# **210CT – Programming, Algorithms, Data Structures**

# **Coursework**

# **Georgi Kabadzhov – 02/12/2016**

**Task 1:** Write a function that randomly shuffles an array of integers and explain the rationale

behind its implementation.

// Task1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <stdlib.h>

#include <time.h>

using namespace std;

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void print(int arr[], int n) {

for (int i = 0; i < n; i++) {

cout << arr[i] << endl;

}

}

void randomize(int arr[], int n) {

srand(time(NULL));

for (int i = n - 1; i > 0; i--) {

int j = rand() % (i + 1);

swap(&arr[i], &arr[j]);

}

}

int main()

{

char close;

int arrSize;

cout << "How big do you want your array to be?" << endl;

cin >> arrSize;

int \*A = new int[arrSize];

cout << "Enter your array" << endl;

for (int i = 0; i < arrSize; i++) {

cin >> A[i];

}

randomize(A, arrSize);

print(A, arrSize);

system("pause");

delete[] A;

//BIG O NOTATION is 0(n)

return 0;

}

**Task 2:** Count the number of trailing 0’s in a factorial number

// Task 2.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

using namespace std;

int FindZeros(int n) {

int count = 0;

for (int i = 5; n / i; i \*= 5)

count += n / i;

return count;

}

int main()

{

char cl;

int n;

cout << "What is your factoriel? (input only the integer)" << endl;

cin >> n;

cout << "Count of the trailing zeros in " << n << "! is " << FindZeros(n) << endl;;

system("pause");

return 0;

}

//BIG O notation -> 0(n)

**Task 3:** Write the pseudocode for a function which returns the highest perfect square which is less or equal to its parameter (a positive integer). Implement this in a programming language of your choice

// Week2Task1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <cmath>

using namespace std;

/\*

int number

input number

for (int i<-0, i<number, i++)

if (i\*i <= number)

continue

else

answer <- i

break

print answer

\*/

int main()

{

int number;

int answer;

cout << "Input number: " << endl;

cin >> number;

for (int i = 0; i < number; i++) {

if (pow(i, 2) <= number) {

continue;

}

else {

answer = i - 1;

break;

}

}

cout << answer;

}

**TASK 4:** Look back at last week's tasks. Describe the run - time bounds of these algorithms using Big O notation

Explained in comment below the code

**TASK 5:** Write the pseudocode corresponding to functions for addition, subtraction and multiplication of two matrices, and then compute A=B\*C – 2\*(B+C), where B and C are two quadratic matrices of order n. What is the run-time?

input N

input mat1[][]

input mat2[][]

ADD(mat1[][], mat2[][])

for i <- 0 to N

for k <- 0 to N

result <- mat1[i][k] + mat2[i][k]

return result

SUB(mat1[][], mat2[][])

for i <- 0 to N

for k <- 0 to N

result <- mat1[i][k] - mat2[i][k]

return result

MULTMAT(mat1[][], mat2[][])

for i <- 0 to N

for j <- 0 to N

for k <- 0 to N

result [i][j] += mat1[i][k] \* mat2[k][j]

return result[i][j]

MULTNUM (mat[][], mult)

for i <- 0 to N

for j <- to N

result <- mat[i][j] \* mult

return result

**TASK 6:** Write pseudocode and code for a function that reverses words in a sentence. Input: “This is awesome” Output: “awesome is This”. Give the big O notation

// EasyTask1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <algorithm>

#include <stack>

#pragma warning (disable : 4996) // strtok prevents the program from running. With a bit of google help, this makes it work

/\*PSUEDOCODE

string input

stack output

split string into words (strtok function)

for each token

output.push

while(output != empty)

print(output.head)

output.pop

\*/

using namespace std;

int main()

{

// string text;

stack <string> outputSt;

char str[200]; //= "Hello, i am George and this is awesome.";

cin.getline(str, 200);

char \*pCh = strtok(str, " . ,");

while (pCh != NULL) {

outputSt.push(pCh);

pCh = strtok(NULL, " ");

}

while (!outputSt.empty()) {

cout << outputSt.top() << " ";

outputSt.pop();

}

cout << endl;

system("pause");

return 0;

}

//The runtime is 0(n)

**TASK 7** Write a recursive function (pseudocode and code) to check if a number n is prime (hint: check whether n is divisible by any number below n).

// Week3Task2.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

/\* PSUEDOCODE

ISPRIME(number, div)

if num < 2

not prime

if div = 1

prime

else

if num % div == 0

not prime

else

isPrime(num, div-1)

\*/

using namespace std;

bool isPrime(int num, int x) {

if (num < 2)

return false;

if (x == 1)

return true;

else

if (num % x == 0)

return false;

else

return isPrime(num, x - 1);

}

int main()

{

int x;

cout << "Give me a number: ";

cin >> x;

cout << "Is the number " << x << " prime? " << isPrime(x,x/2) << endl;

system("pause");

return 0;

}

**TASK 8** Write a recursive function (pseudocode and code) that removes all vowels from a given string.

// Week3Task3.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

// PSUEDOCODE:

REMOVEVOWEL(word, counter)

Array[vowels]

If vowel[counter].find in word

Word.erase vowel

REMOVEVOWEL(word, count)

Else

If (vowel[counter] not found)

REMOVEVOWEL(word, counter+1)

Else

Print word

break

void removeVowel(string word, int n) {

char vowels[] = { 'a','e', 'i', 'o', 'u','A','E', 'I', 'O', 'U' };

if (word.find(vowels[n]) != -1) {

word.erase(word.find(vowels[n]), 1);

removeVowel(word, n);

}

else {

if (n <= 8)

removeVowel(word, n + 1);

else {

cout << word;

return;

}

}

}

int main()

{

string word;

cin >> word;

removeVowel(word, 0);

system("pause");

return 0;

}

**TASK 9** Adapt the binary search algorithm so that instead of outputting whether a specific value was found, it outputs whether a value within an interval (specified by you) was found. Write the pseudocode and code and give the time complexity of the algorithm using the Big O notation.

Example input: L = [2,3,5,7,9,13] low= 10 high = 14 Output: True

// Week4Easy1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <cstdlib>

/\*PSEUDOCODE

BINARYRANGE(arr[],first, last,low,high)

if first > last

return false

else

midOfArr<-(first+last)/2

if mid > low AND mid < high

return true

else

if low < mid AND high < mid

BINARYRANGE(arr[],first,mid-1,low,high)

else if low > mid AND high > mid

BINARYRANGE(arr[], mid +1, last, low, high)

BIG O NOTATION:

O(logN)

\*/

using namespace std;

bool binaryS(int A[], int first, int last, int lRange, int hRange) {

if (first > last)

return false;

else {

int mid = (first + last) / 2;

if ((A[mid] > lRange) && (A[mid] < hRange)) {

return true;

}

else {

if ((lRange < A[mid]) && (hRange < A[mid] )) {

binaryS(A, first, mid - 1, lRange, hRange);

}

else if ((lRange > A[mid]) && (hRange > A[mid])) {

binaryS(A, mid + 1, last, lRange, hRange);

}

}

}

}

int main()

{

int A[20];

int first, last, lowRange, highRange;

int size;

cout << "How many numbers do you want to input? (A maximum size of 20)" << endl;

cin >> size;

if (size > 20) {

cout << "Told you. Restart application.";

}

else {

for (int i = 0; i < size; i++) {

cout << "Give me number: ";

cin >> A[i];

}

}

cout << "Give me your bottom range: ";

cin >> lowRange;

cout << "Give me your upper range: ";

cin >> highRange;

first = 1;

last = sizeof(A);

cout << binaryS(A, 0, size, lowRange, highRange) << endl;

system("pause");

return 0;

}

**TASK 10** Given a sequence of n integer numbers, extract the sub-sequence of maximum length which is in ascending order

// Week5Task1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <list>

using namespace std;

int main()

{

int sequence[20];

int currLength = 0;

int finalLength = 0;

int size;

int check = 0;

cout << "How many numbers do you want to input? (A maximum size of 20)" << endl;

cin >> size;

if (size > 20) {

cout << "Told you. Restart application.";

}

else {

for (int i = 0; i < size; i++) {

cout << "Give me number: ";

cin >> sequence[i];

}

}

for (int i = 0; i < size; i++) {

if (sequence[i] < sequence[i + 1]) {

currLength++;

if (i == size) {

if (currLength > finalLength) {

check = i - finalLength+1;

}

}

}

else {

currLength++;

if (currLength > finalLength) {

finalLength = currLength;

check = i - finalLength + 1;

currLength = 0;

}

else {

currLength = 0;

}

}

}

for (int i = check; i < finalLength + check; i++) {

cout << sequence[i] << ", ";

}

system("pause");

return 0;

}

**TASK 11** Double Linked list node deletion function

// Week5Taks2.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

//Nodes and lists use public data members for convenience

//This may make some software engineers froth at the mouth

using namespace std;

class Node

{

public:

int value; //This could really be any type

Node\* next;

Node\* prev;

Node(int val) {

std::cout << "Node constructr!" << std::endl;

this->value = val;

this->next = (Node\*)0;

this->prev = (Node\*)0;

}

~Node() {

std::cout << "Node destructor" << std::endl;

}

};

class List

{

public:

Node\* head;

Node\* tail;

List() {

std::cout << "List Constructor!" << std::endl;

this->head = 0;

this->tail = 0;

}

~List() {

std::cout << "List destructor!" << std::endl;

std::cout << "Todo: properly delete nodes..." << std::endl;

}

void insert(Node\* n, Node\* x) {

//Not actually perfect: how do we prepend to an existing list?

if (n != 0) {

x->next = n->next;

n->next = x;

x->prev = n;

if (x->next != 0)

x->next->prev = x;

}

if (this->head == 0) {

this->head = x;

this->tail = x;

x->prev = 0;

x->next = 0;

}

else if (this->tail = n) {

this->tail = x;

}

}

void deleteNode(Node\* n) {

if (n->prev != NULL) {

n->prev->next = n->next;

}

else {

this->head = n->next;

this->head->prev = 0;

}

if (n->next != 0) {

n->next->prev = n->prev;

}

else {

this->tail = n->prev;

this->tail->next = 0;

}

}

void display() {

Node\* i = this->head;

std::cout << "List: ";

while (i != 0) {

std::cout << i->value << ",";

i = i->next;

}

std::cout << std::endl;

}

};

int main(int argc, char \*argv[])

{

List\* l = new List();

Node n1 = Node(7);

Node n2 = Node(3);

Node n3 = Node(5);

Node n4 = Node(12);

Node n5 = Node(1);

l->insert(0, &n1);

l->insert(l->head, &n2);

l->insert(l->head, &n3);

l->insert(l->head, &n4);

l->insert(l->head, &n5);

l->display();

l->deleteNode(&n1);

l->deleteNode(&n3);

cout << "after delete" << endl;

l->display();

delete l;

system("pause");

return 0;

}

**TASK 12** Implement TREE\_SORT algorithm in a language of your choice, but make sure that the INORDER function is implemented iteratively.

#include "stdafx.h"

#include <iostream>

#include <stack>

using namespace std;

//The class below is the definition of the tree nodes

class BinTreeNode {

public:

BinTreeNode(int value) {

this->value = value;

this->left = NULL;

this->right = NULL;

this->parent = NULL;

this->visited = false;

}

int value;

BinTreeNode\* left;

BinTreeNode\* right;

BinTreeNode\* parent;

bool visited;

};

class StackNodes

{

public:

BinTreeNode\* node;

StackNodes\* next;

StackNodes() {

std::cout << "List Constructor!" << std::endl;

this->node = 0;

this->next = 0;

}

~StackNodes() {

std::cout << "List destructor!" << std::endl;

std::cout << "Todo: properly delete nodes..." << std::endl;

}

void push(StackNodes \*\*top\_ref, BinTreeNode \*node) {

StackNodes\* new\_Node = new StackNodes;

if (new\_Node == NULL) {

cout << "stack empty" << endl;

}

new\_Node->node = node;

new\_Node->next = \*top\_ref;

\*top\_ref = new\_Node;

}

bool isEmpty(StackNodes \*top) {

return (top == NULL) ? 1 : 0;

}

BinTreeNode \*pop(StackNodes \*\*top\_ref) {

BinTreeNode \*res;

StackNodes \*top;

if (isEmpty(\*top\_ref))

cout << "Stack Empty" << endl;

else {

top = \*top\_ref;

res = top->node;

\*top\_ref = top->next;

return res;

}

}

void in\_orderGFG(BinTreeNode \*root) {

BinTreeNode \*current = root;

StackNodes \*s = NULL;

bool done = false;

while (!done) {

//Reach left-most node

if (current != NULL) {

//pointer to a tree node in the stack, before traversing

push(&s, current);

current = current->left;

}

//backtrack to the top. If stack is empty, it's done

else {

if (!isEmpty(s)) {

current = pop(&s);

cout << "%d" << current->value << endl;

current = current->right;

}

else

done = 1;

}

}

}

};

BinTreeNode\* tree\_insert(BinTreeNode\* tree, int item) {

if (tree == NULL)

tree = new BinTreeNode(item);

else

if (item < tree->value)

if (tree->left == NULL) {

tree->left = new BinTreeNode(item);

tree->left->parent = tree;

}

else {

tree\_insert(tree->left, item);

}

else

if (tree->right == NULL) {

tree->right = new BinTreeNode(item);

tree->right->parent = tree;

}

else

tree\_insert(tree->right, item);

return tree;

}

void postorder(BinTreeNode\* tree) {

if (tree->left != NULL)

postorder(tree->left);

if (tree->right != NULL)

postorder(tree->right);

std::cout << tree->value << std::endl;

}

void in\_order(BinTreeNode\* tree) {

if (tree->left != NULL)

in\_order(tree->left);

std::cout << tree->value << std::endl;

if (tree->right != NULL)

in\_order(tree->right);

}

//in\_order function iteratively

void in\_orderITER(BinTreeNode \*node) {

//use of stack so i can iterate

stack<BinTreeNode\*> iteratorSt;

iteratorSt.push(node);

while (!iteratorSt.empty()) {

BinTreeNode \*top = iteratorSt.top();

if (top != NULL) {

if (!top->visited)

iteratorSt.push(top->left);

else {

cout << top->value << " ";

iteratorSt.pop();

iteratorSt.push(top->right);

}

}

else {

iteratorSt.pop();

if (!iteratorSt.empty())

iteratorSt.top()->visited = true;

}

}

}

//SEARCH FUNCTION - Talking about this one specifically, i haven't touched anything else in the code

int BST(BinTreeNode\* tree, int target) {

BinTreeNode\* r = tree;

while (r != NULL) {

if (r->value == target) {

return r->value;

}

else if (r->value > target) {

return BST(tree->left, target);

}

else {

return BST(tree->right, target);

}

return 0;

}

}

int main(int argc, char \*argv[])

{

BinTreeNode\* t = tree\_insert(0, 6);

tree\_insert(t, 10);

tree\_insert(t, 5);

tree\_insert(t, 2);

tree\_insert(t, 3);

tree\_insert(t, 4);

tree\_insert(t, 11);

cout << "Post order, elements are: ";

postorder(t);

cout << "In order, elements are: ";

in\_order(t);

cout << "In order iterative, elements are: ";

in\_orderITER(t);

system("pause");

return 0;

}

**TASK 12** Write the pseudocode for an unweighted graph data structure. You either use an adjacency matrix or an adjacency list approach. Also, write a function to add a new node and a function to add an edge. Following that, implement the graph you have designed in the programming language of your choice. You may use your own linked list from previous labs, the STL LL, or use a simple fixed size array (10 elements would be fine).

// Week7Task1.cpp : Defines the entry point for the console application.

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <queue>

/\*PSEUDOCODE:

new NODE{ value, visited, list of edges }

new GRAPH {list of nodes}

CONNECT\_NODES(Node a, Node b)

add a to b.edges

add b to a.edges

ADD\_EDGE(Node a)

add a to graph

\*/

//I know using namespace std is a bad practice, but for a project with the size of lab task is just more convinient

using namespace std;

class Node {

public:

Node(int value) {

this->value = value;

this->visited = false;

}

int value;

vector<Node\*> edges;

bool visited;

void addEdgeToNode(Node \*v) {

edges.push\_back(v);

}

void printNodAndEd() {

cout << "The Node " << value << " is connected to :" << endl;

for (std::vector<Node\*>::iterator it = edges.begin(); it != edges.end(); it++) {

cout << (\*it)->value << ", ";

}

cout << endl;

}

~Node() {}

};

class Graph {

public:

Graph() {

}

vector<Node\*> headNode;

void insertNode(Node \*node) {

headNode.push\_back(node);

}

void connectNodes(Node \*source, Node \*dest) {

source->addEdgeToNode(dest);

dest->addEdgeToNode(source);

}

void printGraph() {

for (std::vector<Node\*>::iterator it = headNode.begin(); it != headNode.end(); it++) {

(\*it)->printNodAndEd();

}

}

};

int main() {

//Creating pointers to the nodes

Node \*s = new Node(0);

Node \*a = new Node(3);

Node \*b = new Node(7);

Node \*c = new Node(5);

Node \*d = new Node(2);

Node \*e = new Node(6);

Node \*f = new Node(1);

Node \*h = new Node(4);

//Creating pointer to the graph

Graph \*g = new Graph();

//Adding nodes

g->insertNode(s);

g->insertNode(a);

g->insertNode(b);

g->insertNode(c);

g->insertNode(d);

g->insertNode(e);

g->insertNode(f);

g->insertNode(h);

//Connecting nodes

g->connectNodes(s, a);

g->connectNodes(a, b);

g->connectNodes(b, c);

g->connectNodes(c, d);

g->connectNodes(d, e);

g->connectNodes(e, f);

g->connectNodes(d, h);

//OUTPUTS:

g->printGraph();

//Freeing memory from the graph to prevent memory leaks

delete g;

system("pause");

return 0;

}

**TASK 14** Implement BFS and DFS traversals for the above graph. Save nodes into text file.

// Week7Task1.cpp : Defines the entry point for the console application.

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <queue>

#include <stack>

#include <fstream>

//I know using namespace std is a bad practice, but for a project with the size of lab task is just more convinient

using namespace std;

//Creating class Node, which contains the value, structure (vector) of edges and visited flag

class Node {

public:

Node(int value) {

this->value = value;

this->visited = false;

}

int value;

vector<Node\*> edges;

bool visited;

void addEdgeToNode(Node \*v) {

edges.push\_back(v);

}

void printNodAndEd(ofstream& myfile) {

myfile.open("Graph traverses.txt", ios::app);

myfile << "The Node " << value << " is connected to :" << endl;

for (std::vector<Node\*>::iterator it = edges.begin(); it != edges.end(); it++) {

myfile << (\*it)->value << ", ";

}

myfile << endl;

myfile.close();

}

~Node() {}

};

//Creating Graph - a vector of all the nodes inside it

class Graph {

public:

Graph() {

}

vector<Node\*> headNode;

//Adding Node to the graph

void insertNode(Node \*node) {

headNode.push\_back(node);

}

//Connecting to nodes

void connectNodes(Node \*source, Node \*dest) {

source->addEdgeToNode(dest);

dest->addEdgeToNode(source);

}

//Printing the graph in txt file

void printGraph(ofstream& myfile) {

myfile.open("Graph traverses.txt");

myfile << "GRAPH DISPLAYED STARTING FROM NODE 3: " << endl;

myfile << " 3 " << endl;

myfile << " / " << (char)92 << endl;

myfile << " 7 0" << endl;

myfile << " / " << (char)92 << endl;

myfile << " 5 NULL" << endl;

myfile << " / " << endl;

myfile << " 2 " << endl;

myfile << " / | " << (char)92 << endl;

myfile << "4 6 1" << endl;

cout << endl;

for (std::vector<Node\*>::iterator it = headNode.begin(); it != headNode.end(); it++) {

(\*it)->printNodAndEd(myfile);

}

myfile.close();

}

//Breadth first search, outputing nodes values into txt file

void BFS(Node \*head, ofstream& myfile)

{

myfile.open("Graph traverses.txt",ios::app);

myfile << endl;

myfile << "BREADTH FIRST TRAVERSAL: " << endl;

queue<Node\*> traverse;

traverse.push(head);

while (!traverse.empty()) {

Node\* n = traverse.front();

traverse.pop();

if (!n->visited)

{

myfile << "Node value is: " << n->value << endl;

cout << "n value is " << n->value << ", " << endl;

n->visited = true;

for (auto e : n->edges)

traverse.push(e);

}

}

cout << endl;

myfile.close();

}

//Depth first search, outputing nodes values into txt file

void DFS(Node\* head, ofstream& myfile) {

myfile.open("Graph traverses.txt", ios::app);

myfile << endl;

myfile << "DEPTH FIRST TRAVERSAL: " << endl;

stack<Node\*> traverse;

traverse.push(head);

while (!traverse.empty()) {

Node\* n = traverse.top();

traverse.pop();

if (!n->visited) {

myfile << "Node value is: " << n->value << endl;

cout << "n value is " << n->value << ", " << endl;

n->visited = true;

for (auto e : n->edges) {

if (!e->visited) {

traverse.push(e);

}

}

}

}

myfile.close();

}

//Reseting the visited flag of all the nodes, so we can run both of the searches in the same execution

void Reset(Node \*head)

{

queue<Node\*> traverse;

traverse.push(head);

while (!traverse.empty()) {

Node\* n = traverse.front();

traverse.pop();

if (n->visited)

{

n->visited = false;

for (auto e : n->edges)

traverse.push(e);

}

}

}

};

int main() {

//Creating file

ofstream myfile;

//Creating pointers to the nodes

Node \*s = new Node(0);

Node \*a = new Node(3);

Node \*b = new Node(7);

Node \*c = new Node(5);

Node \*d = new Node(2);

Node \*e = new Node(6);

Node \*f = new Node(1);

Node \*h = new Node(4);

//Creating pointer to the graph

Graph \*g = new Graph();

//Adding nodes

g->insertNode(s);

g->insertNode(a);

g->insertNode(b);

g->insertNode(c);

g->insertNode(d);

g->insertNode(e);

g->insertNode(f);

g->insertNode(h);

//Connecting nodes

g->connectNodes(s, a);

g->connectNodes(a, b);

g->connectNodes(b, c);

g->connectNodes(c, d);

g->connectNodes(d, e);

g->connectNodes(e, f);

g->connectNodes(d, h);

//OUTPUTS:

g->printGraph(myfile);

cout << "BFS is: " << endl;

g->BFS(a, myfile);

g->Reset(a);

cout << "DFS is: " << endl;

g->DFS(a, myfile);

//Freeing memory from the graph to prevent memory leaks

delete g;

system("pause");

return 0;

}