Algorithms and Data Structures

Module 1

Assignment 1

Time complexity measurements. Sorting.

Adigeev Mikhail Georgievich

mgadigeev@sfedu.ru

adimg@yandex.ru

Programming assignments

- Programming language: C++ (or C)
- Program should have command–line interface
  - ✓GUI is admissible but does not influence the score
- Data input and output via text files
- A solution for a programming assignment should include a zip file with:
  - ✓ a ‘Project’ folder, containing all necessary source (and header) files
  - ✓ an executable file (Release, Win32)
  - ✓ sample input and output files
  - ✓ a .bat file for running the program with command line arguments

Command line interface

```
Program1.exe In.txt 10 Out.txt
```

```
int main(int argc, const char * argv[])
{
    if (argc == 4)
    {
        // argv[0] = program file name
        string InputFile = argv[1];
        int BufLen = atoi(argv[2]);
        string OutputFile = argv[3];
        ...
        return 0;
    }
    else
    {
        cout << "Invalid number of arguments: " << argc << " instead of 3." << endl;
        return 1;
    }
}
```

Assignment 1. Insertion sort implementation using dynamic list

Using ‘Assignment01_Sample.cpp' as a draft, implement insertion sort algorithm storing items to be sorted in a dynamic list. Measure the time complexity of the sorting function.

- You will get 6 points for fully implemented assignment.
- If you implement insertion sort without dynamic linked list structures, i.e. using 'std::vector' data type, you will get 3 points for correct implementation.

Data structures: List

List : abstract data structure:



Operations:

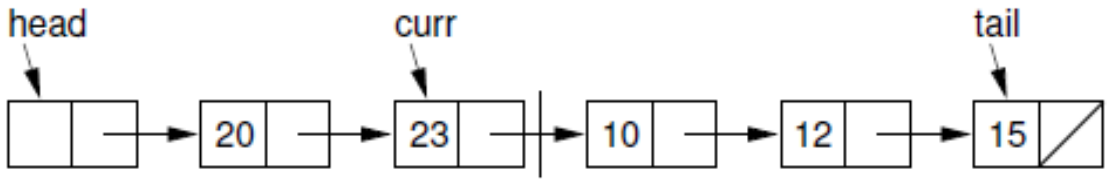
- size()
- append()
- get(i)
- set(i, x)
- find(x)
- add(i, x)
- remove(i)

List: dynamic linked list-based implementation

Dynamic linked list-based implementations:

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

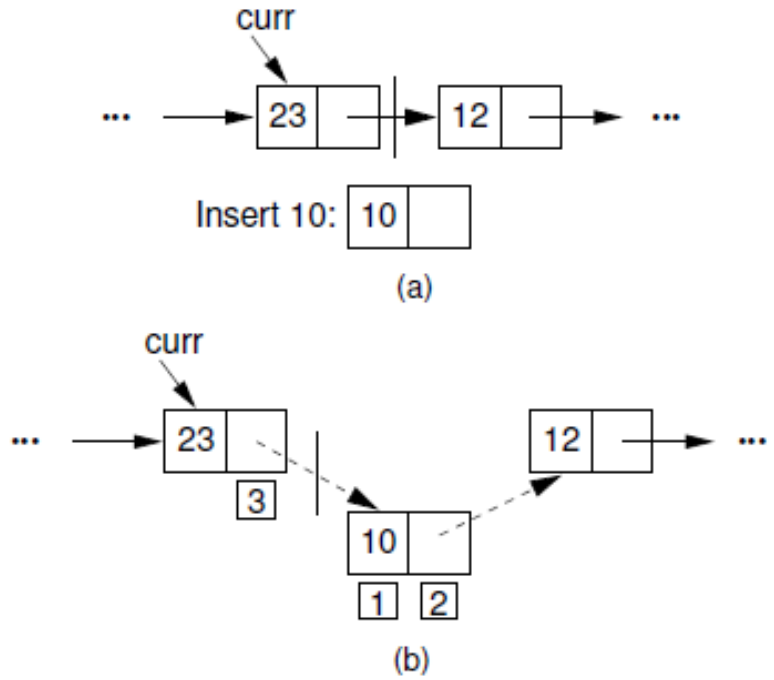
```
class ads_linked_list
{
    size_t length;
    ads_link_node* head;
    ads_link_node* tail;
    ads_linked_list()
    {
        length = 0;
        head = NULL;
        tail = NULL;
    }
};
```



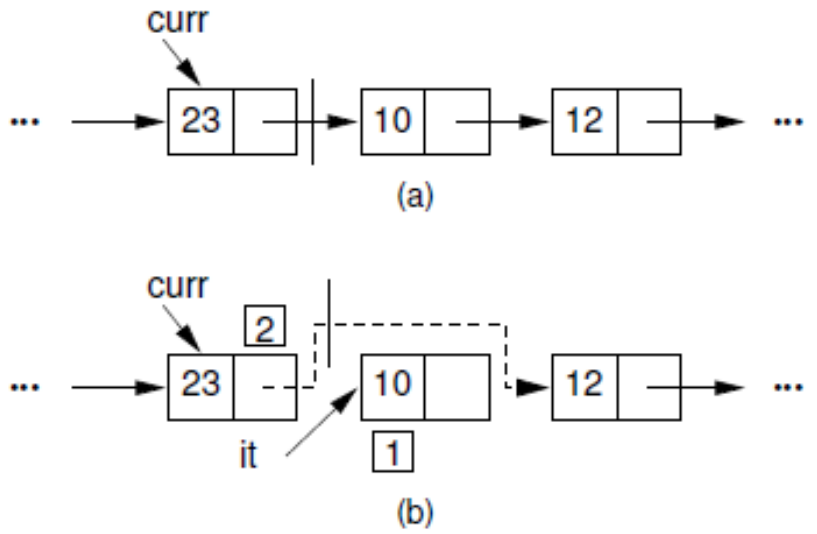
List: dynamic linked list-based implementation

Dynamic linked list-based implementations:

insert(i, 10)



remove(i)



List: dynamic linked list-based implementation

```
class ads_link_node
{
    ads_item item;
    ads_link_node* next;
    friend class ads_linked_list;

public:
    // Counters for assignments and comparisons
    static size_t assignments_count, comparisons_count;

    // Empty constructor
    ads_link_node()
    {
        next = NULL;
    }

    // Constructor with copying ads_item.
    ads_link_node(const ads_item & initial_item)
    :item(initial_item)
    {
        ++assignments_count;
        next = NULL;
    }
};
```

List: dynamic linked list-based implementation

```
class ads_linked_list
{
    size_t length;
    ads_link_node* head;
    ads_link_node* tail;

protected:

    // Binary search for proper position of item within the sorted list.
    size_t binary_search(const ads_item& item, size_t beg=0, size_t end = INT32_MAX)
    {

public:
    ads_linked_list()
    {

    ads_link_node* get(size_t index)
    {

    ads_link_node* add(size_t index, const ads_item& item)
    {

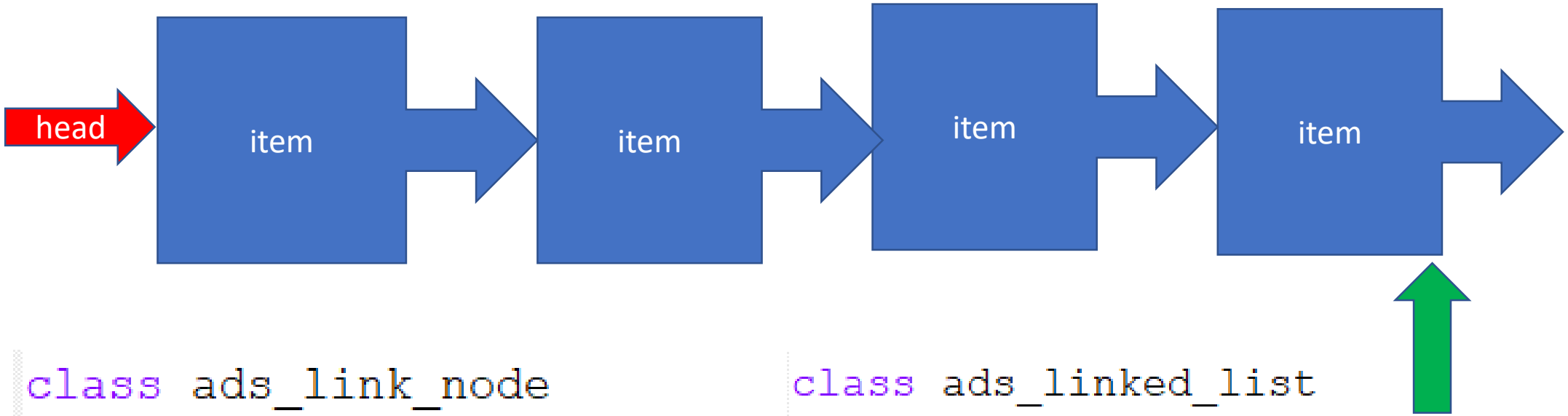
    size_t append(const ads_item& item)
    {
};
```

List: dynamic linked list-based implementation

```
//--------------------------------
// Insertion sort implementation

void insertion_sort_list(const std::vector<ads_item>& unsorted, ads_linked_list& sorted)
{
    // Implement insertion sort for dynamic list as a target storage.
    for (size_t i = 0; i<unsorted.size(); ++i)
    {
        sorted.append(unsorted[i]);
    }
}
```

List: dynamic linked list-based implementation




```
class ads_link_node
{
    ads_item item;
    ads_link_node* next;
```

```
class ads_linked_list
{
    size_t length;
    ads_link_node* head;
    ads_link_node* tail;
```

List: dynamic linked list-based implementation

