**Note: These practice questions are of the style that you will see on the exam. This set of practice problems is longer than what you will see on the actual midterm.**

## Big-(O):

1. **What is the big – O notation for:**
   - **a.)** A method that takes exactly $2n^2+5n+100$ steps O($n^2$)
   - **b.)** for(i = n; i > 0; i = i/2) { O(log n)

```
  //constant time operations

  …

  }
```

   - **c.)** binary search O(logn)

   - **d.)** The following lines of code O(n)

```
current = lst->frontSentinel->next; /* Initialize current */
 while(current != lst->backSentinel)
{
     if(EQ(current->value,e))
           return 1;
   current=current->next;
}
```

   - **e.)**

```
void func_b(int arr[], int n) // n is size of arr
{
for (int a = 0; a < n; ++a)
       for (int b = 0; b < a; ++b)
            for (int c = 0; c < b; ++c)
                 arr[c] += arr[b];
}
```

Order (big-O): O(n3).

## ADTs

1. List the three levels of abstraction in the study of Data Structures?(3 pts)

Interface

Application

Implementation

2. What are the three main operations of a stack ADT ? (3 pts)

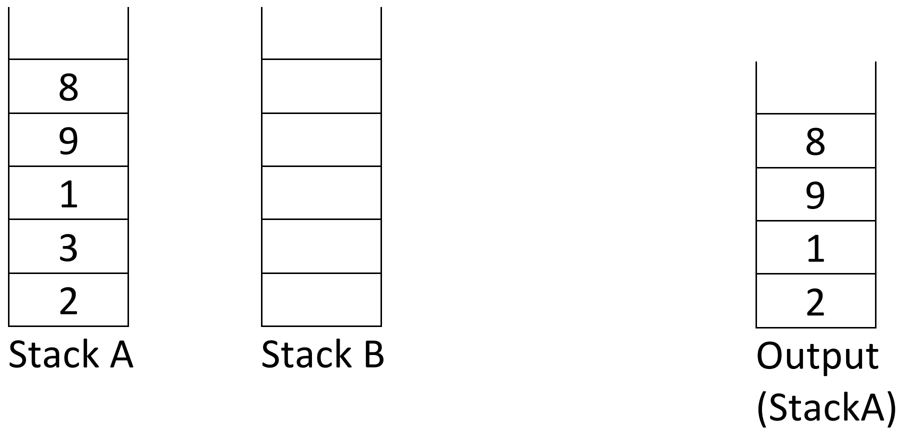
Push , pop, top (also called addLast, getLast, removeLast)

3. Describe the ordering property of a stack? (1 pt)

A stack is LIFO…last in first out

## Stack

1. Assume that there is a Stack “A” and you need to remove an element with value 3 from “A”. you can only use stack “B” as a temp memory. How many pop and push operations you should do to get the output? (7 pop and 6 push operation)



## Dynamic Array

1. Fill in the diagrams showing the cnt, cap, beg, and underlying data array after the following commands are carried out on a Dynamic Array Deque. The code for the arrDeque implementation that we discussed in class is included at the end of the exam if you need it. To receive partial credit, you should show the state of the data structure after each numbered line of code is executed. We have completed step 1 for you. (9 pts)

```
   struct dynArrDeque  d;
1)  initDynArrDeque(&d, 5);
2)  addBackArrDeque(&d, 3.0);
3)  addBackArrDeque(&d, 5.0);
4)  addBackArrDeque(&d, 1.0);
5)  removeFrontArrDeque(&d);
6)  addBackArrDeque(&d, 2.0);
7)  addFrontArrDeque(&d, 10.0);
8)  removeBackArrDeque(&d);
9)  removeFront(ArrDeque &d);
```

1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = 5 | cnt = 0 | beg= 0 |
| 0 | 1 | 2 | 3 | 4 | | | |

2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

3

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

5

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

6

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

7

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

8

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

9

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | cap = | cnt = | beg= |
| 0 | 1 | 2 | 3 | 4 | | | |

2. Write a function _dynArrayRemoveAt(…) which removes an element at a particular index in a dynamic array. (12 pts)

```
struct dynArray {
  TYPE * data;
  int size;
  int capacity;
}
```

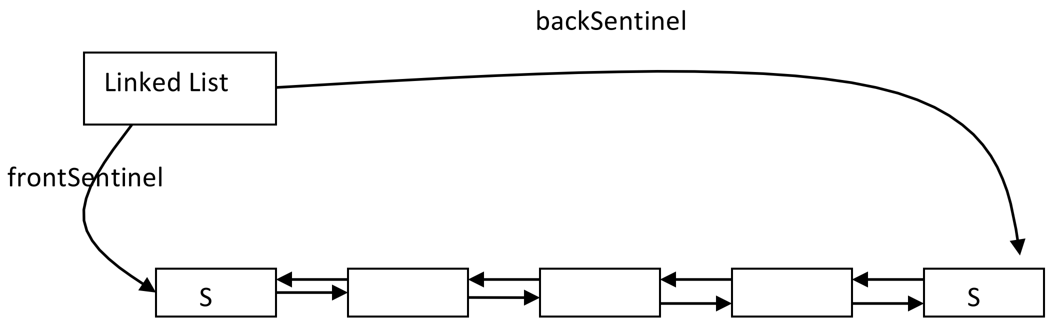
```
void dynArrayRemoveAt (struct DynArr * da, int index) {

int I;

assert( idx > 0);

assert(idx < da->size);

for (i = index; i < da->size-1; i++)

  {

    da->data[i] = da->data[i+1];

}

da->size--;

}
```

3. What is the algorithmic complexity (average case big-Oh) of dynArrayRemoveAt() from the previous question.

O(n)

## Link List:

1. Below is a depiction of a doubly-linked list with two sentinels.



```
struct DLink {
  TYPE value;
  struct DLink * next;
  struct DLink * prev;
};
```

```
struct linkedList {
  int size;
  struct DLink *frontSentinel;
  struct DLink *backSentinel;
};
```

a). Write a function printLinkedList( …) which runs through the list printing the value of all elements. 4 pts)

```
void  _printLinkedList(struct linkedList *list)

{
  struct DLink *cur;
cur  = list->frontSentinel->next;
while(cur != list->backSentinel)
  {
     printf("Value  = %d\n", cur->value);
    cur = cur->next;
  }
}
```

2. Assume we have two implementations of the deque and bag interfaces: the dynamic array deque (arrayDeque) and the doubly linked list deque (listDeque) Give the AVERAGE or EXPECTED case and WORST CASE execution time (big-oh) of the following functions (8 pts)

| | arrayDeque | | listdeque | |
|---|---|---|---|---|
| | AVE | WORST | AVE | WORST |
| add(Object) | 1+ | N | 1 | 1 |
| contains(Object) | N | N | N | N |
| addLast(Object) | 1+ | N | 1 | 1 |
| addFirst(Object) | 1+ | N | 1 | 1 |

## Iterator:

1. Write the Iterator functions hasNext() and next() for the linkedList. Assume linkedList is our doubly linked list with two sentinels (shown partially at the end of the exam). [10 points]

```
struct linkedListIterator {
      struct linkedList * lst;
      struct DLink * currentLink;
}

void linkedListIteratorInit (struct linkedList *lst,
                                  struct linkedListIterator * itr)
{
      itr->lst = lst;
      itr->currentLink = lst->frontSentinel;
}

int linkedListIteratorHasNext (struct linkedListIterator*itr) {

      if(itr->cur->next != itr->lst->backSentinel)
      {
            itr->cur = itr->cur->next;
            return 1;
      } else return 0

}

TYPE linkedListIteratorNext (struct linkedListIterator *itr) {

    Return (itr->cur->value)
}
```

## Ordered Array and Binary Search

Assume you have written the  binary search function that takes an array, count, and value, and returns an integer representing the location where val is either located or should be located:

**int _binarySearch(TYPE *data, int cnt, TYPE val);**

Write the `contains()` function for the DynArray implementation of a SortedBag that runs in O(log n) time.  You MUST make use of the **_binarySearch** function.  (6 pts)

```
int containsSortedArray(struct DynArr *da, TYPE val)
{

  int idx ;

  idx = _binarySearch(da->data, da->size, val);

  if(da->data[idx] == val)
         return 1;

  else return 0;
}
```

```
/* dynArray.h *
struct DynArr
{
      TYPE *data;        /* pointer to the data array */
      int size;          /* Number of elements in the array */
      int capacity;      /* capacity ofthe array */
};
typedef struct DynArr DynArr;

/* Dynamic Array Functions */
void initDynArr(DynArr *v, int capacity);
DynArr *newDynArr(int cap);

void freeDynArr(DynArr *v);
void deleteDynArr(DynArr *v);
int sizeDynArr(DynArr *v);
void addDynArr(DynArr *v, TYPE val);
TYPE getDynArr(DynArr *v, int pos);
void putDynArr(DynArr *v, int pos, TYPE val);
void swapDynArr(DynArr *v, int i, int  j);
void removeAtDynArr(DynArr *v, int idx);
```

```
/* Stack interface. */
int isEmptyDynArr(DynArr *v);
void pushDynArr(DynArr *v, TYPE val);
TYPE topDynArr(DynArr *v);
void popDynArr(DynArr *v);

/* Bag Interface */
int containsDynArr(DynArr *v, TYPE val);
void removeDynArr(DynArr *v, TYPE val);


/* Useful Internal Function - we defined this in our .c file */
void _dynArrSetCapacity(DynArr *v, int newCap);




/* Double Link*/
struct DLink {
      TYPE value;
      struct DLink * next;
      struct DLink * prev;
};

/* Double Linked List with Head and Tail Sentinels  */

struct linkedList{
      int size;
      struct DLink *frontSentinel;
      struct DLink *backSentinel;
};

/*
      initList
      param lst the linkedList
      pre: lst is not null
      post: lst size is 0
*/

void _initList (struct linkedList *lst) {
   /* create the sentinels */
   lst->frontSentinel = (struct DLink *) malloc(sizeof(struct DLink));
   lst->backSentinel = (struct DLink *) malloc (sizeof(struct DLink));
   assert (lst->frontSentinel != 0);
   assert(lst->backSentinel != 0);

   lst->frontSentinel->prev = 0;
   lst->frontSentinel->next = lst->backSentinel;

   lst->backSentinel->next = 0;
   lst->backSentinel->prev = lst->frontSentinel;

   lst->size = 0;
}
```

```
/*
 createList
 param: none
 pre: none
 post: frontSentinel and backSentinel reference sentinels
 */

struct linkedList *createLinkedList()
{
      struct linkedList *newList = malloc(sizeof(struct linkedList));
      _initList(newList);
      return(newList);
}
```