

**CS2515**

**2024/25**

**SAMPLE CLASS TEST**

You have 45 minutes to complete answers to these questions.

Please complete the information on this page.

There are questions worth **20 marks** on the paper. You should answer all questions.

Write your answers directly into space provided on the question sheet.

**Student ID number:**

**Place X in box if registered with DSS:** ☐

**Place X in box if availing of a spelling and grammar waiver:** ☐

(for marking guidelines, see <https://www.ucc.ie/en/dss/publications/>)

1. You are implementing an application to maintain a music library. The library contains all tracks ever played or 'liked' by a user. The user can request to see those tracks ordered by data of release, or ordered alphabetically by artist. You have two functions you can use to re-order the collections on demand. The first has complexity  $O(n^2)$  to build an ordered collection of  $n$  tracks, and can be implemented in about 10 lines of code. The second has complexity  $O(n \log n)$  to build an ordered collection, and will take at least 300 lines of code to implement. Which function do you choose for the library? Justify your answer. **(2 marks)**

2. Consider the following sequence of actions to be applied to an empty stack.  
*push(x), push(a), print(pop()), push(c), push(y), print(pop()), print(pop()),*  
*push(d), push(s), push(p), print(pop()), print(pop()), print(pop())*  
Show the resulting output (ie what is printed, in what order?). **(1 marks)**

3. State the four standard methods for the Queue ADT, including any parameters and return values, with a brief comment explaining what each one does. **(2 marks)**

4. Describe in detail how an array-based list is implemented, so that accessing by index is  $O(1)$ , and appending and popping the last item are both  $O(1)$  on average, regardless of how large the list becomes. **(4 marks)**

5. A DoublyLinkedList is represented as a chain of *nodes*, specified by the class

```
class DLLNode:
    def __init__(self, item, prevnode, nextnode):
        self._element = item
        self._next = nextnode
        self._prev = prevnode
```

Give python code for the function below, which given a reference to a DLLNode 'nodebefore' which is somewhere in a doubly linked list with dummy head and tail nodes, and a piece of data 'item', creates a new DLLNode pointing to the item and adds it into the list immediately after 'nodebefore'.

```
def add_after(item, nodebefore)
```

(5 marks)

6. We are given the following hash function, represented as a lookup table:

input:	120	121	122	123	124	125	126	127
hash:	6935	7218	9426	8114	3427	9516	4354	6414

For a hash table implemented by an empty array of size 10, using open addressing with linear probing, show the changes to the array as we execute each of the following steps in turn, using the above hash function. There is no need to grow the array.

Steps	0	1	2	3	4	5	6	7	8	9
setitem(123,'x')										
setitem(125,'y')										
setitem(127,'w')										
setitem(120,'z')										
delitem(127)										
setitem(125,'t')										
setitem(126,'p')										

(6 marks)

**END OF PAPER**

*Page 3 of 3*