1. What’s wrong with this definition:

**Arrays arrays = new Arrays();**

**الحل:**

Arrays.sort(numbers);

1. Write and test this method:

void reverse(int[] a)

// reverses the elements of a[]

الحل:

void reverse(int[] a) {

int start = 0;

int end = a.length - 1;

while (start < end) {

// قم بتبديل قيمتين في المصفوفة

int temp = a[start];

a[start] = a[end];

a[end] = temp;

// انتقل إلى العنصر التالي

start++;

end--;

}

}

1. If linked lists are so much better than arrays, why are arrays used at all?

الوصول العشوائي  
الأداء  
الاحتفاظ بالتسلسل  
تحديد الحجم الثابت  
دعم للعمليات الشائعه

1. **Mark the following statements as true or false.**
   1. In a linked list, the order of the elements is determined by the order in which the nodes were created to store the elements. True
   2. In a linked list, memory allocated for the nodes is sequential. False
   3. A single linked list can be traversed in either direction. False
   4. In a linked list, nodes are always inserted either at the beginning or the end because a linked link is not a random access data structure. True
   5. The head pointer of a linked list cannot be used to traverse the list. false

**Consider the linked list shown in Figure. Assume that the nodes are in the usual Element-Next form. Use this list to answer Exercises 5 through 8. If necessary, declare additional variables. (Assume that list, p, s, A, and B are references of type Node.)**



Linked list for Exercises 2–7

1. What is the output of each of the following java statements?
   1. System.out.println( list.getElement());

The output would depend on the implementation of the getElement() method and the content of the current node referenced by list.

* 1. System.out.println( A. getElement());

The output would depend on the implementation of the getElement() method and the content of the node referenced by A

* 1. System.out.println( B.getNext().getElement());

The output would depend on the implementation of the getNext() and getElement() methods, as well as the content of the node referenced by B and the node that follows it

* 1. System.out.println( list.getNext().getNext().getElement());

The output would depend on the implementation of the getNext() and getElement() methods, as well as the content of the nodes referenced by list and the nodes that follow them.

1. What is the value of each of the following relational expressions?
   1. list. getElement() >= 18

True

* 1. list.getNext() == A

False

* 1. A.getNext().getElement() == 16
  2. True
  3. B.getNext() == (NULL)

True

* 1. list. getElement() == 18

True

1. Write java Fragment code to do the following:
   * + 1. Make A point to the node containing element 23.

A = new Node(23);

* + - 1. Make list point to the node containing 16.

list = new Node(16);

* + - 1. Make B point to the last node in the list

B = list; while(B.getNext() != null) { B = B.getNext(); }

* + - 1. Make list point to an empty list.

list = null;

* + - 1. Set the value of the node containing 25 to 35.

Node currentNode = list; while(currentNode != null) { if(currentNode.getElement() == 25) { currentNode.setElement(35); break; } currentNode = currentNode.getNext(); }

* + - 1. Create and insert the node with element 10 after the node pointed by A.

Node newNode = new Node(10); newNode.setNext(A.getNext()); A.setNext(newNode);

* + - 1. Delete the node with element 23. Also, deallocate the memory occupied by this node.

Node previousNode = null;

Node currentNode = list;

while(currentNode != null) {

if(currentNode.getElement() == 23) {

if(previousNode != null) {

previousNode.setNext(currentNode.getNext()); }

else { list = currentNode.getNext(); }

currentNode.setNext(null); break; }

previousNode = currentNode; currentNode = currentNode.getNext();

}

What is the output of the following java code?

p = list;

while (p != NULL){

System.out.println( p.getElement());

p = p.getNext(); }

سيتم طباعة قيمة كل عنصر في القائمة المرتبطة من خلال الاستدلال على العقدة الأولى والانتقال من عقدة إلى أخرى حتى نهاية القائمة.

1. Show what is produced by the following java code. Assume the node is in the usual **getElement()-getNext()** form with the info of type int. (**list** and **p** are pointers of type **node<E>()**.)
   * + 1. list = new node<E>();

list.setElement(10);

p = new node<E>();

p. setElement(13);

p.setNext(null);

list.setNext(p);

p = new node<E>(18, list.getNext());

list.setNext(p);

System.out.println(list.getElement());

System.out.println(p.getElement());

p = p.getNext();

System.out.println(p.getElement());

تم إنشاء قائمة مرتبطة جديدة وتعيين القيمة 10 لعنصر العقدة الأولى.

تم إنشاء عقدة جديدة وتعيين القيمة 13 لعنصرها وإعدادها لتكون العقدة التالية بعد العقدة الأولى.

تم إنشاء عقدة جديدة وتعيين القيمة 18 لعنصرها، وإعدادها لتكون العقدة التالية بعد العقدة الأولى.

يتم طباعة قيمة العقدة الأولى (10).

يتم طباعة قيمة العقدة الثالثة (18).

يتم تحديث المرجع p ليشير إلى العقدة التالية بعد العقدة الثالثة.

يتم طباعة قيمة العقدة التي يشير إليها p (ليست موجودة)

* + - 1. list = new node<E>();

list.setElement(20);

p = new node<E>();

p. setElement(28);

p.setNext(NULL);

list. setNext(p);

p = new node<E>();

p.setElement(30);

p.setNext(list);

list = p;

p = new node<E>();

p.setElement(42);

p.setNext(list.getNext());

list.setNext(p);

p = List;

while (p != NULL)

{

System.out.println( p.getElement());

p = p.getNext(); }

تم إنشاء قائمة مرتبطة جديدة وتعيين القيمة 20 لعنصر العقدة الأولى.

تم إنشاء عقدة جديدة وتعيين القيمة 28 لعنصرها وإعدادها لتكون العقدة التالية بعد العقدة الأولى.

تم إنشاء عقدة جديدة وتعيين القيمة 30 لعنصرها وإعدادها لتكون العقدة التالية قبل العقدة الأولى.

تم تحديث المرجع list ليشير إلى العقدة الجديدة.

تم إنشاء عقدة جديدة وتعيين القيمة 42 لعنصرها وإعدادها لتكون العقدة التالية بعد العقدة التي تأتي بعدها العقدة الثانية.

يتم طباعة قيمة العقدة الثالثة (30) والعقدة الخامسة (42) بالترتيب

1. **Consider the following java statements. (The class SingleLinkedList is as defined in the lectures).**

SingleLinkedList<int> list;

list.addFirst(15);

list.addLast(28);

list.addFirst(30);

list.addFirst(2);

list.addLast(45);

list.addFirst(38);

list.addLast(25);

list.removeNode(30);

list.addFirst(18);

list.removeNode(28);

list.removeNode(12);

list.print();

What is the output of this program segment?

38

2

15

45

25

18

1. For the following doubly linked list figure, show by java code how to insert value (info) 20 between values 15 & 24?



1. Write and test this method for **SingleLinkedList class** :

**Public int sum(Node<int> list)**

// returns: the sum of the integers in the specified list;

For example, if list is {25, 45, 65, 85}, then sum(list) will return 220.

public int sum(Node<Integer> list) {

int sum = 0;

Node<Integer> current = list;

while (current != null) {

sum += current.getElement();

current = current.getNext();

}

return sum;

}

1. Write and test this method for **DoublyLinkedList class**:

**Public E removeLast(Node<E> list)**

// precondition: the specified list has at least two nodes;

// postcondition: the last node in the list has been deleted;

For example, if list is {22, 44, 66, 88}, then removeLast(list) will change it to {22, 44, 66}.

public E removeLast(Node<E> list) {

Node<E> current = list;

while (current.getNext() != null) {

current = current.getNext();

}

Node<E> previous = current.getPrevious();

previous.setNext(null);

current.setPrevious(null);

return current.getElement();

}

1. Write and test this method for **DoublyLinkedList class**:

**Public E removeLast(Node<E> list)**

// precondition: the specified list has at least two nodes;

// postcondition: the last node in the list has been deleted;

For example, if list is {22, 44, 66, 88}, then removeLast(list) will change it to {22, 44, 66}.

public E removeLast(Node<E> list) {

if (list == null || list.getNext() == null) {

return null;

}

Node<E> current = list;

while (current.getNext().getNext() != null) {

current = current.getNext();

}

Node<E> lastNode = current.getNext();

current.setNext(null);

lastNode.setPrevious(null);

return lastNode.getElement();

}

1. Write and test this method for **SingleLinkedList class**:

**Public void append(Node<E> list1, Node<E> list2)**

// precondition: list1 has at least one node;

// postcondition: list1 has list2 appended to it;

For example, if list1 is {22, 33, 44, 55} and list2 is {66, 77, 88, 99}, then append(list1, list2) will change list1 to {22, 33, 44, 55, 44, 55, 66, 77, 88}. Note that no new nodes are created by this method.

public void append(Node<E> list1, Node<E> list2) {

if (list1 == null) {

return;

}

Node<E> current = list1;

while (current.getNext() != null) {

current = current.getNext();

}

current.setNext(list2);

}

1. Write and test this method for **SingleLinkedList class**:

**Public Node<E> concat(Node<E> list1, Node<E> list2)**

// returns: a new list that contains a copy of list1, followed by a copy of list2;

For example, if list1 is {22, 33, 44, 55} and list2 is {66, 77, 88, 99}, then concat(list1, list2) will return the new list {22, 33, 44, 55, 44, 55, 66, 77, 88}. Note that the three lists should be completely independent of each other. Changing one list should have no effect upon the others.

public Node<E> concat(Node<E> list1, Node<E> list2) {

if (list1 == null) {

return list2;

}

Node<E> newList = new Node<>(list1.getElement());

Node<E> current = newList;

list1 = list1.getNext();

while (list1 != null) {

current.setNext(new Node<>(list1.getElement()));

current = current.getNext();

list1 = list1.getNext();

}

while (list2 != null) {

current.setNext(new Node<>(list2.getElement()));

current = current.getNext();

list2 = list2.getNext();

}

return newList;

}

1. Write and test this method for **DoublyLinkedList class**:

**Public void swap(Node<E> list, int i, int j)**

// swaps the ith element with the jth element;

For example, if list is {22, 33, 44, 55, 66, 77, 88, 99}, then swap(list, 2, 5) will change list to {22, 33, 77, 55, 66, 44, 88, 99}.

public void swap(Node<E> list, int i, int j) {

if (list == null || i == j) {

return;

}

Node<E> node1 = getNodeAtIndex(list, i);

Node<E> node2 = getNodeAtIndex(list, j);

if (node1 == null || node2 == null) {

return;

}

E temp = node1.getElement();

node1.setElement(node2.getElement());

node2.setElement(temp);

}

private Node<E> getNodeAtIndex(Node<E> list, int index) {

int count = 0;

Node<E> current = list;

while (current != null && count < index) {

current = current.getNext();

count++;

}

return current;

}

1. Describe in detail(without java code) an algorithm for reversing a singly linked list *L* using only a constant amount of additional space.

قم بتهيئة ثلاثة مؤشرات

: prev (المؤشر السابق)، current (المؤشر الحالي)، وnext (المؤشر التالي).

قم بتحديث المؤشرات حتى يتم تحويل العقدة الأولى في القائمة إلى آخر عقدة القائمة، وذلك عن طريق التحرك بشكل تتابعي عبر القائمة وتحديث المؤشرات في كل تكرار.

استمر في تحديث المؤشرات حتى يتم تحويل العقدة الأخيرة في القائمة إلى العقدة الأولى، وبالتالي عكس القائمة بالكامل.

1. Implement the equals( ) method for the DoublyLinkedList class.

**@Override**

**public boolean equals(Object obj) {**

**if (this == obj) {**

**return true;**

**}**

**if (obj == null || getClass() != obj.getClass()) {**

**return false;**

**}**

**DoublyLinkedList<E> otherList = (DoublyLinkedList<E>) obj;**

**if (size != otherList.size) {**

**return false;**

**}**

**Node<E> current1 = head;**

**Node<E> current2 = otherList.head;**

**while (current1 != null) {**

**if (!current1.getElement().equals(current2.getElement())) {**

**return false;**

**}**

**current1 = current1.getNext();**

**current2 = current2.getNext();**

**}**

**return true;**

**}**

1. Implement the rotate() methode in CircularLinkedList class.

**التحقق مما إذا كانت القائمة فارغة أو تحتوي على عنصر واحد فقط.**

**إعادة توجيه رأس القائمة ليصبح العقدة التالية للرأس الحالي.**

**اكتمال عملية التدوير.**

1. Implement the addFirst() method in CircularLinkedList class.

**public void addFirst(E element) {**

**Node<E> newNode = new Node<>(element);**

**if (isEmpty()) {**

**newNode.setNext(newNode);**

**head = newNode;**

**} else {**

**newNode.setNext(head);**

**Node<E> lastNode = head;**

**while (lastNode.getNext() != head) {**

**lastNode = lastNode.getNext();**

**}**

**lastNode.setNext(newNode);**

**head = newNode;**

**}**

**size++;**

**}**