**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**

**SOLUTIONS**

**Lecturer(s): Tony Mullins**

**Dr. Aqeel H. Kazmi**

**Elaine Tynan**

**Ms Begum Genc**

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

**Date: Aug 2018 Time: TBC**

**THIS PAPER CONSISTS OF TWELVE QUESTIONS**

**TEN QUESTIONS TO BE ATTEMPTED**

**ALL QUESTIONS CARRY EQUAL MARKS**

# APPENDIX AT THE BACK OF THE EXAMINATION PAPER

**General Note: for each question if the student provides any relevant piece of work give an attempt mark. This mark should be no more than 20% of the marks for the given section unless stated otherwise. Do not use fractions – For example, if there are 3 marks the attempt mark should be 1.**

**Question 1**

static int fib(int n){

if(n == 0) return 1;

else if(n == 1) return 1;

else

return fib(n-1)+fib(n-2);

}

**5 marks**

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

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

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

}

**5 marks**

**Question 2**

int f[] = new int[100]; 110

10 10 20 50 +10+100+10

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

int s = 0; 10

10 10 20 20 + 50

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

Cost = 110+10 +101\*10+100\*170+10+10+101\*10+100\*70

As long as student shows where the costs are allocated give full marks. We are not checking their ability to add!

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;

}

Let n = f.length

Cost = c1+n\*K+c2, where K = c3+(n-b)\*c4+c5

Cost = c1+n(c3+n\*n\*c4 – n\*b\*c4)+n\*c5

O(cost) = O(c1)+O(n\*n\*c4) –O(n\*b\*c4)+O(n\*c5)

= O(1)+O(n\*n)-O(n)+O(n)

= O(n\*n)

**5 marks**

**Total 10 marks**

**Question 3**

class IntSet{

private Integer set[];

private int size;

IntSet(int n){

set = new Integer[n];

size = 0;

}

boolean contains(Integer x){ **3 marks**

for(Integer y : set) if(x.equals(y)) return true;

return false;

}

void add(Integer x){ **3 marks**

if(!this.contains(x)){

if(size < set.length){

set[size] = x; size++;

}

}

}

boolean empty(){return size == 0;} **2 marks**

Integer max(){ **2 marks**

if(size == 0) return null;

Integer m = set[0];

for(Integer y : set) if(m < y) m = y;

return m;

}

}

**Total 10 marks**

**Question 4**

Any sorting function **7 marks**

Analysis of why quicksort outperforms merge sort **3 marks**. Total 10 marks

**Question 5**

Difference between singly linked and doubly linked list **2 marks** – diagram will suffice for full marks

Drawing of doubly linked list **2 marks**

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(){ **3 marks**

if(head != null){

head = head.next();

}

}

public int size{ **3 marks – if student introduces size attribute and modifies**

**add then give full marks**

int s = 0;

Node tmp = head;

while(tmp != null){s++; tmp = tmp.next();}

return s;

}

}

**Question 6**

Genericity? **1 mark.** Why? **1 mark**

Difference between dynamic memory allocation and fixed size – **2 marks**

Purpose of iterator **1 mark** Methods **1 mark**

Comparable interface with compareTo method **2 marks**; data is sorted – **1 mark** why immutable **1 mark**

**Total 10 marks**

**Question 7**

Semantics of join, leave, head – **1 mark each**

class Q7Queue<E> implements Queue<E>{

private ArrayDeque<E> queue = new ArrayDeque<>(); **1 mark**

public boolean join(E x){ **2 marks**

if(x != null){

queue.addLast(x);

return true;

}

return false;

}

public boolean leave(){ **2 marks**

if(queue.isEmpty()) return false;

queue.pollFirst();

return true;

}

public E head(){ **2 marks**

if(queue.isEmpty()) return null;

return queue.getFirst();

}

}

**Total 10 marks**

**Question 8**

Predicate<Integer> pos = x -> x > 0; **2 marks**

Higher-order functions – **3 marks**

//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

List<Integer> ls = dt.get(x -> x % 2 == 0); - **3 marks**

List<Integer> ls1 = dt.map(x -> x\*x); - **2 marks**

**Question 9**

Definition – **2 marks** Diagram **2 marks**

Delete 5 **3 marks**

Post-order algorithm – left right root **1 mark**

Listing elements: 3,4,1,7,6,5,9,15,17,13,11 – **2 marks**

**(Total 10 marks)**

**Question 10**

Relationship between key and value? **2 marks**

**2 marks for each method correct**

public class CarSales{

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

public Set<String> cars(){

//return the list of cars

return map.keySet();

}

public Integer sales(String car){

// return the total sales for a given car

if(map.keySet().contains(car)) return map.get(car);

}

public Integer totalSales(){

//calculate total sales of all cars recorded

int total = 0;

for(String car : map.keySet()) total += map.get(car);

return total;

}

public void add(String car){

if(map.keySet().contains(car))

map.put(car,map.get(car)+1);

else

map.put(car,1);

}

public String toString(){

return map.toString();

}

}

**Total 10 marks**

**Question 11**

Height of left sub-tree never differs from height of right by more than 1 – keeps tree balanced **3 marks**

Avl tree with 8,12,9 showing rotations **3 marks.** If student just provides a diagram with no rotations shown deduct a mark

Tree constructed showing construction rotations that keep balance invariant **4 marks – if no construction shown and final diagram correct deduct 2 marks**

**Total 10 marks**

**Question 12**

Adjacency array showing lists of nodes – **3 marks**

Depth-first traversal – 0,1 2 3 4 7 5 6 **2 marks**

Discussion of hash tables showing how cost of insertion, using hashing function to map to index position, is optimised. Similarly, for retrieval.

**5 marks**

**Total 10 marks**