Data structures and algorithms

Tutorial 3

Amr Keleg

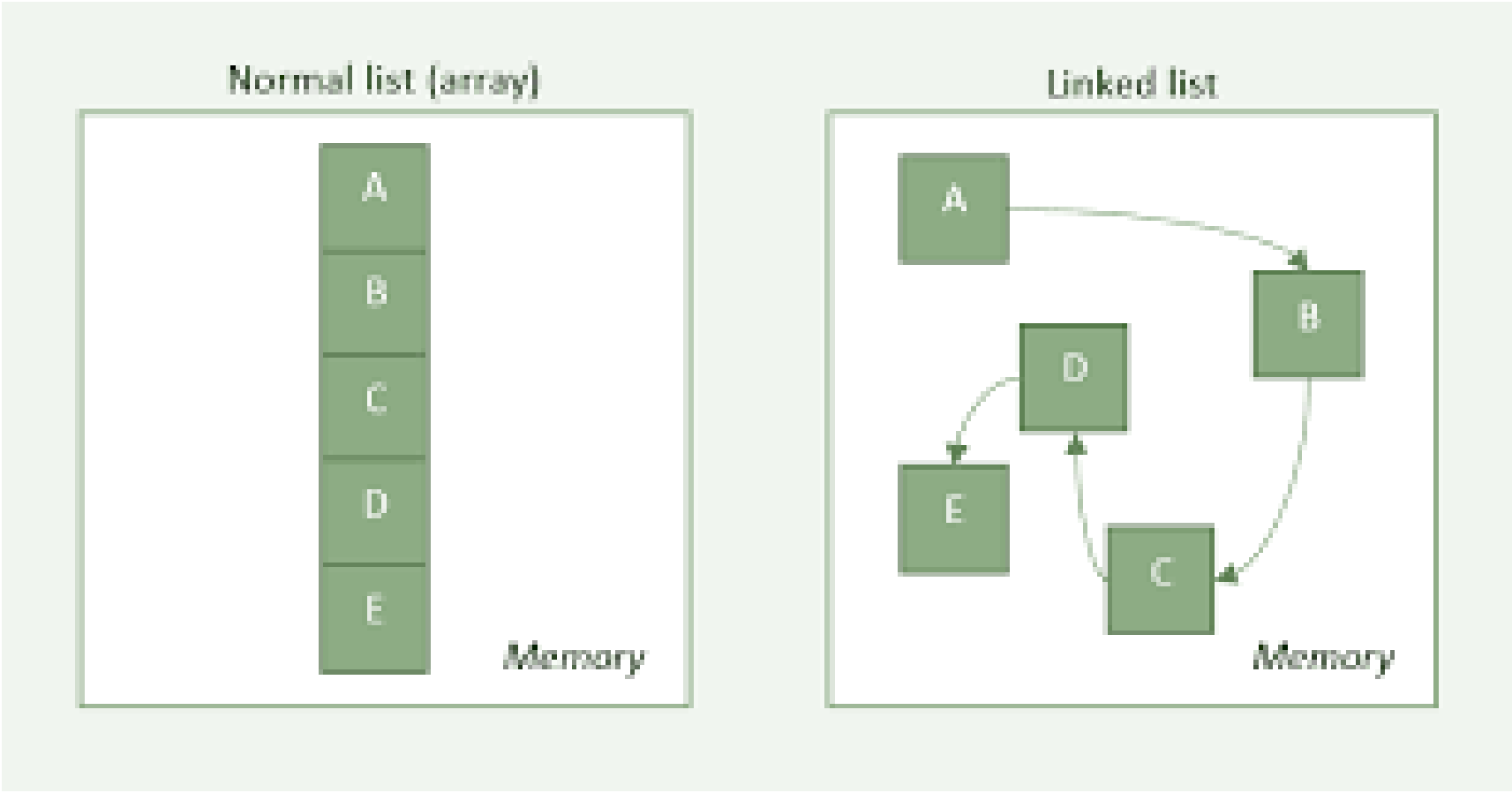
Faculty of Engineering, Ain Shams University

March 10, 2020

Contact: amr_mohamed@live.com

Outline

Outline


Normal list (array)
A
B
C
D
E
Memory
Linked list
A
B
D
E
C
Memory

Outline

```
class Node{
  private:
    int data;
    Node * next;

  public:
    Node(int d){
      this->data = d;
      this->next = nullptr;
    }

    void set_next(Node * next){
      this->next = next;
    }
    Node * get_next(){
      return next;
    }
    // Same for data
    friend class LinkedList;
};
```

Outline

```
class LinkedList{
  private:
    Node * head;

  public:
    LinkedList();
    bool empty();
    int length();
    void push_front(int d);
    void pop_front();

    void push_back(int d);
    void pop_back();
    void clear();
    void bubble_sort();
    void print();
    bool contains(int d);
    ~LinkedList();
};
```

```
void push_back(int d){
  if(empty()){
    push_front(d);
  }
  else{
    Node * last_node = head;
    while(last_node->next != nullptr){
      last_node = last_node->next;
    }
    last_node->next = new Node(d);
  }
}
```

```
void pop_back(){
  if(empty()){
    return;
  }
  Node * pre_last_node = nullptr;
  Node * last_node = head;
  while(last_node->next !=nullptr){
    pre_last_node = last_node;
    last_node = last_node->next;
  }

  delete last_node;

  if (pre_last_node!=nullptr){
    pre_last_node->next = nullptr;
  }
  else{
    head = nullptr;
  }
}
```

```
void clear(){
  while(!empty()){
    pop_front();
  }
}
```

```
~LinkedList(){
   clear();
}
```

Outline

```
// Implement print recursively
void print_node(Node * n){
  if (n==nullptr)
  {
    cout<<"\n";
    return;
  }
  cout<<n->data<<" ";
  print_node(n->next);
}

void print(){
  print_node(head);
}
```

```
bool contains(int d){
  if (empty())
    return false;
  Node * cur_node = head;
  while(cur_node != nullptr){
    if (cur_node->data == d)
      return true;
    cur_node = cur_node->next;
  }
  return false;
}
```

```
// Try implementing these functions
// Delete the element at index
void delete_at(int index);

// Insert a new element after the node at index
void insert_at(int index, int d);
```

Outline

Outline

Given an array, implement a function to sort it.

```
void sort(int arr[], int arr_len){
  // Apply a sorting algorithm
}
```

Outline

As long as the array isn't sorted:
Find two successive elements such that arr[i] is bigger than arr[i+1], then swap them.

Outline

```
for (int i=0; i<arr_len -1; i++){
  if (arr[i] > arr[i+1])
    swap(arr[i], arr[i+1]);
}
```

What is the required number of iterations to ensure that the array is sorted?

HINT: What happens to the array after one iteration?

Outline

```
void sort(int arr[], int arr_len){
  for (int iter=0; iter< arr_len; iter++){
    for (int i=0; i<arr_len - 1; i++){
      if (arr[i] > arr[i+1])
        swap(arr[i], arr[i+1]);
    }
  }
}
```

What is the complexity of the bubble sort?

Outline

```
void sort(int arr[], int arr_len){
  for (int iter=0; iter< arr_len -1; iter++){
    bool swapped = false;
    for (int i=0; i<arr_len -1-iter; i++){
      if (arr[i] > arr[i+1]){
        swap(arr[i], arr[i+1]);
        swapped = true;
      }
    }
    if (!swapped){
      return ;
    }
  }
}
```

What is the complexity of improved bubble sort?

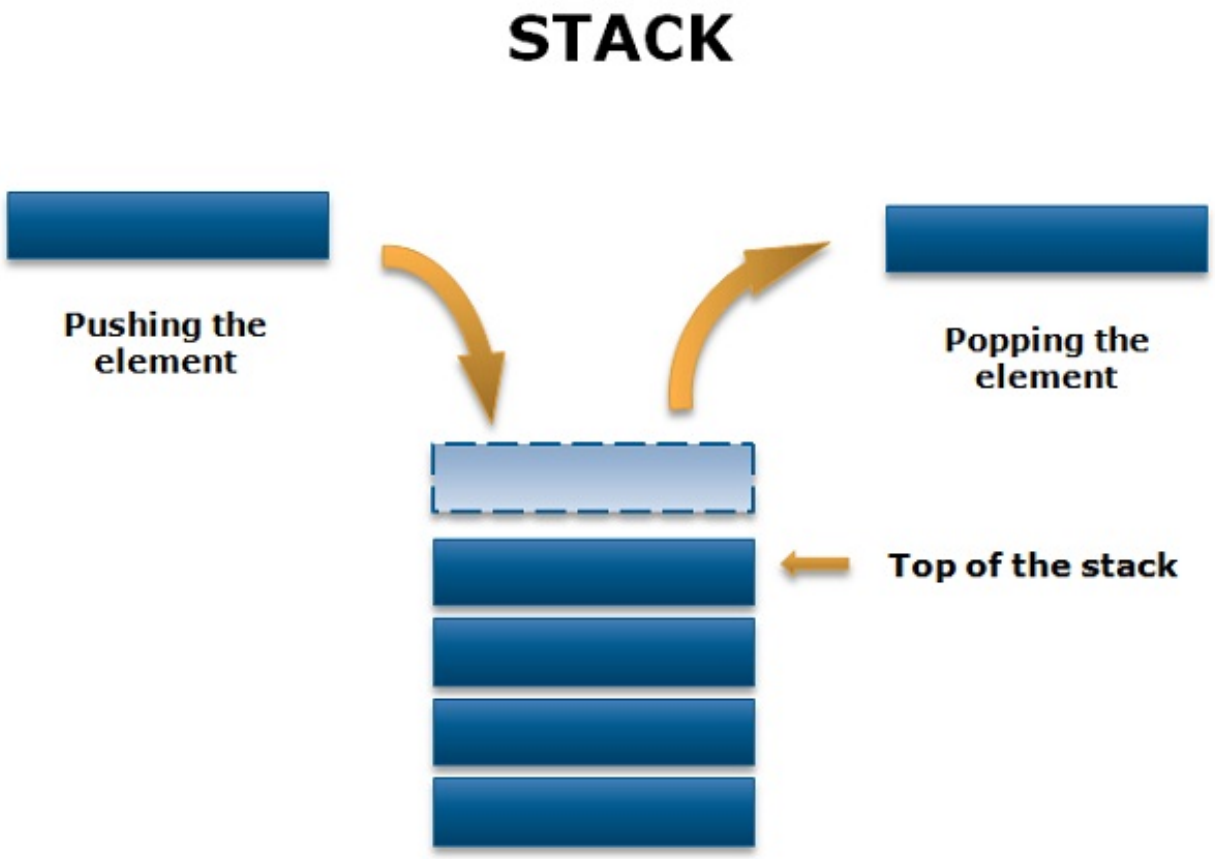
Visualization: `https://visualgo.net/bn/sorting`

Outline

```
void bubble_sort(){
  if(empty() || head->next == nullptr)
    return;
  for(int it=0; it<length() - 1; it++){
    Node * pre_last_node = head;
    Node * last_node = head->next;
    while(last_node != nullptr){
      if (pre_last_node->data > last_node->data){
        swap(pre_last_node->data, last_node->data);
      }
      pre_last_node = last_node;
      last_node = last_node->next;
    }
  }
}
```

Outline

Outline


STACK
Pushing the element
Popping the element
Top of the stack

```
class Stack{
  public:
    Stack();
    int top();
    void push(int v);
    void pop();
};
```

Outline

```
class Stack{
    int arr[1000];
    int size;
  public:
    Stack(){ size=0; }
    int top(){
      assert(size>0);
      return arr[size-1];
    }
    void push(int v){
      assert(size<1000);
      arr[size] = v;
      size++;
    }
    void pop(){
      assert(size>0);
      size--;
    }
    bool empty(){ return size==0; }
};
```

Outline

4. Create a member function for stack to compare between two stacks, the function should takes one stack as a parameter and compares it to the class stack it return either true or false.

```
class Stack{
    ....
    bool is_equal(Stack s);
};
```

```
bool is_equal(Stack s){
  // Store the popped values in this stack
  Stack temp_stack;
  // Since s is a copy of the second stack, then we will not care about its elements

  // Compare the top elements in both stacks
  while((!s.empty()) && (!this->empty())){
    // The top elements are equal
    if (s.top() == top()){
      temp_stack.push(top());
      // pop the top element in both stacks
      s.pop();
      pop();
    }
    // The top elements in the two stacks aren't equal
    else{
      // recover the elements from the temp_stack;
      while(!temp_stack.empty()){
        this->push(temp_stack.top());
        temp_stack.pop();
      }
      // return that the stacks aren't equal
      return false;
    }
  }

  // At least one of the stacks is currently empty, They are equal if both are empty
  bool stacks_are_equal = s.empty() && empty();
  // First, recover the elements to the current stack
  while(!temp_stack.empty()){
    this->push(temp_stack.top());
    temp_stack.pop();
  }
  return stacks_are_equal;
```

```
// Another solution
bool is_equal(Stack s){
  while(!empty() && !s.empty()){
    if (top() != s.top())
      return false;
    pop();
    s.pop();
  }
  return empty() && s.empty();
}
```

What is the problem here?

```
bool is_equal(Stack s){
  Stack c = *this;
  while(!c.empty() && !s.empty()){
    if (c.top() != s.top())
      return false;
    c.pop();
    s.pop();
  }
  return c.empty() && s.empty();
}
```

What will happen if the array was dynamically allocated? - This code won't work as expected since the object c won't be copied properly (Shallow copy). - We will need to implement the copy constructor in class Stack and make sure Deep copy is performed.

```
bool is_equal(Stack s){
  Stack c = *this;
  while(!c.empty() && !s.empty()){
    if (c.top() != s.top())
      return false;
    c.pop();
    s.pop();
  }
  return c.empty() && s.empty();
}
```

Outline

Use stack to check the consistency of an XML file, the XML is based on having opening tag and closing tag, between the open and closing tags there exist the information for the element. a consistent XML file has balanced number of open and closing tags. For example the opposite figure shows a balanced XML file

```
<?xml version="1.0"?>
- <job>
  - <production>
      <ApprovalType>WebCenter</ApprovalType>
      <Substrate>carton 150 gr</Substrate>
      <SheetSize>220-140</SheetSize>
      <press>SuperFlat2</press>
      <finishing>standard</finishing>
      <urgency>normal</urgency>
    </production>
  - <customer>
      <name>FruitCo</name>
      <number>2712</number>
      <currency>USD</currency>
    </customer>
</job>
```

Outline

Assumption: < **and** > **for** the same tag will appear in the same line.

- Read the file line by line
- Ignore comments and preprocessor tags
- While the line contains tags
- Extract them
- If openning tag - Push to stack
- If closing tag - Check the top of the stack
- If stack isn't empty in the end - ERROR

Feedback form:

Amr: `https://forms.gle/Wut8xjzbCjNbzCqv5`

Fady: `TODO`