**GRIFFITH COLLEGE DUBLIN**

**GRIFFITH COLLEGE CORK**

**GRIFFITH COLLEGE LIMERICK**

**QUALITY AND QUALIFICATIONS IRELAND**

**EXAMINATION**

**HIGHER CERTIFICATE IN COMPUTING**

**STAGE II**

**DATA STRUCTURES AND ALGORITHMS**

**Module Code: HCC-DSA**

**BACHELOR OF SCIENCE IN COMPUTING**

**STAGE II**

**DATA STRUCTURES AND ALGORITHMS**

**Module Code: BSCO-DSA**

**BACHELOR OF SCIENCE (HONS) IN COMPUTING SCIENCE**

**STAGE II**

**DATA STRUCTURES AND ALGORITHMS**

**Module Code: BSCH-DSA**

**Lecturer(s): Tony Mullins**

**Dr. Aqeel H. Kazmi**

**Elaine Tynan**

**Ms Begum Genc**

**External Examiner(s): Dr Joseph Timoney**

**Date: 20th August 2018 Time: 2.15-4.15**

**THIS PAPER CONSISTS OF TWELVE QUESTIONS**

**TEN QUESTIONS TO BE ATTEMPTED**

**ALL QUESTIONS CARRY EQUAL MARKS**

**APPENDIX AT THE BACK OF THE EXAMINATION PAPER**

**QUESTION 1**

1. Write a function to compute *fib(n)*, a term in the Fibonacci sequence, that is defined recursively as:



**(5 marks)**

1. Write a recursive function that computes the sum of the terms in an integer array.

**(5 marks)**

**Total (10 marks)**

**QUESTION 2**

1. Using the statement execution times defined for HAL (See **Appendix A** at the end of the exam paper), calculate the running time of the given code fragment.

int f[] = new int[100];

for(int j = 0; j < f.length; j = j + 1) f[j] = (int)(Math.random()\*50);

int s = 0;

for(int j = 0; j < f.length; j = j + 1) s += f[j];

**(4 marks)**

1. Using the laws of *big O* prove that the computation time of the given code fragment is *O(N2)*. (The laws are given in **Appendix A** of this exam paper.)

static boolean unique(int f[]){

boolean all = true; int a = 0;

while(a < f.length){

int b = a + 1;

while(b < f.length){

all = all && f[a] != f[b];

b = b + 1;

}

a = a + 1;

}

return all;

}

**(6 marks)**

**Total (10 marks)**

**QUESTION 3**

A set is a data structure that does not allow duplicate values. This means that any attempt to add a new element that already exists in the set results in no change to the set. The class IntSet, given below, uses an array of Integer to model a set. The constructor for the class is given and the maximum capacity of the set is given by argument n. Your task is to complete the methods whose signatures are given.

The method contains should return true if x an element of the set; false otherwise **(3 marks)**. The method add should append a new element to the set under the rule given above; if the set is full then it does nothing **(3 marks)**. The method empty returns true if the set has no elements; false otherwise **(2 marks)**. Finally, the method max returns the largest value in a non-empty set; null if the set is empty. **(2 marks)**.

class IntSet{

private Integer set[];

private int size;

IntSet(int n){

set = new Integer[n];

size = 0;

}

boolean contains(Integer x){ … }

void add(Integer x){ … }

boolean empty(){ … }

Integer max(){ … }

}

**(10 marks)**

**QUESTION 4**

1. Write a function that sorts an array of integer values. You may use any sorting algorithm you have studied.

**(7 marks)**

1. Explain why *quick sort* is often preferred over *merge sort* even though it has a worst case performance of *O(n2).*

**(3 marks)**

**Total (10 marks)**

**QUESTION 5**

1. What is the difference between a singly linked list and a doubly linked list? Draw a diagram to illustrate a doubly linked list of integer values. The list of data is: *5, 9, 5, 1, 6, 7, 7, 2*. Items should be inserted at the tail of the list.

**(4 marks)**

1. Given below are two classes, Node and LinkedListInt. Class Node implements a node in a linked list and class LinkedListInt implements a singly linked list of nodes where new nodes are inserted at the head of the list. Your task is to write two methods for this class. The methods are: delHead that deletes the current head of a non-empty list **(3 marks)** and size that returns the number of elements in the list **(3 marks)**.

class Node{

int data;

Node next;

public Node(int x){data = x; next = null;}

public Node next(){return next;}

public void setNext(Node p){next = p;}

public int data(){return data;}

}

class LinkedListInt{

Node head = null;//empty list

public void add(int x){ //add at head

Node nw = new Node(x);

nw.setNext(head);

head = nw;

}

public void delHead(){ … }

public int size(){ … }

}

**Total (10 marks)**

**QUESTION 6**

1. In relation to the design of data structures explain what the term *genericity* means. Why is it important to make data structures *generic*?

**(2 marks)**

1. Explain the difference between dynamic data structures and fixed size data structures.

**(2 marks)**

1. With reference to a data structure class what is the purpose of an iterator? What methods must an implementation of the Iterator interface implement?

**(2 marks)**

1. When you are planning to use the data structure TreeSet to manage your collection of objects what method must your class implement? Why must you implement this method? Why should the attributes used by this method be immutable?

**(4 marks)**

**Total (10 marks)**

**QUESTION 7**

1. A queue is a *first in, first out* linear data structure. Queues are typically characterized by three methods: join, leave, and head. Explain the semantics of each of these operations.

**(3 marks)**

1. Given below is an interface for a generic queue. Provide an implementation of this interface using an appropriate class from the Collections library. (See **Appendix B** for a list of classes and their methods)

interface Queue<E>{

public boolean join(E x);

public boolean leave();

public E head();

}

**(7 marks)**

**Total (10 marks)**

**QUESTION 8**

1. Using class Predicate<T> write a predicate called pos that takes an integer as argument and returns true if it is a positive number; false otherwise.

**(2 marks)**

1. What are higher order functions?

**(3 marks)**

1. Given below is the class ListInteger that encapsulates a list of integer values. It has three public methods: add that takes a list as argument and appends its elements to the data list; get that takes a predicate as argument and returns only those values that satisfy the given predicate; map that takes a function as argument and returns a list obtained by applying the function to each element in the data list. Your task is to complete the given code fragment by writing lambda expressions for the public methods get and map.

//Code fragment

ListInteger dt = new ListInteger();

dt.add(Arrays.asList(0,1,2,3,4,5,6,7,8,9,10));

//retrieve the list of even numbers in dt

//retrieve the list of numbers squared from dt

**(5 marks)**

class ListInteger{

private ArrayList<Integer> data = new ArrayList();

public void add(List<Integer> ls){data.addAll(ls);}

public List<Integer> get(Predicate<Integer> pred){

List<Integer> tmp = new ArrayList<>();

for(Integer x : data) if(pred.test(x)) tmp.add(x);

return tmp;

}

public List<Integer> map(Function(Integer,Integer) fn){

List<Integer> tmp = new ArrayList<>();

for(Integer x : data) tmp.add(fn.apply(x));

return tmp;

}

}

**Total (10 marks)**

**QUESTION 9**

1. State the definition of a binary search tree and, using a diagram, insert the following list of elements in a binary search tree: *11, 9, 13, 5, 6, 17, 1, 4, 3, 7, 15*.

**(4 marks)**

1. Delete the node containing 5 from the tree you constructed for part (a).

**(3 marks)**

1. State the algorithm for a *post-order* traversal of a binary search tree and use this algorithm to list the output from a traversal of your tree in part a.

**(3 marks)**

**Total (10 marks)**

**QUESTION 10**

1. A Map consists of a set of *key-value* pairs. What is the relationship between *key* elements and *value* elements?

**(2 marks)**

1. A class is required to record the sale of cars. An outline of this class is given below and the Map that associates a car (String) with its sales to date (Integer) is given. Your task is to complete the methods whose signatures are given. In each case the description of the purpose of the method is provided as a comment.

**(2 marks for each method)**

public class CarSales{

private Map<String, Integer> map = new TreeMap<>();

public Set<String> cars(){//return the list of cars}

public Integer sales(String car){

// return the total sales for a given car

}

public Integer totalSales(){

//calculate total sales of all cars recorded

}

public void add(String car){

//add sale of 1 car to the map

}

public String toString(){return map.toString();}

}

(See **Appendix B** at the end of the exam paper for relevant methods for this question.)

**Total (10 marks)**

**QUESTION 11**

1. Creating a binary search tree with a sorted list of values results in a linear linked list. Explain why this cannot happen with an *avl* tree.

**(3 marks)**

1. Create an *avl* tree by inserting the numbers in order 8,12,9.

**(3 marks)**

1. Using the tree created in part (b) insert the following numbers 4, 7, 3, 1. For full marks you must show the construction of the tree at each stage.

**(4 marks)**

**Total (10 marks)**

**QUESTION 12**

1. Given in the figure below is a graph G. Draw a picture of an adjacency list data structure that represents this graph



**(3 marks)**

1. List the order of nodes visited using a *depth-first* traversal of the graph.

**(2 marks)**

1. Explain, with the aid of diagrams, how a hash table can be used to optimise, provide an *O(1)* solution, the cost of insertion and retrieval for data collections.

**(5 marks)**

**Total (10 marks)**

**Appendix A**

|  |  |
| --- | --- |
| Calculating Running Times on HAL | |
| **Statement** | **Unit cost (ns)** |
| -, \*, /, %, ^, <, >, ==, >=, <=, !=, = | *10ns* |
| Function invocation | *50ns* |
| Argument passing | *10ns* per argument |
| Return | *50ns* |
| if(b) s1; else s2 | the cost of b plus the max cost of s1, s2 |
| for, while loops | *totalCost = cost of initialization of variables +*  *(n+1) \* cost of evaluating guard on loop*  *+*  *n \* cost of executing loop body,*  *where n equals the number of iterations of the loop.* |
| new | *100ns* |
| Calculating array indices | *50ns* |
| Math.random() | *100ns* |

**Laws of *big-O***

The laws of *big-O are*:

1. **Summation**

*O(1)+O(1)+..+O(1) = k \* O(1) = O(1)*, where *k* is a constant.

*O(n) + O(n)+..+O(n) = k \* O(n) = O(n)*, where *k* is a constant

*O(n) + O(m) = max(O(n), O(m))*

e.g. *O() + O( ) = O( )*

1. **Product**

*O(n) \* O(n) = O*

*n \* O(n) = O*

*O(n) \* O(m) = O(n \* m)*

*O(k \* f(n)) = k \* O(f(n)) = O(f(n))*, where *k* is a constant

*O() \* O() = O()*

The *big-O* sets of order functions form a chain of sub-sets as follows:

**Appendix B**

|  |  |
| --- | --- |
| Constructor | ArrayList<E>()  ArrayList<E>(Collection)  LinkedList<E>()  LinkedList<E>(Collection) |
| Insert item | add(E elem) |
| Insert list | addAll(Collection<? extends E> lst) |
| Remove item | remove(Object ob) |
| Contains item | Boolean contains(Object ob) |
| Number of elements | int size() |
| Convert to string | toString() |
| Empty set | Boolean isEmpty() |
| Remove elements | clear() |
| Retrieve element given index value | E get(int index); |
| Insert element at index | add(int index, E elem); |
| Change element at index | E set(int index, E elem); |
| Remove element at index | E remove(int index) |
| Get index of object | int indexOf(E elem); |
| **Additional Methods for LinkedList class** |  |
| Add new element at head of list | addFirst(E elem) |
| Return element at head of list | E getFirst() |
| Remove element at head of list | E removeFirst() |
| Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array. If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list. | <T> T[] toArray(T[] a)  An example is:  ArrayList<Integer> lst = new ArrayList<>(Arrays.asList(3,2,6,9,1));  Integer f[] = new Integer[lst.size()];  f = lst.toArray(f); |
| Applies the given action function to all the elements in the list in order. | forEach(Consumer<? super E> action) |
| Removes all values that satisfy the given predicate filter | removeIf(Predicate<? super E> filter) |
| Replaces each element of this list with the result of applying the operator function op to that element. | replaceAll(UnaryOperator<E> op) |
| Sorts this list according to the order specified by the given Comparator cmp. | sort(Compaparator<? super E> cmp) |

|  |  |
| --- | --- |
| Constructor | ArrayDeque<E>()  ArrayDeque<E>(Collection)  ArrayDeque(int numElements) |
| Insert item | addFirst(E elem)  addLast(E elem) |
| Get element without removing it – throws exception if queue empty | E getFirst()  E getLast() |
| Get element without removing it – returns null is queue empty | E peekFirst()  E peekLast() |
| Contains item | Boolean contains(Object ob) |
| Number of elements | int size() |
| Returns true if queue empty | Boolean isEmpty() |
| Convert to string | toString() |
| Empty set | Boolean isEmpty() |
| Remove elements | clear() |
| Retrieve head or tail element, returning null if queue empty | E pollfirst()  E pollLast() |
| Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array. If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list. | <T> T[] toArray(T[] a)  An example is:  ArrayDeque<Integer> dlst = new ArrayDeque<>(Arrays.asList(3,2,6,9,1));  Integer f[] = new Integer[dlst.size()];  f = dlst.toArray(f); |

|  |  |
| --- | --- |
| Constructor | HashMap<K,V>()  HashMap <K,V>(Map<? extends K,  ? extends V> mp)  TreeMap<K,V>()  TreeMap <K,V>( Map<? extends K,  ? extends V> mp)  EnumMap(Class<K> keyType) |
| Add or replace a key-value pair | put(K key, V value)  putAll(Map<? extends K,  ? extends V> mp) |
| If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current value. | V putIfAbsent(K key, V value) |
| Remove key-value pair and returns value associated with key, or null | V remove(Object key) |
| Replaces the entry for the specified key only if it is currently mapped to some value. | V replace(K key, V value) |
| Replaces the entry for the specified key only if currently mapped to the specified value. | boolean replace(K key, V oldValue, V newValue) |
| Contains key | boolean containsKey(Object key) |
| Contains value | boolean containsValue(Object value); |
| Number of elements | int size() |
| Convert to string | toString() |
| Empty set | boolean isEmpty() |
| Remove elements | clear() |
| Retrieve value | V get(Object key); |
| Retrieve the key set | Set <K> keySet(); |
| Retrieve values | Collection<V> values(); |

**Table of Specialized Functions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function**  **Name** | **Argument Type** | **Return**  **Type** | **Abstract Method Name** | **Purpose** |
| Function<T,R> | T | R | apply | Takes one argument and return a value of type R |
| BiFunction<T,U,R> | T,U | R | apply | Takes two arguments and return a value of type R |
| Supplier<T> | None | T | get | Takes no argument and return a value of type T |
| Consumer<T> | T | void | accept | Consumes a value of type T |
| BiConsumer<T,U> | T, U | void | accept | Consumes values of type T and U |
| UnaryOperator<T> | T | T | apply | A function that takes a value of type T as argument and returns a value of type T |
| BinaryOperator<T> | T, T | T | apply | A function that takes two values of type T as argument and returns a value of type T |
| Predicate<T> | T | boolean | test | A function that takes a value of type T and returns a boolean value. |
| BiPredicate<T, U> | T, U | boolean | test | A function that takes two arguments of type T and U and returns a boolean value. |