**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: May 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**

Fac(6) = 6\*fac(5) = 6\*5\*fac(4)=6\*5\*4\*fac(3) = … = 360 **4 marks**

static boolean bSearch(int f[], int lb, int ub, int x){

if(lb+1 == ub) return x == f[lb];

else{

int m = (lb+ub)/2;

if(x >= f[m])

return bSearch(f,m,ub,x)

else

return bSearch(f,lb,m,x);

}

} **6 marks**

**Question 2**

// 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;}

Must show actual calculations and use of log function for A

A is *630ns* and that of B is 30 + *30 \* 2 ^ 20ns*

If the student makes a mistake in the calculations but derives the correct function in each case then give full marks **5 marks**

**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;

}

O(sumSq1) = O(1) and O(sumSq2) = O(n)

O(1) << O(n)

**5 marks**

**Question 3**

Best case: n is odd, hence loop does not execute O(1)

Worst case: div 2 gives O(log n)

**4 marks**

Difference between dynamic and fixed sixe data structures – **2 marks**

class Array<T>{

private T[] data;

int size = 0; int increment = 20

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

public void add(T x){

if(size == data.length){

T[] tmp = (T[])(new Object[data.length+increment]);

int j = 0;

while(j < size){tmp[j] = data[j];j++;}

tmp[j] = x;

size++;

data = tmp;

}

}

} **4 marks**

**Question 4**

Any correct Sorting function **7 marks**

Analysis **3 marks**

**Question 5**

Diagram of singly linked list showing pointers **3 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;

} **4 marks**

public int sum(){

int s = 0;

Node tmp = head;

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

} **3 marks**

}

**Question 6**

Explanation of what the terms *generic* and *dynamic allocation of memory mean*. **3 marks**

Classes must implement comparable interface that provides a compareTo method **2 marks**

Example data structure – TreeSet, TreeMap **1 mark**

HashSet: must implement equals and hashCode **1 mark**

Used to insert and retrieve elements in the set – **2 marks**

Should be immutable because if modified in situ item not retrievable – **1 mark**

**Question 7**

Difference between a Stack and a Queue **2 marks**

Use linked list because no shuffling required – O(1) cost of join and leave **2 marks**

class Stack<E>{

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

public boolean push(E x){

stack.addFirst(x);

return true;

}

public boolean pop(){

if(stack.isEmpty()) return false;

stack.removeFirst();

return true;

}

public E top(){

if(stack.isEmpty()) return null;

return stack.getFirst();

}

} **6 marks – 2 for each method**

**Question 8**

Function<T,R> abs = x -> x >= 0 ? x:-x; **2 marks**

Higher-order functions – functions that take functions as arguments **– 3 marks**

//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

Intege sum = dt.sum(x -> x >= 0 ? x : -x);

dt.map(x -> x\*x);

**5 marks**

**Question 9**

Def of binary search tree – **2 marks**

Not binary search because all elements in left sub-tree not < root node **2 marks**

Construction of tree **3 marks**

Pre-order - Process root;left;right **3 marks**

**Question 10**

Relationship between *key* elements and *value* elements – 2 marks

Difference between TreeMap and HashMap – 2 marks

public class WordFreq{

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

public Set<String> words(){//return the set of words

return map.keySet();

}

public Integer freq(String wd){// return the frequency of occurrence of wd

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

return 0;

}

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

if(map.keySet().contains(wd))

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

else map.put(wd,1);

}

public List<String> getWords(int n){

//return list of words with frequency n

List<String> wds = new ArrayList<>();

for(String wd : map.keySet())

if(map.get(wd) == n) wds.add(wd);

return wds;

}

public

public String toString(){

return map.toString();

}

} **6 marks – 2 marks for each method correct**

**Question 11**

Explain how binary search trees can become unbalanced and, hence, not deliver optimal insertion and retrieval performance.

An example of how this occurs will suffice **2 marks**

How do *avl* trees overcome this unbalancing? Because avl trees are self balancing **2 marks**

Construction of tree showing rotations. Noe if final tree given and correct then only award 2 marks because the question clearly states that the construction of the tree is required. – **6 marks**

**Question 12**

Adjacency list – **3 marks**

Breadth first: 0, 1, 2, 4, 7, 6, 3, 5 **2 marks**

BTrees grow from bottom up and not top down **3 marks**

Optimise cost of insertion because tree is always minimum height and contains nodes of ordered values **2 marks**