CSE 205 Review Session #3

Questions: [kvlopez1@asu.edu](mailto:kvlopez1@asu.edu)

**Quick Sort**

* T(n) = O(n log n) \*Same as Merge Sort
* Pivot value = first number of array.
* Purpose of i = keeps moving to the right until finding the first number >= pivot value.
* Purpose of j = keeps moving to the left until finding the first number <= pivot value.

1. Given the following integer array, show the resulting sub arrays after the first partition using Quick Sort. Identify the pivot value.

[13, 15, 5, 16, 6, 7, 11, 3, 18, 8]

**Recursion**

1. Show the output of the following program.

**public** **class** CSE205ReviewSession3

{

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

{

**int**[] testArray = {3,0,2,0,-5,0,15,-8,0,9,4,-1,6,0,58,-9,7};

**int** result = *methodX*(testArray, 0, testArray.length-1);

System.***out***.println("Result is: " + result);

}

**public** **static** **int** methodX(**int**[] intArray, **int** startIndex, **int** endIndex)

{

**if**(startIndex == endIndex)

{

**if**(intArray[startIndex] == 0)

**return** 1;

**else**

**return** 0;

}

**else**

{

**int** midIndex = (startIndex + endIndex)/2;

**int** x = *methodX*(intArray, startIndex, midIndex);//4

**int** y = *methodX*(intArray, midIndex+1, endIndex);//1

**return** x+y;

}

}

}

**OUTPUT:**

Result is: 5

**Linked List**

* Data Structures:
  1. Static
     + Fixed size
  2. Dynamic
     + Grows and shrinks as required
* Two primary data structures
  1. Array = array list, stack & queue
  2. LinkedList = stack & queue, binary tree
* Comparison of operations of ArrayList vs. LinkedList

|  |  |  |
| --- | --- | --- |
|  | ArrayList | LinkedList |
| Access | O(1) | O(n) |
| Add | O(n) | O(1) |
| Remove | O(n) | O(1) |

1. Show the output of the following program.

**public** **class** LinkedListReview

{

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

{

LinkedList list = **new** LinkedList();

ListIterator iterator = list.listIterator();

iterator.add("C");

iterator.add("S");

iterator.add("E");

*printList*(list);

iterator = list.listIterator();

iterator.next();

iterator.next();

iterator.add("K");

*printList*(list);

iterator = list.listIterator();

iterator.next();

iterator.remove();

iterator.next();

iterator.remove();

*printList*(list);

}

**public** **static** **void** printList(LinkedList list)

{

ListIterator iterator = list.listIterator();

String result = "{";

**while**(iterator.hasNext())

{

result += iterator.next() + "";

}

result += "}";

System.***out***.println(result);

}

}

**Output:**

{CSE}

{CSKE}

{KE}

1. **Given the following linked list and class Node write the following codes.**

Current

First

3

-23

-5

8

……..

class Node

{

int data;

Node next;

}

1. Write the code to insert a new Node with value 18 between Node 3 and Node 8.

**Answer:**

Node newNode = new Node();

newNode.data = 18;

newNode.next = current.next;

current.next = newNode;

1. Assuming the name of the previous linked list is reviewList and an iterator was already created, write the code to output the number of negative nodes in it.

**Answer:**

for(int i =0; i <= size()-1; i++)

{

int result = 0;

if(i.data < 0)

result += 1;

else

result =0;

iterator.next();

return result;

}

**Stacks**

1. Given the following program, write the output. (pretend StackInArrayList is a given class)

**public** **class** StacksReview

{

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

{

StackInArrayList myStack = **new** StackInArrayList();

myStack.push("United States");

myStack.push(**new** Integer(8));

myStack.push("Canada");

myStack.push(**new** Integer(7));

myStack.push("Germany");

myStack.push(**new** Integer(6));

System.***out***.println(myStack);

myStack.pop();

myStack.pop();

System.***out***.println(myStack);

myStack.push("Mexico");

myStack.push(**new** Integer(5));

System.***out***.println(myStack);

}

}

**Output:**

United States 8 Canada 7 Germany 6

United States 8 Canada 7

United States 8 Canada 7 Mexico 5

**Queues**

1. Given the following program, write the output. (pretend Queue class is given).

**public** **class** QueueReview

{

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

{

Queue myQueue = **new** Queue();

myQueue.enqueue("Black");

myQueue.enqueue(**new** Integer(0));

myQueue.enqueue("Blue");

myQueue.enqueue(**new** Integer(1));

System.***out***.println(myQueue);

myQueue.enqueue("Orange");

myQueue.enqueue(**new** Integer(2));

myQueue.dequeue();

myQueue.dequeue();

System.***out***.println(myQueue);

myQueue.enqueue("Yellow");

myQueue.enqueue(**new** Integer(3));

myQueue.dequeue();

System.***out***.println(myQueue);

}

}

**Output:**

Black 0 Blue 1

Blue 1 Orange 2

1 Orange 2 Yellow 3

**Binary Trees**

* Hierarchy data structure (nonlinear)
* Root = first node
* Left child = node pointed by the left pointer of the root
* Right child = node pointed by the right pointer of the root
* Leaf: any node without children
* Binary Tree Traversal:
  1. Is the process of visiting nodes in the tree exactly once
  2. Recursive procedure
  3. For Linked List it’s iterators
* Pre-order: root, left child, right child
* In-order: left child, root, right child
* Post-order: left, right, root
* Depth-first: exactly the same as pre-order (root, left, right)
* Breath-first: according to level from 0 … n. within each level traverse left to right
* Full binary tree: Is a binary tree in which, except for the leaf node, every node has exactly two children.
* Complete binary tree:
  1. Is a full binary tree
  2. Except for the last level, every level is completely filled and all nodes are filled as far left as possible

1. Draw the binary search tree given the following values.

23 43 7 18 6 81 14 20 67 19 3

**Heaps**

* **O(log n)**
* Useful for designing efficient sorting algorithms and priority queues.
* Max-heap: it’s an almost complete binary tree. Every node in the tree is >= than its two children.
* Min-heap: it’s an almost complete binary tree. Every node in the tree is <= than its two children