**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**

**Patrick Corrigan/Elaine Tynan**

**External Examiner(s): Thanh Thoa Pham Thi**

**Date: 11th May 2017 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 recursive function that implements Euclid’s algorithm to find the greatest common divisor of two positive numbers. The algorithm is defined as follows:

**(5 marks)**

1. Test your solution by showing that: gcd(45,60) = 15.

**(3 marks)**

1. What is the difference between a tail recursive function and non-tail recursive function?

**(2 marks)**

**Total (10 marks)**

**QUESTION 2**

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

int f[] = new int[1000];

for(int j = 0; j < f.length; j++){

if(j % 2 != 0) f[j] = 1;

else f[j] = 0;

}

**(5 marks)**

1. Show that function sumN is *O(1)* and sumN1 is *O(n)*. What conclusion can be drawn from this analysis?

**static** **long** sumN(**long** n){

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

**return** s;

}

**static** **long** sumN1(**long** n){

**long** s = 0;

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

**return** s;

}

**(5 marks)**

**Total (10 marks)**

**QUESTION 3**

1. What do we mean by stating that a program whose cost function is *)* *performs better* that one that has a cost function of *O(n)*? Explain why binary searching *performs better* than linear searching.

**(3 marks)**

1. Would you say that divide and conquer algorithms are *O(n)* or *)*?

**(1 mark)**

1. Draw a diagram to illustrate a linked list of integer values. The list should be constructed by entering the following list of numbers in the given order: 1, 4, 5, 6, 7, 2, 10, 12. Numbers should be inserted at the tail of the list.

**(3 marks)**

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

**(3 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**

Given below is the class StringList that uses a singly linked list to manage a collection of strings. New elements are inserted at the head of the list and the method add(String x) is given. The private class Node is used to implement nodes in the list and encapsulates both the data element x and a pointer to the next node in the list, if any. Its methods should be familiar to you from the work covered in lectures and labs. Your task is to complete the three methods whose signatures are given. Method add(String f[]) that inserts an array of strings in the list **(3 marks)**; method numChars() that counts the number of characters in all the strings in the list **(4 marks)** and method size() should return the number of strings in the list **(3 marks)**.

class StringList{

Node head = null;

public void add(String x){

Node nw = new Node(x);

if(head == null) head = nw;

else{

nw.setNext(head);

head = nw;

}

}

public void add(String f[]){..}

public int numChars(){..}

public int size(){..}

private class Node{

String data;

Node next;

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

public Node next(){return next;}

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

public String data(){return data;}

}

**Total (10 marks)**

**QUESTION 6**

1. A stack is a *last in, first out* linear data structure. It is characterized by two main operations: push and pop. The push operation adds a new item to the top of the stack, or initializes the stack if it is empty. The pop operation removes the element at the top of the stack, if not empty. This means that elements are removed in inverse order to their insertion. Those last in get to leave first. To inspect the current element at the head of the stack a method top is provided. Given below is the generic class StackArray<E> that uses an ArrayList to implement stack behaviour. The methods size() and toString() are provided. Your task is to implement methods pop, push and top. (See **Appendix** at the end of the exam paper for relevant methods.)

class StackArray<E>{

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

…

public int size(){return stack.size();}

public String toString(){

return stack.toString();

}

}

**(6 marks)**

1. Using your class StackArray write a code fragment that creates a stack of 10 randomly generated integer values such that the value at the top of the stack is always the largest value.

**(4 marks)**

**Total (10 marks)**

**QUESTION 7**

1. Using class Function<T,R> write a function called square that takes an integer as argument and returns the square of its value. Write an assert statement to test your function.

**(3 marks)**

1. Write a Predicate function called allEven that takes a list of integers as argument and returns true if the list contains only even numbers; false otherwise.

**(2 marks)**

1. What are higher order functions?

**(3 marks)**

1. Using the higher order method replaceAll from class ArrayList write a lambda expression as argument that multiplies all values in lst, given below, by 2.

ArrayList<Integer> lst = new ArrayList<>(Arrays.asList(2,3,4,5,6,7,8,9));

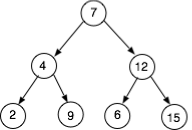
**(2 marks)**

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

**Total (10 marks)**

**QUESTION 8**

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



**(2 marks)**

1. Using a diagram insert the following list of elements in a binary search tree:

*6,3,8,7,2,0,10,1.*

**(3 marks)**

1. In the binary search tree, created for part b, list the order in which the nodes are visited under *preorder* and *postorder* traversals.

**(4 marks)**

1. Name the traversal required that retrieves an ordered list.

**(1 mark)**

**Total (10 marks)**

**QUESTION 9**

Given below is a class called PersonHobbies that uses a map to model the relationship between persons and their hobbies. The constructor creates a default map with some sample persons and their hobbies.

Your tasks are:

1. List the values of the set returned by the method persons();

**(2 marks)**

1. List the values returned by the method hobbies();

**(2 marks)**

1. Complete the method listPerson(String h) that takes a hobby as argument and returns those persons that participate in h;

**(3 marks)**

1. Complete the method numHobbies(String p) that takes a person p as argument and returns the number of hobbies for p.

**(3 marks)**

class PersonHobbies{

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

public PersonHobbies(){

map.put("John", new ArrayList<>(Arrays.asList("Football","Cinema","Golf")));

map.put("Mary",new ArrayList<>(Arrays.asList("Cinema","Walking")));

map.put("Sheila",new ArrayList<>(Arrays.asList("Golf")));

}

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

public Set<String> hobbies(){

Set<String> tmp = new TreeSet<>();

for(String n : map.keySet()) tmp.addAll(map.get(n));

return tmp;

}

public Set<String> listPerson(String h){..}

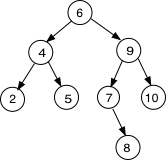
public int numHobbies(String p){..}

}

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

**Total (10 marks)**

**QUESTION 10**

1. Show that the *avl* tree given to the right is balanced.

**(3 marks)**

1. What is the advantage of using *avl* trees over binary search trees?

**(2 marks)**

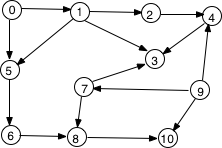
1. Insert the given list of values in an *avl* tree. The list is: 7, 4, 10, 3, 5, 6, 2, 1. For full marks you must show the construction of the tree.

**(5 marks)**

**Total (10 marks)**

**QUESTION 11**

1. Using the given graph below list the order of nodes visited using both a breadth first traversal and a depth first traversal.



**(4 marks)**

1. Construct a B-tree with a maximum of 4 items per node for the list of numbers: *9, 20, 1, 12, 25, 7, 14, 21, 6, 5, 10, 18, 15, 11, 4*.

**(6 marks)**

**Total (10 marks)**

**QUESTION 12**

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)**

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

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

**(5 marks)**

**Total (10 marks)**

**Appendix**

|  |  |
| --- | --- |
| 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:

|  |  |
| --- | --- |
| 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. |