1. Which of the JCF interfaces would be the most useful if we want to store a collection of students enrolled in COMP2402 so that we can quickly check if a student is enrolled in COMP2402?

(a) Collection
(b) Set
(c) SortedSet
(d) Map
(e) SortedMap

2. What if we also want to be able to quickly output a list of students, sorted by (lastname, firstname)?

(a) Collection
(b) Set
(c) SortedSet
(d) Map
(e) SortedMap

3. What if, in addition, we also want to store some auxiliary information (e.g., a mark) with each student?

(a) Collection
(b) Set
(c) SortedSet
(d) Map
(e) SortedMap

4. A Bag is like a Set except that equal elements can be stored more than once. Which of the following is best suited to implement a Bag<T>?

(a) Set<T>
(b) Map<T,Integer>
(c) Map<T,List<T>>
(d) SortedSet<T>
(e) Either (b) or (c) depending on what behaviour we want if we add two elements that are equal but not identical.

5. The running time of the methods get(i) and remove(i) for an ArrayList are

(a) O(1) and O(1), respectively
(b) O(1 + i) and O(1 + i), respectively
(c) O(1) and O(1 + i), respectively
(d) O(1 + i) and O(1 + size() − i), respectively
(e) O(1) and O(1 + size() − i), respectively

6. The running time of the methods get(i) and remove(i) for a LinkedList are
(a) $O(1+i)$ and $O(1+i)$, respectively
(b) $O(1)$ and $O(1+\text{size()}−i)$, respectively
(c) $O(1+\text{size()}−i)$ and $O(1)$, respectively
(d) $O(1+\min\{i,\text{size()−i}\})$ and $O(1+\min\{i,\text{size()−i}\})$, respectively
(e) $O(1)$ and $O(1+\text{size()}−i)$, respectively

7. public static void frontGets(List<Integer> l, int n) {
   for (int i = 0; i < n; i++) {
      l.get(0);
   }
}

The above method is

(a) much faster when $l$ is an ArrayList
(b) much faster when $l$ is a LinkedList
(c) about the same speed independent of whether $l$ is an ArrayList or a LinkedList

8. public static void randomGets(List<Integer> l, int n) {
   Random gen = new Random();
   for (int i = 0; i < n; i++) {
      l.get(gen.nextInt(l.size()));
   }
}

The above method is

(a) much faster when $l$ is an ArrayList
(b) much faster when $l$ is a LinkedList
(c) about the same speed independent of whether $l$ is an ArrayList or a LinkedList

9. public static void insertAtBack(List<Integer> l, int n) {
   for (int i = 0; i < n; i++) {
      l.add(new Integer(i));
   }
}

The above method is

(a) much faster when $l$ is an ArrayList
(b) much faster when $l$ is a LinkedList
(c) about the same speed independent of whether $l$ is an ArrayList or a LinkedList

10. public static void insertAtFront(List<Integer> l, int n) {
    for (int i = 0; i < n; i++) {
        l.add(0, new Integer(i));
    }
}

The above method is

(a) much faster when $l$ is an ArrayList
(b) much faster when $l$ is a LinkedList
11. public static void insertInMiddle(List<Integer> l, int n) {
    for (int i = 0; i < n; i++) {
        l.add(new Integer(i));
    }
    for (int i = 0; i < n; i++) {
        l.add(n/2+i, new Integer(i));
    }
}

The above method is
(a) much faster when l is an ArrayList
(b) much faster when l is a LinkedList
(c) about the same speed independent of whether l is an ArrayList or a LinkedList

12. public static void insertInMiddle2(List<Integer> l, int n) {
    for (int i = 0; i < n; i++) {
        l.add(new Integer(i));
    }
    ListIterator<Integer> li = l.listIterator(n/2);
    for (int i = 0; i < n; i++) {
        li.add(new Integer(i));
    }
}

The above method is
(a) much faster when l is an ArrayList
(b) much faster when l is a LinkedList
(c) about the same speed independent of whether l is an ArrayList or a LinkedList

13. Recall that an ArrayStack stores n elements in a backing array a at locations a[0],...,a[n-1]:

    public class ArrayStack<T> extends AbstractList<T> {
        T[] a;
        int n;
        ...
    }

Also recall that, immediately after the backing array a is resized by grow() or shrink it has a.length = 2n.
When adding an element, the ArrayStack grows the backing array a if it is full, i.e, if a.length = n.
If are currently about to grow the backing array a, what can you say about the number of add() and remove() operations (as a function of the current value of n) since the last time the ArrayStack was resized?

(a) At least n/2 add() operations have occurred since then
(b) At least 2n/3 add() operations have occurred since then
(c) At least n/2 remove() operations have occurred since then
(d) At least 2n/3 remove() operations have occurred since then
(e) We can not bound either the number of add() nor remove() operations

14. Recall that we shrink the backing array a when 3n < a.length. If we are currently about to shrink the backing array a, what can you say about the number of add() and remove() operations since the last time the ArrayStack was resized?

(a) At least n/2 add() operations have occurred since then
(b) At least 2n/3 add() operations have occurred since then
(c) At least n/2 remove() operations have occurred since then
(d) At least 2n/3 remove() operations have occurred since then
(e) We can not bound either the number of add() nor remove() operations

15. From the previous two questions, what can you conclude about the total number of elements copied by grow() and shrink() if we start with an empty ArrayStack and perform m add() and remove operations.

(a) At most m elements are copied by grow() and shrink()
(b) At most 2m elements are copied by grow() and shrink()
(c) At least m elements are copied by grow() and shrink()
(d) At least 2m elements are copied by grow() and shrink()
(e) We can not bound the number of elements copied by grow() and shrink()

16. Recall that an ArrayDeque stores n elements at locations a[j], a[(j+1)%a.length], ..., a[(j+n-1)%a.length]:

public class ArrayDeque<T> extends AbstractList<T> {
    T[] a;
    int j;
    int n;
    ...
}

What is the amortized running time of the add(i,x) and remove(i) operations?

(a) O(1 + i)
(b) O(1 + |i - n/2|)
(c) O(1 + n - i)
(d) O(1 + min{i, n - i})
(e) O(1 + min{i - n, n - i})

17. If m = 2^{10} then the binary representations of m and m - 1 are

(a) 10000000000 and 09999999999, respectively
(b) 10000000000 and 01111111111, respectively
(c) 01111111111 and 10000000000, respectively
(d) 10101010001 and 00101010111, respectively
(e) 10000000000 and 11111111111, respectively
18. From the previous question, if the binary representation of \( x \) is 000111000111000111000101010101010101, then the binary representation of \( x \bmod m \) is

(a) 000111000111000111000101010101010101

(b) 000000000000000000000000000000000000

(c) 00011100011100011100010101000000000000

(d) 000000000000000000000000000000000000

(e) 000000000000000000000000000000000000

19. Recall that a DualArrayDeque implements the List interface using two ArrayStacks:

```java
public class DualArrayDeque<T> extends AbstractList<T> {
    ArrayStack<T> front;
    ArrayStack<T> back;
    ...
}
```

In order to implement \( \text{get}(i) \) we need to get it from the ArrayStack, front or back. We can express this as

(a) \( \text{front.get}(i) \)

(b) \( \text{front.get(front.size()-i-1)} \)

(c) \( \text{back.get(i-front.size())} \)

(d) Either (b) or (c) depending on the value of \( i \) and \( \text{front.size()} \)

(e) Either (a) or (c) depending on the value of \( i \) and \( \text{front.size()} \)

20. Recall that a RootishArrayStack stores a list in a sequence of arrays (blocks) of sizes 1, 2, 3, 4,....

```java
public class RootishArrayStack<T> extends AbstractList<T> {
    List<T[]> blocks;
    int n;
    ...
}
```

21. If a RootishArrayStack has 10 blocks (so \( \text{b.size()} = 10 \)), then how many elements can it store?

(a) 90

(b) 110

(c) 45

(d) \( 55 = 10 \cdot 11/2 \)

(e) none of the above

22. In a RootishArrayStack, a call to \( \text{get}(13) \) will return

(a) \( \text{blocks.get(0)}[13] \)

(b) \( \text{blocks.get(13)}[0] \)

(c) \( \text{blocks.get(4)}[3] \)

(d) \( \text{blocks.get(3)}[4] \)

(e) \( \text{blocks.get(5)}[4] \)
23. Recall our implementation of a singly-linked list (SLList):

```java
protected class Node {
    T x;
    Node next;
}
public class SLList<T> extends AbstractList<T> {
    Node head;
    Node tail;
    int n;
    ...
}
```

Consider how to implement a Queue as an SLList. When we enqueue \((\text{add}(x))\) an element, where does it go? When we dequeue \((\text{remove}())\) an element, where does it come from?

(a) We enqueue \((\text{add}(x))\) at the head and we dequeue \((\text{remove}())\) at the tail
(b) We enqueue \((\text{add}(x))\) at the tail and we dequeue \((\text{remove}())\) at the head
(c) We enqueue \((\text{add}(x))\) at the head and we dequeue \((\text{remove}())\) at the head
(d) We enqueue \((\text{add}(x))\) at the tail and we dequeue \((\text{remove}())\) at the tail
(e) None of the above

24. Consider how to implement a Stack as an SLList. When we push an element where does it go? When we pop an element where does it come from?

(a) We push at the head and we pop at the tail
(b) We push at the tail and we pop at the head
(c) We push at the head and we pop at the head
(d) We push at the tail and we pop at the tail
(e) None of the above

25. Using the best method you can think of, how quickly can we find the \(i\)th node in an SLList?

(a) in \(O(1 + i)\) time
(b) in \(O(1 + n - i)\) time
(c) in \(O(1 + n - i)\) time
(d) in \(O(1 + \min\{i, n - i\})\) time
(e) in \(O(1 + \min\{i, n \cdot (n - i - 1)\})\) time

26. Recall our implementation of a doubly-linked list (DLList):

```java
protected class Node {
    Node next, prev;
    T x;
}
public class DLLList<T> extends AbstractSequentialList<T> {
    protected Node dummy;
    protected int n;
    ...
}
```
Explain the role of the dummy node. In particular, if our list is non-empty, then what are `dummy.next` and `dummy.prev`?

(a) `dummy.next` and `dummy.prev` are both the first node in the list
(b) `dummy.next` and `dummy.prev` are both the last node in the list
(c) `dummy.next` is the last node in the list and `dummy.prev` is the first node in the list
(d) `dummy.next` is the first node in the list and `dummy.prev` is the last node in the list
(e) None of the above is true

27. Consider the correctness of the following two methods that add a node u before the node p in a DLList.

```java
protected Node add(Node u, Node p) {
    u.next = p;
    u.prev = p.prev;
    u.next.prev = u;
    u.prev.next = u;
    n++;
    return u;
}

protected Node add(Node u, Node p) {
    u.next = p;
    u.next.prev = u;
    u.prev = p.prev;
    u.prev.next = u;
    n++;
    return u;
}
```

(a) The first method is correct
(b) The second method is correct
(c) Neither method is correct
(d) Both methods are correct
(e) Both (c) and (d)

28. What is the running-time of `add(i,x)` and `remove(i)` in a DLList?

(a) $O(1 + i)$ and $O(1 + i)$, respectively
(b) $O(1)$ and $O(1 + \text{size}() - i)$, respectively
(c) $O(1 + \text{size}() - i)$ and $O(1)$, respectively
(d) $O(1 + \min\{i, \text{size}() - i\})$ and $O(1 + \min\{i, \text{size}() - i\})$, respectively
(e) $O(1)$ and $O(1 + \text{size}() - i)$, respectively

29. If we place $n$ distinct elements into a hash table of size $m$ using a good hash function, how many elements do we expect to find in each table position?

(a) $O(n/m)$
(b) $O(m/n)$
(c) $O(n)$
(d) $O(m)$
30. Recall the multiplicative hash function \( \text{hash}(x) = (x.\text{hashCode()} \times z) \gg \gg w-d \), where \( w \) is the number of bits in an integer. How large is the table that is used with this hash function? (In other words, what is the range of this hash function?)

(a) \{0, \ldots, 2^d\}
(b) \{0, \ldots, 2^d - 1\}
(c) \{0, \ldots, 2^{w-d}\}
(d) \{0, \ldots, 2^{w-d} - 1\}
(e) \{0, \ldots, 2^w - 1\}

31. In more standard mathematical notation, the above hash function can be written as (here div denotes integer division without any remainder)

(a) \( \text{hash}(x) = (x.\text{hashCode()} \cdot z) \div 2^{w-d} \)
(b) \( \text{hash}(x) = ((x.\text{hashCode()} \cdot z) \mod 2^w) \div 2^{w-d} \)
(c) \( \text{hash}(x) = ((x.\text{hashCode()} \cdot z) \mod 2^w) \div 2^d \)
(d) \( \text{hash}(x) = ((x.\text{hashCode()} \cdot z) \mod 2^{w-d}) \div 2^d \)
(e) \( \text{hash}(x) = ((x.\text{hashCode()} \cdot z) \mod 2^{w-d}) \div 2^{w-d} \)

32. Consider the following implementation of a \( \text{hashCode}() \) method that uses the bitwise exclusive-or (\(^\wedge\)) operator

```java
public class Point2D {
    Double x, y;
    ...
    public int hashCode() {
        return x.hashCode() ^ y.hashCode();
    }
}
```

Which of the following statements are true about two instances \( p \) and \( q \) of a Point2D?

(a) \( p.\text{hashCode()} = 0 \) if \( p.x = p.y \)
(b) \( p.\text{hashCode()} \neq q.\text{hashCode()} \) if \( p.x \neq q.y \) or \( p.y \neq q.x \)
(c) \( p.\text{hashCode()} = q.\text{hashCode()} \) if \( p.x = q.y \) and \( p.y = q.x \)
(d) Both (a) and (b) are true
(e) Both (a) and (c) are true

33. Consider the following implementation of a \( \text{hashCode}() \) method

```java
public class Point2D {
    Double x, y;
    ...
    public int hashCode() {
        return x.hashCode() + y.hashCode();
    }
}
```
Which of the following statements are true about two instances `p` and `q` of a `Point2D`?

(a) `p.hashCode() = 0` if `p.x = p.y`
(b) `p.hashCode() ≠ q.hashCode()` if `p.x ≠ q.y` or `p.y ≠ q.x`
(c) `p.hashCode() = q.hashCode()` if `p.x = q.y` and `p.y = q.x`
(d) Both (a) and (b) are true
(e) Both (a) and (c) are true

34. Consider the following implementation of a `hashCode()` method

```java
public class Point2D {
    Double x, y;
    ...
    public int hashCode() {
        return 37*x.hashCode() + y.hashCode();
    }
}
```

Which of the following statements are true about two instances `p` and `q` of a `Point2D`?

(a) `p.hashCode() = 0` if `p.x = p.y`
(b) `p.hashCode() ≠ q.hashCode()` if `p.x ≠ q.y` or `p.y ≠ q.x`
(c) `p.hashCode() = q.hashCode()` if `p.x = q.y` and `p.y = q.x`
(d) Both (a) and (b) are true
(e) Both (a) and (c) are true

35. Below is a portrait of

(a) Robert Endre Tarjan
(b) Carl Friedrich Gauss
(c) Zeno of Elea
(d) Pat Morin wearing a Robert Endre Tarjan mask
(e) Michiel Smid wearing a Zeno of Elea costume