Joe Aguilera

CS 161

Professor: Peter Casey

June 9th, 2015

CS 161 final Exam

* 1. 150NlogN
  2. N2
  3. 150NlogN

1. 2n, n3,n2, sqrt(n), Nlog2N, logN
2. 1. N
   2. N2
   3. N
   4. N3
   5. N
3. I have No idea
4. You do not have to go through every element, but if it is written incorrectly, it can mess up.
5. No idea
6. No idea
7. No idea

// Final Exam Code Spring 2015 CS 162

// Joe Aguilera

//

// Implement as many functions as you can.

// Start with the Constructors, test as you go.

#include "stdafx.h"

#include <iostream>

using namespace std;

/\*----------------------------

\* Basic Stack Class

\* Implement using Dynamic Memory

\* Use 'int' for your stack elements

\*/

class Stack {

public:

// Construct Stack with space for n items

Stack(int);

// Copy constructor for Stack

Stack(const Stack&);

// Destructor for Stack

~Stack();

// Add an element to the Stack

// Return a reference to the stack

Stack& push(int);

// Return the value at the top of the Stack

// leaving the stack unchanged

int top();

// Remove the top item from the stack

void pop();

// Return true with empty stack, false otherwise

bool isEmpty();

// Return true with full stack, false otherwise

bool isFull();

void make\_space(int);

int sizeReturn() const;

int get(int) const;

int capacityReturn() const;

private:

int size;

int capacity;

int \* storage;

};

/\* Stack Implementation Code Here \*/

Stack::Stack(int a){

size = a;

capacity = a;

storage = new int[a];

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

storage[i] = 0;

}

}

Stack::Stack(const Stack& other):size{ other.size }, storage{ new int[other.size] }

{

size = other.size;

storage = new int[size];

capacity = other.capacity;

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

storage[i] = other.storage[i];

}

return;

}

Stack::~Stack(){

delete[] storage;

}

int Stack::sizeReturn() const{

return size;

}

int Stack::get(int n) const {

return storage[n];

}

int Stack::capacityReturn() const{

return capacity;

}

void Stack::make\_space(int z\_alloc)

{

int index = size;

size = 0;

int size\_counter=0;

if (z\_alloc <= capacity) return; // never decrease allocation

int\* p = new int[z\_alloc]; // allocate new space

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

p[i] = storage[i]; // copy old elements

size\_counter++;

}

delete[] storage; // deallocate old space

storage = p; // storage is set to the needed size

capacity = z\_alloc;

size = size\_counter;

return;

}

Stack& Stack::push(int a){

if (capacity == 0){

make\_space(100);

} // make space for 100 more elements when add to end

else if (size == capacity){

make\_space(2 \* capacity);

} // get more space

storage[size] = a; // add at end of Number\_Store

++size; // increase size (sz is the number of elements)

return \*this;

}

int Stack::top(){

int index = size-1;

int value = storage[index];

return value;

}

void Stack::pop(){

int index = size-1;

size = index;

}

bool Stack::isEmpty(){

if (size == 0){

return true;

}

else{

return false;

}

}

bool Stack::isFull(){

if (size == capacity){

return true;

}

else{

return false;

}

}

// Return true if ==, false otherwise

// Use ONLY stack methods, leave the stacks unchanged.

bool operator==(const Stack& lhs, const Stack& rhs) {

bool test = false;

if (lhs.sizeReturn()==rhs.sizeReturn()&&lhs.capacityReturn()==rhs.capacityReturn()){

for (int i = 0; i < rhs.sizeReturn(); ++i){

if (lhs.get(i) == rhs.get(i)){

test = true;

}

else{

test = false;

return test;

}

}

}

return test;

}

// Print the values on the stack, using ONLY

// stack methods, leave the stack unchanged.

ostream& operator<<(ostream&ost, const Stack& s) {

int const sizeHolder = s.sizeReturn();

ost << "(";

for (int i = 0; i < s.sizeReturn(); ++i) {

ost << s.get(i);

if (i < s.sizeReturn() - 1){

ost << ',';

}

}

ost << ")";

return ost;

}

/\* End of Stack Implementation Code \*/

void testStack(void);

int main() {

cout << "Joe Aguilera" << endl;

testStack();

return 0;

}

// Test your Stack methods below.

// Add additional tests to the following code

void testStack(void) {

cout << "Stack Testing..." << endl;

Stack s(10);

//

s.push(3).push(7).push(1).push(9);

s.pop(); s.pop(); s.push(8);s.pop(); s.pop();

cout << s.top() << endl;

}