**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/Dapeng Dong**

**Elanie Tynan**

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

**Date: 12th May 2016 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 sum that takes an array and an index value as argument and calculates the summation of the values in f. By evaluating (executing) this function with the given array dt show that the value of k, in the given code fragment, is 11.

{ int dt[] = {3,5,2,1};

int k = sum(dt,0);

}

static int sum(int f[], int j){

if(j == f.length) return 0;

return (f[j] + sum(f,j+1));

}

**(5 marks)**

1. Explain why the recursive function sum, is not tail-recursive. **(2 marks)** Re-write it so that it becomes a tail-recursive function. **(3 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 times for each of the separate code fragments A and B.

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

int x = 100;

int y = x \* 5 + x \* x - 2;

int z = x + y \* y - 56;

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

int k = 0; int s = 0;

while(k < 100){

s = s + (k + 1) \* (k + 1);

k = k + 1;

}

**(5 marks)**

1. Using the laws of *big O* 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. To prove that *f(n)* is *O(g(n))* we show that . Using this method, prove that , is *O(n2)*.

**(5 marks)**

1. Calculate a cost function for the given code fragment and show that it is *O(n)*.

int n = 1048576;

while(n > 0) n = n -1;

n = 1048576;

while(n > 0) n = n/2;

(Hint: 1048576 = 1Mb)

**(5 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. 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. Use a diagram to explain the difference between an array and a singly linked list.

**(3 marks)**

1. Explain why searching a linked list is always *O(n)*, where n equals the number of nodes in the list.

**(2 marks)**

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*.

**(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?

**(3 marks)**

**Total (10 marks)**

**QUESTION 6**

Given below is the class IntList that uses a singly linked list to manage a collection of integer values. New elements are inserted at the head of the list and the method add(int 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 sum() should calculate the sum of the elements currently in the list **(3 marks)**; method toString() should return a string representation of the data values in the list **(4 marks)** and method size() should return the number of nodes in the list **(3 marks)**.

class IntList{

Node head = null;

public void add(int x){

Node nw = new Node(x);

if(head == null) head = nw;

else{

nw.setNext(head);

head = nw;

}

}

public int sum(){ … }

public String toString(){ … }

public int size(){ … }

private 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;}

}

}

**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. Explain why circular arrays provide an optimal solution to the implementation of a queue.

**(1 mark)**

1. Given below is an interface for a generic queue. A partial implementation of this interface by class MyQueue<E> is listed below. This class uses an ArrayDeque instance to model the queue. Your task is to complete the methods join, leave and head. (See **Appendix** at the end of this paper for relevant methods.)

**(6 marks)**

interface Queue<E>{

public boolean join(E x);

public boolean leave();

public E head();

public boolean empty();

}

class MyQueue<E> implements Queue<E>{

private ArrayDeque<E> queue = new ArrayDeque<>();

…

public boolean empty(){

return queue.isEmpty();

}

}

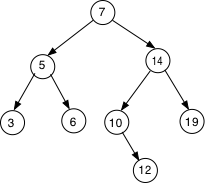
**Total (10 marks)**

**QUESTION 8**

1. State the definition of a binary search tree and, using a diagram, insert the following list of elements in a binary search tree:  *12, 6, 3, 7, 15, 13, 18, 21, 1*.

**(4 marks)**

1. In the binary search tree, given below, list the order in which the nodes are visited under *preorder, inorder* and *postorder* traversals.



**(6 marks)**

**Total (10 marks)**

**QUESTION 9**

1. A map is a collection of items where each item has two parts called a *key* and a *value*. In the case of a map define the relationship between keys and their associated values.

**(2 marks)**

1. Both HashMap and TreeMap provide implementations of the Map interface. What is the difference between them?

**(2 marks)**

1. The class NatAuthors listed below uses a TreeMap to model the relationship between nationalities and an associated list of authors. Each nationality is mapped to a list of authors. The constructor adds three nationalities and their associated authors to the list.

Your tasks are: draw a graph of the map created by the constructor **(2 marks)**; complete the method nationalities() that returns a list of the nationalities in the map and the method authors() that returns a list of authors in the map **(4 marks)**. In both cases the signatures of the methods are given.

class NatAuthors{

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

public NatAuthors(){

map.put("Irish", new ArrayList<>(Arrays.asList("Banville","Doyle","Barry")));

map.put("French",new ArrayList<>(Arrays.asList("Sartre","Flaubert")));

map.put("English",new ArrayList<>(Arrays.asList("Faulks","Dickens","Dahl")));

}

public Set<String> nationalities(){ … }

public List<String> authors(){ … }

}

**Total (10 marks)**

**QUESTION 10**

1. What do we mean by stating that an *avl* tree is a balanced binary search tree?

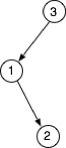
**(2 marks)**

1. Show that the given binary search tree can be balanced by a left rotation followed by a right rotation.

**(3 marks)**

1. Insert the given list of values in the re-balanced tree for part b. The list is: 4, 5, 7.

Note: You must show the construction of the tree as part of your answer.



**(5 marks)**

**Total (10 marks)**

**QUESTION 11**

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

**(2 marks)**

1. Why do B-trees *perform better* than binary search trees?

**(1 mark)**

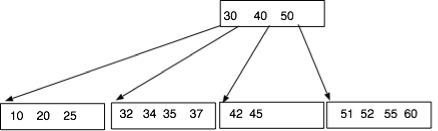
1. How do B-trees optimize the cost of insertion?

**(2 marks)**

1. Given below is a partially constructed B-Tree where each node has a maximum of 4 elements. Your task is to insert the following values in this tree: 22, 47, 65.

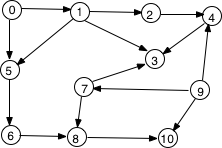
Note: You must show the construction of the tree as part of your answer.

**(5 marks)**



**QUESTION 12**

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



**(6 marks)**

1. Explain the difference between text files and binary files on disk.

**(4 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); |

|  |  |
| --- | --- |
| 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(); |