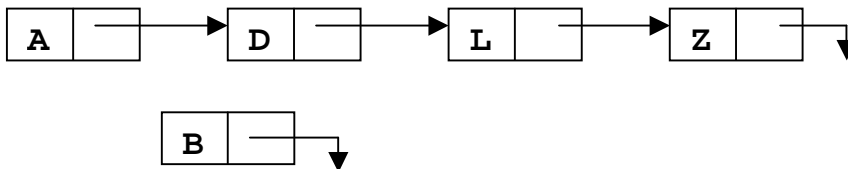


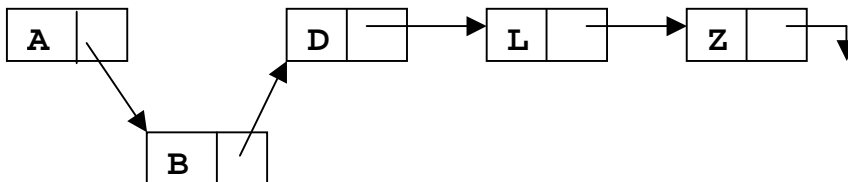
## CSE1303 Part A – Data Structures and Algorithms

### Question A1 (0.5 marks)

Given the following representation of a linked list and a new node:



Inserting the new node at position 1, you would end up with a linked list:



Number the links that were changed in the order they were done.

### Question A2 (0.5 marks)

Given the following section of C code:

```
#define size 10

int i, n;
float array[size];

for (i = 0; i < 5; i++){
    for ( n = 0; n < size; n++){
        array[n] = array[n] + i;
    }
    /**** A ****/
}
```

Which answer indicates the complexity of this section of code in big O notation?  
(Circle the correct answer)

- (a)  $O(n^2)$
- (b)  $O(n)$
- (c)  $O(\log(n))$
- (d)  $O(15)$

<b>1</b>

**Question A3 (1 mark)**

Given the following array values:

```
array = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

What are the contents of `array` at `/* ** * A * ** */` on the previous page?

--	--	--	--	--	--	--	--	--	--

<b>1</b>

Blank page for working

**Question A4 (3 marks)**

Assume a node in a linked structure is defined as:

```
#define MAXSTRING 20

struct NodeRec
{
    int num;
    float price;
    char *title;
    char *author;
    struct NodeRec nextPtr;
};

typedef struct NodeRec Node;
```

Also assume that:

```
Node* makeNode(char *title, char *name, int num, float price)
```

uses the function `char* strdup(const char *s)`.

Write the function:

```
void killNode(Node* NodePtr)
```

which deallocates all the memory associated with the node pointed to by `NodePtr` as well as the node.

<b>3</b>

Blank page for working

**Question A5 (3 + 2 = 5 marks)**

(a) The following function implements the selection sort algorithm. Insert the missing code.

```
void selectionSort(float array[], int size)
{
    int j, k;
    int maxPosition;
    float tmp;

    for (k = size-1; k > _____ ; k--)
    {
        maxPosition = 0;
        for (j = _____ ; j <= _____ ; j++)
        {
            if (array[j] _____ array[maxPosition])
                maxPosition = _____;
        }
        tmp = array[_____];
        array[_____] = array[_____];
        array[_____] = tmp;

        /**** A ****/
    }
}
```

(b) Given the following parameter values:

`array = {10, 3, 13, 7, 11} , size = 5`

Fill in the missing values in the table each time `**** A ****` is reached

Times <code>**** A ****</code> is reached	<code>array[0]</code>	<code>array[1]</code>	<code>array[2]</code>	<code>array[3]</code>	<code>array[4]</code>
1					
2					
3					
4					

<b>5</b>

End of test