# 

# Department of Computing

**CS 250: Data Structures and Algorithms**

# Name: Mahum Samar

# CMS ID: 290647

# Class: BSCS-9A

**Lab 07: Asymptotic Complexity of Algorithms**

**CLO1: Understand the fundamentals of data structures and algorithms**

**Date: November 23, 2020**

**Time: 10:00 am -1:00pm**

# Instructor: Dr. Yasir Faheem

# Lab 7: Asymptotic Complexity Analysis

**Introduction**

This lab is based on the analysis of different algorithms.

**Objectives**

Objective of this lab is to make students analyze different algorithms and their asymptotic complexities.

**Tools/Software Requirement**

Visual Studio 2012 or gcc or g++

**Tasks**

**Task 1:**

You have already implemented a function that prints all elements of a list of size n, where n>=0. What is the Big-O complexity of that operation?

|  |
| --- |
| Worst case Complexity:  Big-O = O(n) |

**Task 2:**

In assignment 1, you were asked to implement a function that prints all elements of a singly linked list in the **reverse order.** Your task is to answer the following questions:

1. Suppose the elements of a singly linked list are printed using an iterative approach with the help of two nested loops. What is the Big-O time complexity of printing n values in the reverse order? What is the Big-O **space complexity**?

|  |  |
| --- | --- |
| Worst case Complexity:  Big-O = O() | Space Complexity:  O(1) |

1. Suppose the elements are printed using a recursive function given below. What is the Big-O time complexity of this function? What is the Big-O space complexity? Hint: stack, function calls!

Void RecursivePrint( node \*temp){

If (temp!=NULL){

RecursivePrint(temp->next);

cout<<temp->data;

}// end of if.

}

|  |  |
| --- | --- |
| Worst Case Complexity:  Big-O = O(n) | Space Complexity:  O(n) |

**Task 3:**

Suppose you have a **singly linked list** of size **n**. Implement a function takes a position number **pos** as input from the user, and returns the value stored at that position. What is the Big-O time complexity of this function? What is its best-case time complexity?

|  |  |
| --- | --- |
| Worst Case Complexity:  Big-O = O(n) | Best case Complexity:  O(1) |

**Task 4:**

Suppose you have an **array-based list** of size **n**. Implement a function takes a position number **pos** as input from the user, and returns the value stored at that position. What is the Big-O time complexity of this function? What is its best-case time complexity?

|  |  |
| --- | --- |
| Worst Case Complexity:  Big-O = O(1) | Best case Complexity:  O(1) |

**Task 5:**

What is best-case and worst-case time complexity to destroy a linked list of size n?

|  |  |
| --- | --- |
| Worst Case Complexity:  Big-O = O(n) | Best case Complexity:  O(n) |

**Task 6:**

What is best-case and worst-case time complexity to destroy an array-based list of size n?

|  |  |
| --- | --- |
| Worst Case Complexity:  Big-O = O(1) | Best case Complexity:  O(1) |

**Task 7:**

Your task is to reverse the order of all n elements of a singly linked list using stack. Declare a stack of pointers to class node (It should store the address to an object of class node). Travers the linked list and push the address of every node onto a stack. Pop the elements and update the links by reversing order of nodes in a list. Update the start and last pointers. What is the best case and Big-O time complexities to reverse a linked list using this approach? What is its Big-O space complexity?

CODE:

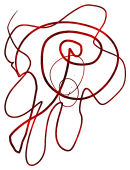
|  |
| --- |
| # include <iostream>  using namespace std;  #define SIZESTACK 100  template <class StackType> class Stack  {  //generic stack class.  public:  int top; //to keep track of the top of the stack  StackType array[SIZESTACK]; //array for implementing array based implementation.  Stack()  {  top = -1; //initializing the stack  }  bool isEmpty()  {  //method to check if the stack is empty  if(top == -1)  {  //if stack empty  return true;  }  else  {  //if stack is not empty  return false;  }  }  bool isFull()  {  //method to check if the stack is full  if(top == (SIZESTACK-1))  {  //stack full  return true;  }  else  {  return false;  }  }  void Push(StackType x)  {  //method to push the values in the stack  if(isFull())  {  cout << "push is full Stack Overflow.";  }  else  {  top++;  array[top] = x;  }  //PrintStack();  }  StackType Pop()  {  //method to pop the values out of the stack  if(isEmpty())  {  cout << "pop is empty Stack Underflow.";  }  else  {  StackType x = array[top];  top--;  return x;  }  }  StackType Top()  {  //method to get the top value stored in the stack  if(isEmpty())  {  cout << "top is empty Stack Underflow.";  }  else  {  return (array[top]);  }  }  void PrintStack()  {  //method to print the stack on the screen  while(top != -1)  {  cout << " " << Top();  Pop();  }  cout << endl;  }  };  class ListNode  {  public:  int data;  ListNode \*next;  };  class SinglyLinkedList  {  public:  ListNode \*headNode;  ListNode \*lastNode;  SinglyLinkedList()  {  headNode = NULL;  lastNode = NULL;  }  bool isEmpty()  {  return headNode == NULL;  }  void InsertNode(int x)  {  ListNode \*newNode = new ListNode();  newNode -> data = x;  if(isEmpty())  {  headNode = newNode;  lastNode = newNode;  }  else  {  lastNode -> next = newNode;  lastNode = newNode;  newNode -> next = NULL;  }  }  void Print()  {  if(isEmpty())  {  cout << "The list is empty." << endl;  }  else  {  ListNode \*temp = new ListNode();  temp = headNode;  while(temp != NULL)  {  cout << temp -> data << " ";  temp = temp -> next;  }  }  }  void ReversingTheList(SinglyLinkedList \*singlyLinkedList)  {  //method to reverse the list by passing the linked list address  Stack <ListNode\*> \*stackChar = new Stack<ListNode\*>();  ListNode \*tempNode = new ListNode();  tempNode = singlyLinkedList -> headNode;  while(tempNode -> next != NULL)  {  //pushing nodes into the stack  stackChar -> Push(tempNode);  tempNode = tempNode -> next;  }  singlyLinkedList -> headNode = tempNode;  while(!(stackChar -> isEmpty()))  {  //poping the values from stack and putting into the list again in reverse order.  tempNode -> next = stackChar -> Top();  stackChar -> Pop();  tempNode = tempNode -> next;  }  tempNode -> next = NULL;  }  };  int main()  {  SinglyLinkedList \*singlyLinkedList = new SinglyLinkedList();  singlyLinkedList -> InsertNode(1);  singlyLinkedList -> InsertNode(2);  singlyLinkedList -> InsertNode(3);  singlyLinkedList -> InsertNode(4);  singlyLinkedList -> InsertNode(5);  singlyLinkedList -> InsertNode(6);  singlyLinkedList -> InsertNode(7);  cout << "printing list" << endl;  singlyLinkedList -> Print() ;  cout << endl;  singlyLinkedList -> ReversingTheList(singlyLinkedList);  cout << "Priting the list after reversing it using the stack: " << endl;  singlyLinkedList -> Print();  } |

|  |  |  |
| --- | --- | --- |
| Worst Case Complexity:  Big-O = O(n) | Best case Complexity  O(1)  If list is empty or only one node. | Space Complexity  O(n) |

Task 8:

In the below given table,

|  |  |  |
| --- | --- | --- |
| **Operation** | **Big-O Complexity** | **Best-case Complexity** |
| Insert an element at the front of a singly linked list of size n | O(1) | O(1) |
| Insert an element at the tail end of a singly linked list of size n. **plast** points to last node. | O(1) | O(1) |
| Delete the last node of a singly linked list of size n. **plast** points to its last node. | O(n) | O(1) |
| Insertion at the front of an array list of size n | O(n) | O(1) |
| Insertion at the tail end of an array list of size n | O(1) | O(1) |
| Enqueue in a queue of length n. | O(1) | O(1) |
| Dequeue in a queue of length n. | O(1) | O(1) |
| Converting an expression of length n from infix to postfix form using stack | O(n) | O(n) |
| Finding an element via Binary Search algorithm in a sorted array-list of size n. | O() | O(1) |
| Finding an element via Binary Search algorithm in an **unsorted** array-list of size n. Think about it! | O()  If we first sort the list and then apply the binary search. | O(1) |

****

**Deliverable**You are required to upload the lab tasks on LMS before the deadline.