**CSCE 146 Exam 1 Review**

Part 1 – Java Review

This part went over the basics of Java, and the focus was mostly on arrays. There will be no questions that involve reading from and writing to files.

Short Answer

Given this programming snippet, what will print out at the end?

**final** **int** a = 10;

**for**(**int** i=0;i<a;i+=2)

{

**for**(**int** j=i;j<a;j++)

{

System.*out*.print("\*");

}

System.*out*.println();

}

\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*

\*\*

Will this piece of code give an error? If so why? If not what will it print out?

**int**[] a = {1,2,3,4,5,6,7,8,9,10};

**for**(**int** i=1;i<5;i++)

{

System.*out*.println(a[i]);

System.*out*.println(a[i\*2]);

}

2

3

3

5

4

7

5

9

Programming

Write a method that takes in an array and will print out any values in array that is divisible by 7

**public** **void** printOut7s(**int**[] a)

{

//PUT YOUR CODE HERE

**for**(**int** i=0;i<a.length;i++)

{

**if**(a[i]%7 == 0)

{

System.*out*.println(a[i]);

}

}

}

Write a static method that takes two vectors and subtracts them and then returns the results. Also both arrays should be the same size or else return null.

**public** **static** **int**[] subtractVect(**int**[] vect1, **int**[] vect2)

{

**if**(vect1.length != vect2.length)

**return** **null**;

**int**[] retVect = **new** **int**[vect1.length];

**for**(**int** i=0;i<retVect.length;i++)

{

retVect[i] = vect1[i]-vect2[i];

}

**return** retVect;

}

Part 2 – Linked Lists

Building on arrays, we introduce a new way to group together like information. Using a nodes which contains data and links we created a resizable collection of data.

Short Answer

What’s one of the greatest advantages of using a linked list over an array?

An array is fixed in size, but a linked list can grow and shrink.

Given the linked list below, redraw the links after a new node is inserted after the node that contains 4. Indicate new links by arrows and mark out the broken links. Also assume the new node contains the value 8

3

12

4

20

Programming

Write a method that inserts a new node after another node with a given value is found. If a node with that value doesn’t exist put the new node at the end. The method should not return any values and should take in both the value being search for of generic type T and also the value of newly created node of generic type T. Also assume that nodes have both data and link values as seen in given code description.

**public** **void** insertAfter(T findValue, T nodeValue)

{

LinkNode newNode = **new** LinkNode();

newNode.data = nodeValue;

LinkNode currNode = head;

**while**(currNode != **null**)

{

**if**(currNode.link == **null**)

{

currNode.link = newNode;

**break**;

}

**else** **if**(currNode.data.equals(findValue))

{

newNode.link = currNode.link;

currNode.link = newNode;

**break**;

}

currNode = currNode.link;

}

}

Write a program where a user populated Array List of positive whole numbers is averaged. First the user can enter as many numbers as they like as long as they’re positive. The user indicates they have entered all the numbers they want by then entering a negative number. Once the first negative number is found the Array List of whole numbers is averaged and the result is printed out.

**public** **static** **void** main(String[] args) {

ArrayList<Integer> intArr = **new** ArrayList<Integer>();

Scanner keyboard = **new** Scanner(System.*in*);

**while**(**true**)

{

System.*out*.println("Enter a positive number to add to the array list. "

+ "Enter a negative to quit");

**int** input = keyboard.nextInt();

**if**(input < 0)

{

**break**;

}

intArr.add(input);

}

**int** result = 0;

**for**(**int** a : intArr)

{

result += a;

}

result /= intArr.size();

System.*out*.println("The average is "+result);

}

Part 3 – Queues

Grouping data on the principle first in first out (FIFO) we learned how add data (enqueue) and remove data (dequeue) in these elegant data structures. It is important to know how the operations enqueue, dequeue, and peek all work using either arrays or linked lists.

Given this programming snippet, what will print out at the end? Add is the same as enqueuer and remove is the same as dequeuer.

Queue<Integer> intQueue = **new** LinkedList<Integer>();

**for**(**int** i=0;i<21;i+=3)

{

intQueue.add(i);

}

**while**(intQueue.size()>3)

{

intQueue.remove();

}

**for**(**int** i : intQueue)

{

System.*out*.println(i);

}

12  
15  
18

Given and initial queue these lines of pseudo-code, fill in how the queue looks after each step.

1. Dequeue 3 times
2. Enqueue 7
3. Enqueue 12
4. Enqueue 23
5. Dequeue 2 times

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |  |
| 25 | 74 | 84 | 3 | 23 | 85 | 96 |  |

1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |  |
| 3 | 23 | 85 | 96 |  |  |  |  |

2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |  |
| 3 | 23 | 85 | 96 | 7 |  |  |  |

3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |  |
| 3 | 23 | 85 | 96 | 7 | 12 |  |  |

4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |  |
| 3 | 23 | 85 | 96 | 7 | 12 | 23 |  |

5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Head |  |  |  |  |  |  |  |
| 85 | 96 | 7 | 12 | 23 |  |  |  |

Programming

With the given linked list in this exam, write a method that enqueues a new node with a double value given in the parameter. Make sure to change the links properly to ensure that the data structure remains a queue.

**public** **void** enqueue(double data)

{

ListNode newNode = **new** ListNode(data,**null**);

**if**(head == **null**)

{

head = newNode;

tail = head;

**return**;

}

**this**.tail.link = newNode;

**this**.tail = newNode;

}

Part 4 – Stacks

This data structure used the last in first out (LIFO) principle in order to add (push) and remove (pop) data. It’s important to know how the operations push, pop, and peek work using either arrays or linked lists.

Short Answer

Given this programming snippet what will print out?

**for**(**double** i=0;i<100.0;i+=13.0)

{

dubStack.push(i);

}

**for**(**int** i=0;i<2;i++)

{

dubStack.pop();

}

dubStack.push(50.0);

**while**(dubStack.isEmpty()==**false**)

{

System.*out*.println(dubStack.pop());

}

50  
65  
52  
39  
26  
13  
0

Given this pseudo-code what will the stack look like at each step.

1. Push 23
2. Pop 3 times
3. Push 18
4. Push 22
5. Pop

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Head |  |
| 25 | 74 | 84 | 3 | 23 | 85 | 96 |  |

1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | Head |
| 25 | 74 | 84 | 3 | 23 | 85 | 96 | 23 |

2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Head |  |  |  |
| 25 | 74 | 84 | 3 | 23 |  |  |  |

3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Head |  |  |
| 25 | 74 | 84 | 3 | 23 | 18 |  |  |

4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | Head |  |
| 25 | 74 | 84 | 3 | 23 | 18 | 22 |  |

5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Head |  |  |
| 25 | 74 | 84 | 3 | 23 | 18 |  |  |

Programming

With the given linked list structure write a method for pop for a stack of type double.

**public** double pop()

{

if(head != null)

{

**double** retData = head.data;

head = head.link;

**return** retData;

}

**return** 0.0;

}

Part 5 – Recursion

This mind bending way to create algorithms relies on methods that call themselves. While having more overhead than an iterative loop, recursion enables programmers to create more elegant code.

Short Answer

Given this recursive method what would the result print out?

**public** **static** **void** main(String[] args) {

**int**[] a = {1,2,3,4,5,6,7,8,9,10};

*splitPrint*(a);

}

**public** **static** **void** splitPrint(**int**[] a)

{

System.*out*.println(a[a.length-1]);

**int** halfSize = a.length/2;

**if**(halfSize > 0)

{

**int**[] newArr = **new** **int**[a.length/2];

**for**(**int** i=0;i<newArr.length;i++)

{

newArr[i] = a[i];

}

*splitPrint*(newArr);

}

}

10  
5  
2  
1

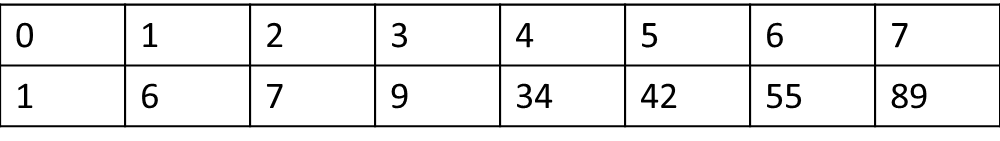
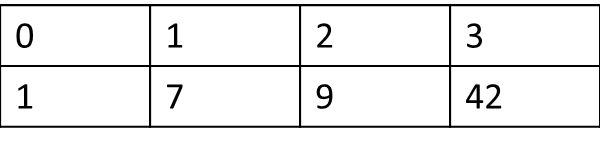
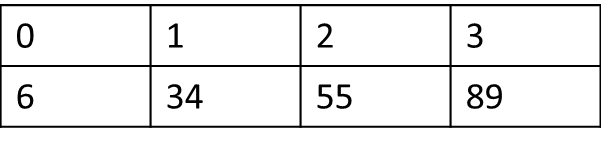
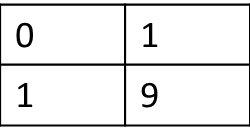
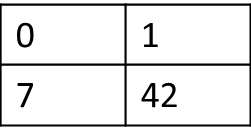
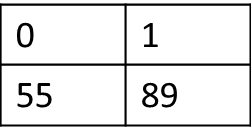
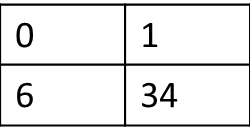
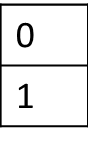
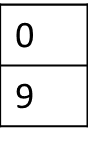
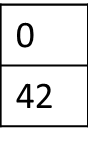
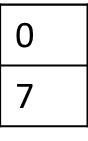
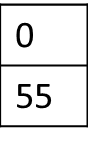
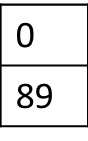
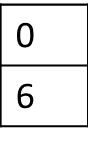
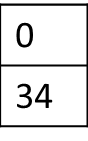
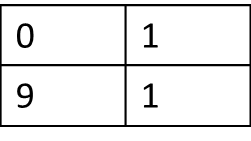
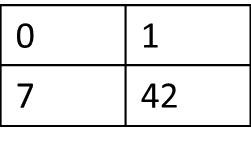
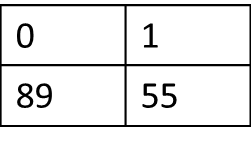
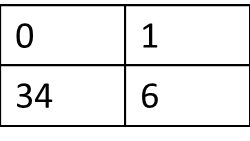
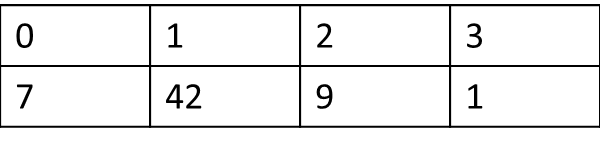
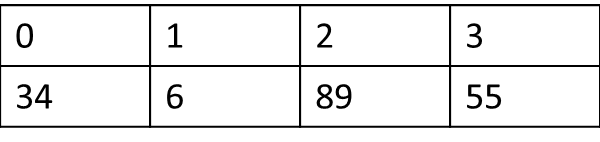
Part 6 – Searching / Sorting / Asymptotics

If given an array of 2048 items, how many times will binary search need to be recursively called to determine a value is not in that array?

11 because log2(2048) = lg(2048) = 11.

Given the following array simulate merge sort.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Value | 34 | 6 | 89 | 55 | 7 | 42 | 9 | 1 |



Order these Big Oh complexities by which ones grow the fastest (run slower) to ones that grow the slowest (run faster) as n approaches infinity.

O(n2), O(n!), O(2n), O(n), O(lg(n)), O(nlg(n)), O(), O(n3), O(1)

O(n!), O(2n), O(n3), O(n2), O(nlg(n)), O(n), O(), O(lg(n)), O(1),

Generic Linked List for reference

**public** **class** LinkedList<T> {

**public** **class** LinkNode

{

T data;

LinkNode link;

**public** LinkNode(T aData, LinkNode aLink)

{

**this**.data = aData;

**this**.link = aLink;

}

}

LinkNode head;

LinkNode tail;

}