CS2515

2024/25

IF YOU HAVEN"T MADE A SERIOUS ATTEMPT AT THE SAMPLE PAPER, THEN GO BACK AND TRY IT FIRST BEFORE READING THESE SOLUTIONS…

SAMPLE CLASS TEST

SOLUTIONS

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.

(2marks)

Choose the O(n log n) function. Over time, you can expect that the number of tracks liked or played will become very large, and so the user will be repeatedly sorting very large lists. For large lists, a run time of O(n^2) is likely to be significantly slower than O(n log n), so the extra effort of writing 300 lines of code up front will be worth it.

2. Consider the following sequence of actions to be applied to a stack (which is initially empty).

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.

(1 marks)

aycpsd

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)

enqueue(item) % add item into the queue
dequeue() % remove and return the data item in the queue for the longest time
front() % return data item that has been in the queue for the longest time (but leave it in the queue)
length() % return the number of items on the queue

4. Describe in detail how an array-based list is implemented, so that the list size can grow arbitrarily large, but 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.

(4marks)

The list maintains references to data stored elsewhere. The references are stored packed into in a single block of data with no gaps. Accessing an element in index i is then just a jump to the location (start of list) + i*(size of one reference). If there is space reserved at the end, adding an item just assigns a reference to a cell, so O(1); if there is no space, a new block of (e.g.) twice the current size is allocated, and all existing references are copied across then the new one added - so O(n) -- but the next (n-1) additions are just O(1) each, so O(n) + O(n) = O(n) to add those n items, so O(1) on average for each item. Popping the last just wipes the reference. If the link shrinks to less than 0.25*space, allocate a new list of half the size and copy across.

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)
    newnode = DLLNode(item, None, None)
    newnode._prev = nodebefore
    newnode._next = nodebefore._next
    nodebefore._next._prev = newnode
    nodebefore._next = newnode

(could have done
def add_ after(item, nodebefore)
    newnode = DLLNode(item, nodebefore, nodebefore._next)
    nodebefore._next._prev = newnode
    nodebefore._next = newnode
)
```

(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')					(123,'x')					
setitem(125,'y')					|		(125,'y')			
setitem(127,'w')					|	(127,'w')	|			
setitem(120,'z')					|	|	|	(120,'z')		
delitem(127)					|	A	|	|		
setitem(125,'t')					|	|	(125,'t')	|		
setitem(126,'p')					|	(126,'p')	|	|		

(6 marks)

Total: 20 marks

END OF PAPER