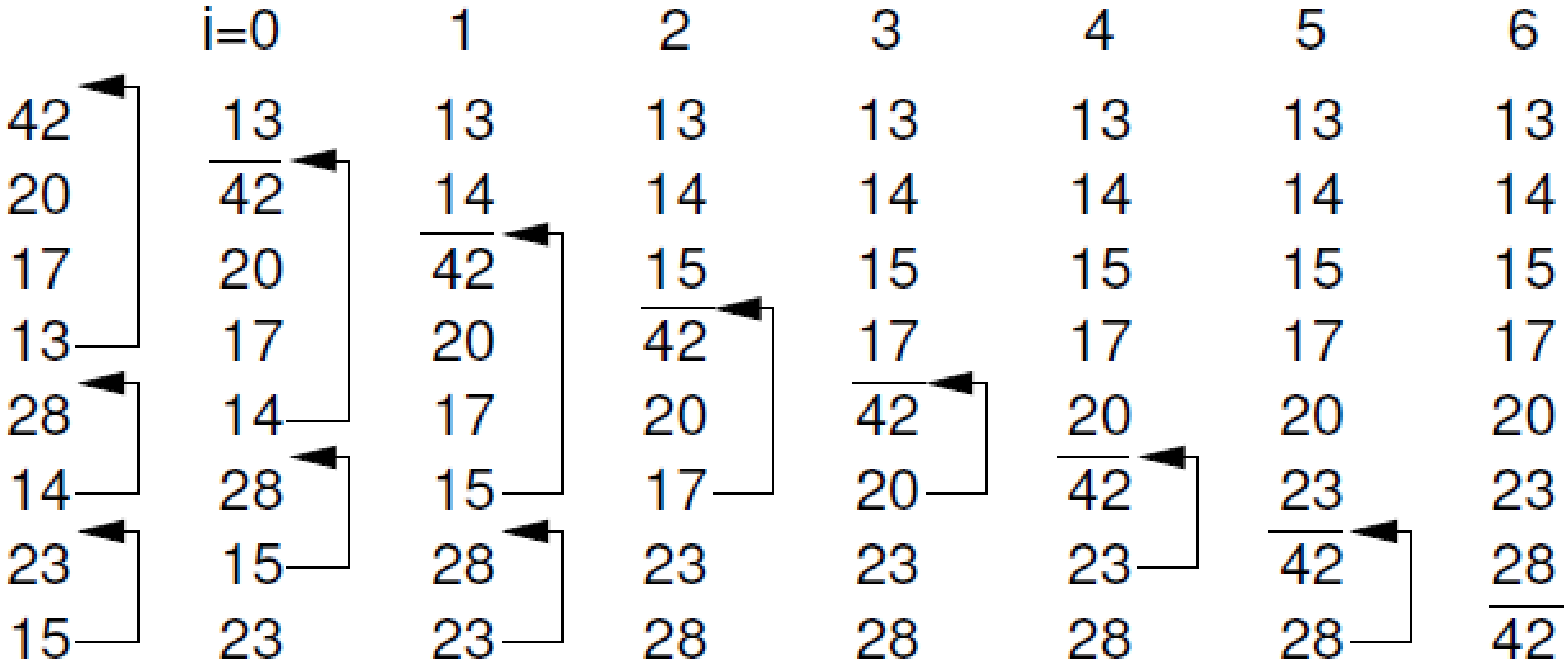
mgadigeev@sfedu.ru

adimg@yandex.ru

Bubble sort

- Scan the array *n* times.
- During each scan: compare neighbor items and swap them if they are ordered incorrectly.

Bubble sort



https://people.cs.vt.edu/~shaffer/Book/

Bubble sort

- Time complexity (=number of comparisons):

$$\sum_{i=1}^{n} i = \frac{n(n-1)}{2} = O(n^2)$$

- Space complexity: $O(n)$, an *in-place* sorting.

Insertion sort

- For each A[i]: insert A[i] to the proper position within A[1..i-1].

	i=1	2	3	4	5	6	7
42	20	17	13	13	13	13	13
20	42	20	17	17	14	14	14
17	17	42	20	20	17	17	15
13	13	13	42	28	20	20	17
28	28	28	28	42	28	23	20
14	14	14	14	14	42	28	23
23	23	23	23	23	23	42	28
15	15	15	15	15	15	15	42

Insertion sort

For each A[i]: insert A[i] to the proper position within A[1..i-1].

Operations:

- Search for the proper position (j) in the sorted part of array.
- Insert A[i] to the position.

Insertion sort

- Search for the proper position (j) in the *sorted* part of array.

=> ‘Dichotomy’ / ‘binary search’.

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key	11	13	21	26	29	36	40	41	45	51	54	56	65	72	77	83

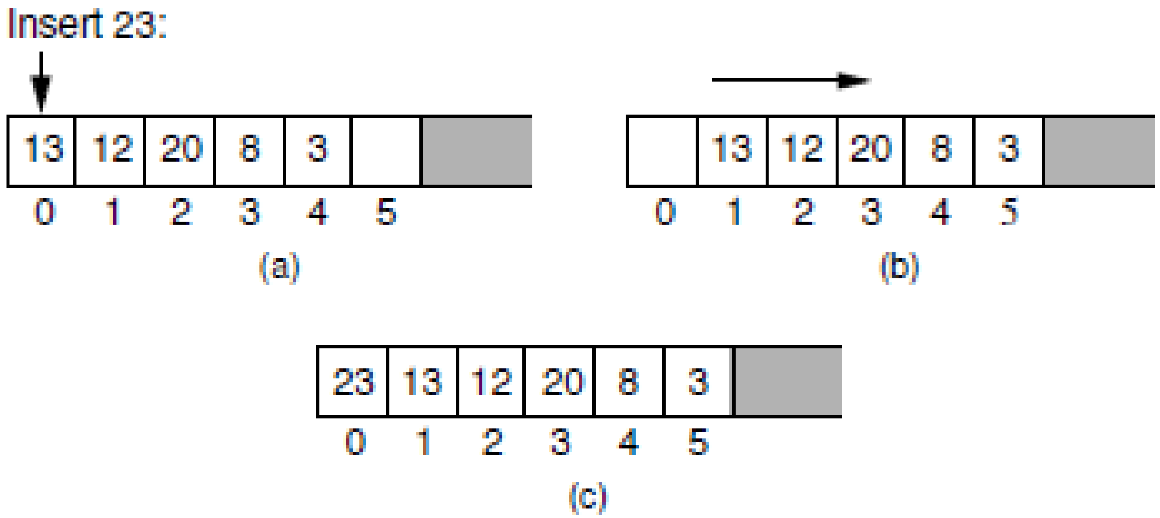
Searching in array of length *n* requires $[\log_2 n]$ comparisons.

Insertion sort

- Insert A[i] to the position.

=> shift A[j..i-1] to the right: $O(i)$ assignments.

$$\sum_{i=1}^{n} i = \frac{n(n-1)}{2} = O(n^2)$$



https://people.cs.vt.edu/~shaffer/Book/

Insertion sort

Total time complexity:

- Search for the proper position (j) in the sorted part of array: $O(n \log n)$ comparisons.
- Insert A[i] to the position: $O(n^2)$ assignments.

Total time complexity: $O(n^2)$.

Space complexity: $O(n)$, an *in-place* sorting.

Insertion sort

Total time complexity:

- Search for the proper position (j) in the sorted part of array: $O(n \log n)$ comparisons.
- Insert A[i] to the position: $O(n^2)$ assignments.

Total time complexity: $O(n^2)$.

Space complexity: $O(n)$, an *in-place* sorting.

Can we reduce the number of operations?

Insertion sort

Total time complexity: $O(n^2)$.

Can we reduce the number of operations?

a) Use another algorithm.

b) Use other data structures.

Insertion sort

```
class BigData
{
	int key;
	char data[1000000000];
}
```

If *assingment* is more costly operation than *comparison*, we can reduce the overall time by performing more comparisons and less assignments.

Data structures

Data structure is a data organization, management, and storage format that enables efficient data *processing* (access and modification).

Examples:

- Array
  - ✓ sorted
  - ✓ unsorted
- List
- Stack
- Queue
- Tree
- …

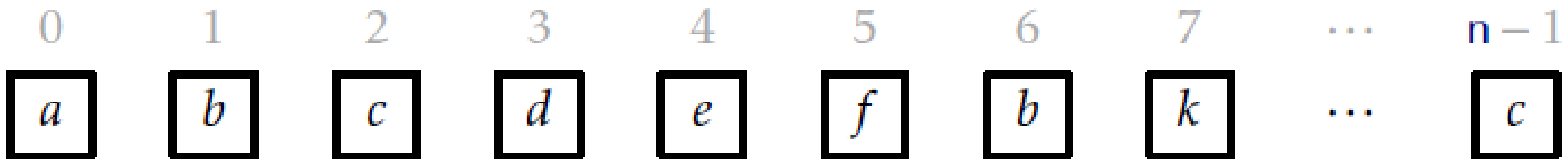
Data structures

Abstract data structure defines an interface, i.e. a set of operations one can perform on data.

Data structure implementation defines the internal representation of the data + algorithmic implementations of operations.

Data structures: List

List : abstract data structure:



Operations:

- size()
- append()
- get(i)
- set(i, x)
- find(x)
- add(i, x)
- remove(i)

Data structures: List

List implementations:

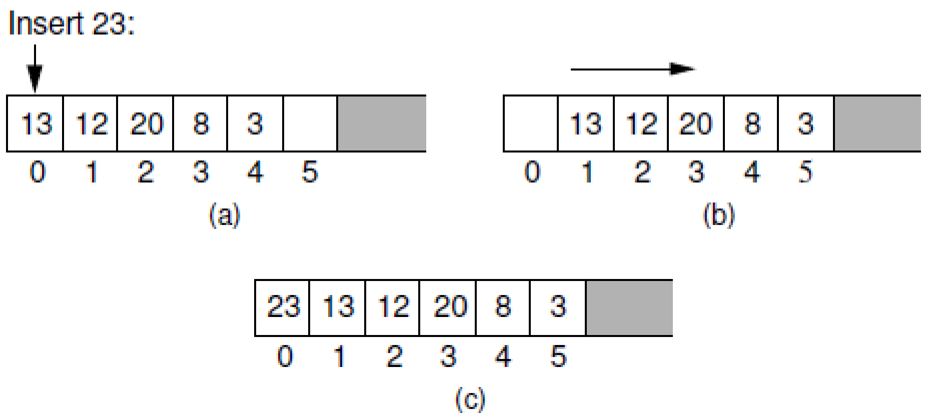
- (static) array-based
  - ✓ unsorted array
  - ✓ sorted array
- (dynamic) linked list

Data structures: List

Static array-based implementations:

```
class ads_array_list
{
    int data[MANY];
    size_t length;

    ads_array_list()
    {
        std::fill(data, data+MANY, 0);
        length = 0;
    }
};
```

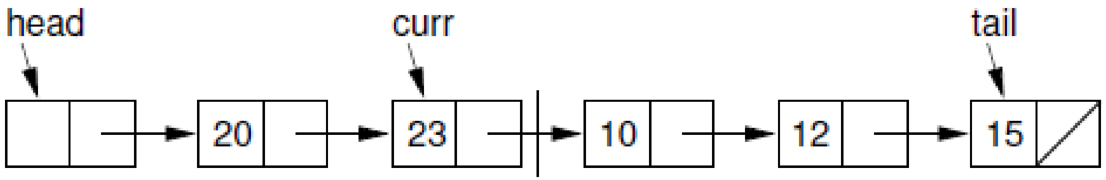


Data structures: List

Dynamic linked list-based implementations:

```
class ads_link_node
{
    int data;
    ads_link_node* next;
    ads_link_node()
    {
        data = 0;
        next = NULL;
    }
};
```

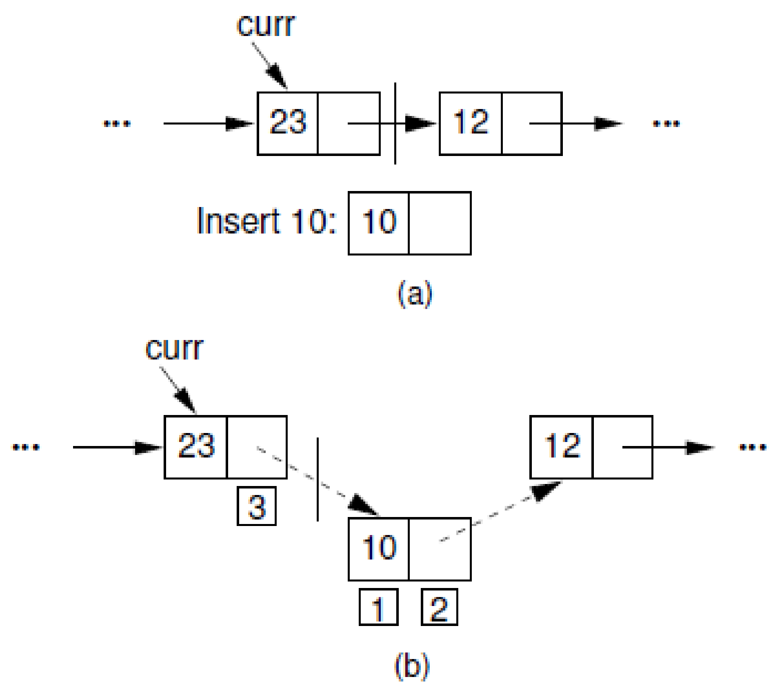
```
class ads_linked_list
{
    size_t length;
    ads_link_node* head;
    ads_link_node* tail;
    ads_linked_list()
    {
        length = 0;
        head = NULL;
        tail = NULL;
    }
};
```



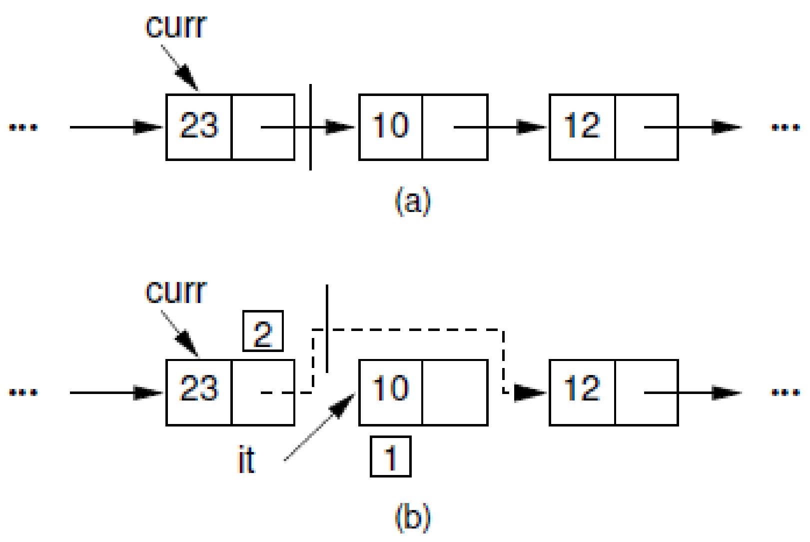
Data structures: List

Dynamic linked list-based implementations:

insert(i, 10)



remove(i)



Data structures: List

Operations' complexities:

	Unsorted array	Sorted array	Linked list
get(i) / set(i, x)	$O(1)$	$O(1)$	$O(n)$
append(x)	$O(1)$	$O(n)$	$O(1)$
add(i, x) / remove(i)	$O(n)$	$O(n)$	$O(1)$
find(x)	$O(n)$	$O(\log n)$	$O(n)$