**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: 17th May 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. Given below is a recursive function called fac that calculates n! Use this function to calculate **fac(6).**

static int fac(int n){

if(n == 0) return 1;

else return n \* fac(n-1);

}

**(4 marks)**

1. Given a sorted integer array f, write a recursive binary search function that searches the array in *O(log n)* for a given value x, where n = f.length. The signature of your function should be: static boolean bSearch(int f[], int lb, int ub, int x).

**(6 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), by calculating the running times for each of the separate code fragments A and B listed below show that the cost of A is *630ns* and that of B is 30 + *30 \* 2 ^ 20ns*. (**Note**: *2 ^ 20* means 2 to the power of 20).

// A ===========================

int k = 1; int N = 2 ^ 20;

while(k < N){ k = k \* 2;}

// B ===========================

int k = 1; int N = 2 ^ 20;

while(k < N){ k = k + 1;}

**(5 marks)**

1. Calculate *big-O* for each of the given functions and, hence, show that function sumSq1 *performs better* than sumSq2.

**static** **long** sumSq1(**long** n){

**long** s = n\*(n+1)\*(2\*n+1)/6;

**return** s;

}

static long sumSq2(long n){

long s = 0;

for(int j=0; j < n; j++) s=s+(j+1)\*(j+1);

return s;

}

**(5 marks)**

**Total (10 marks)**

**QUESTION 3**

1. Show that the function div2Count has a best case performance of *O(1)* and a worst case performance of .

static int div2Count(int n){

int count = 0;

while(n % 2 == 0){

count++;

n = n/2;

}

return count;

}

**(4 marks)**

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

**(2 marks)**

1. Given below is the class Array<T> that encapsulates an array data. This class has a default constructor that creates an Object array typecast to type T. This array has an initial length of 20 (increment). It has attributes: size that always equals the current size or number of items in the data array and increment that equals the additional memory required when the current array is full, i.e. size == data.length. Your task is to write the add method that appends x to the array increasing its memory allocation, if required.

class Array<T>{

private T[] data;

int size = 0; int increment = 20

public Array(){data = (T[])(new Object[increment]);}

public void add(T x){ ... }

}

**(4 marks)**

**Total (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. Analyse the performance of your chosen sorting function and contrast it with any other sorting function you have studied in your course.

**(3 marks)**

**Total (10 marks)**

**QUESTION 5**

1. Draw a diagram of a singly linked list where new elements are added at the tail of the current list. Use the following list of numbers: 3, 5, 6, 7, 1, 2, 4.

**(3 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: add that adds a new element at the current head of the list **(4 marks)** andsum that returns the sum of the 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){ … }

public int sum(){ … }

}

**Total (10 marks)**

**QUESTION 6**

1. All the data structures provided by the Java Collection library are *generic* and use *dynamic allocation of memory*. Explain what the terms *generic* and *dynamic allocation of memory mean*.

**(3 marks)**

1. When writing a generic data structure that imposes an ordering on its data values what interface must all classes using it implement and why? Give an example of a data structure from the Collection classes in Java that requires its data types to implement this interface.

**(3 marks)**

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

**(4 marks)**

**Total (10 marks)**

**QUESTION 7**

1. Explain the difference between a Stack and a Queue.

**(2 marks)**

1. You are asked to choose between using an ArrayList and a LinkedList when implementing a Queue. Which one would you choose and why?

**(2 marks)**

1. Given below is a generic class Stack<E> that implements the standard stack methods push, pop and top using a LinkedList class to store the data references. Your task is to complete each of these methods by using appropriate methods from the LinkedList class. (See **Appendix B** below).

class Stack<E>{

private LinkedList<E> stack = new LinkedList<>();

public boolean push(E x){…}

public boolean pop(){…}

public E top(){…}

}

**(6 marks)**

**Total (10 marks)**

**QUESTION 8**

1. Using class Function<T,R> write a function called abs that takes an integer as argument and returns its absolute value.

**(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; sum that takes a predicate as argument and returns the sum of those values that satisfy the given predicate; modify that takes a function as argument and uses the method replaceAll to apply the function to the elements in the list. Your task is to complete the given code fragment by writing lambda expressions for the public methods sum and modify.

(See **Appendix B** at the back of this paper)

//Code fragment

ListInteger dt = new ListInteger();

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

//calculate sum of positive numbers in dt

//modify the list so that all values are positive

class ListInteger{

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

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

public Integer sum(Predicate<Integer> pred){

int sum = 0;

for(Integer x : data) if(pred.test(x)) sum += x;

return sum;

}

public void modify(Function(Integer,Integer) fn){

data.replaceAll(fn);

}

}

**(5 marks)**

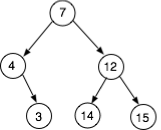
**Total (10 marks)**

**QUESTION 9**

1. State the definition of a binary search tree.

**(2 marks)**

1. Explain why the given tree is a binary tree but not a binary search tree.



**(2 marks)**

1. Create a binary search tree with the following list of elements: *14, 8, 16, 5, 6, 4, 1, 7*.

**(3 marks)**

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

**(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. Explain the difference between TreeMap and HashMap.

**(2 marks)**

1. A class is required to record words and their frequency of occurrence. An outline of this class is given below and the Map that associates a word (String) with its frequency of occurrence (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.

(See **Appendix B** at end of this paper)

**(6 marks)**

public class WordFreq{

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

public Set<String> words(){ return map.keySet();}

public Integer freq(String wd){

// return the frequency of occurrence of wd

}

public void add(String wd){//add word wd to the map}

public List<String> getWords(int n){

//return list of words with frequency n

}

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

}

**Total (10 marks)**

**QUESTION 11**

1. Explain how binary search trees can become unbalanced and, hence, not deliver optimal insertion and retrieval performance. How do *avl* trees overcome this unbalancing?

**(4 marks)**

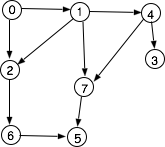
1. Using diagrams create an *avl* tree by inserting the following numbers in sequence: 12, 4, 8, 17, 12, 20, 15. For full marks you must show the construction of the tree at each stage.

**(6 marks)**

**Total (10 marks)**

**QUESTION 12**

1. Given 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 *breadth-first* traversal of the graph.

**(2 marks)**

1. How do B-Trees differ from binary search trees?

**(3 marks)**

1. How do B-trees optimize the cost of insertion?

**(2 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. |