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 function that takes exactly - $2n^4 + 4n^3 - 6n \log n + 100$ steps has steps $O(n^4)$

b.) The following lines of code -

```
for(int i = 0; i*i < n; i ++)
{
    for(int j = 1; j*j < n*n; j++)
    {
        printf("i = , j = " i, j);
 }

}
```

$O(n * \sqrt{n})$

The outer loop will run while $i^2 < n$, or equivalently while $i < \sqrt{n}$. This means the outer loop will run $\sqrt{n}$ times.

The inner loop will run while $j^2 < n^2$, or equivalently while $j < n$. This means the inner loop will run n times (for each iteration of the outer loop).

The total number of iterations is therefore $n * \sqrt{n}$

c.)

```
int i = n ;
 while (i>1) {
 printf( "%d\n", i );
 i = floor( i/2) ;
 }
```

$O(\log(n))$

Time Complexity of a loop is considered as O(logn) if the loop variable is divided / multiplied by a constant amount.

**Please memorize the order of classes of functions

Notation	Name
$O(1)$	constant
$O(\log \log n)$	double logarithmic
$O(\log n)$	logarithmic
$O(n^c),\ 0 < c < 1$	fractional power
$O(n)$	linear
$O(n \log^* n)$	n log-star n
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, or quasilinear
$O(n^2)$	quadratic
$O(n^c),\ c > 1$	polynomial or algebraic
$L_n[\alpha, c],\ 0 < \alpha < 1 = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$	L-notation or sub-exponential
$O(c^n),\ c > 1$	exponential
$O(n!)$	factorial

ADTs

1. List the three levels of abstraction in the study of Data Structures?

   Interface

   Application

   Implementation

2) How is the memory representation of a linked list different from that of a Dynamic Array?)

   The linked list maintains a reference to a collection of elements of type link and allocates a new link every time a new element is added whereas dynamic array uses a fixed large block of memory.Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted but for linked list each new addition to the chain of links requires only a constant amount of work. Unlike a Dynamic Array, in linked list random access is not allowed. One needs to access elements sequentially starting from the very first node. Another diference is, extra memory space for a pointer (reference to another link) is required with each element of type link of the list.

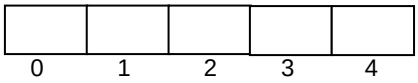
3) Mention five applications of stack.

1)Back and Forward Buttons in a Web Browser
2)Buffered Character Input
3)Checking Balanced Parenthesis
4)Conversion of infix to postfix
5)Evaluation of a postfix expression

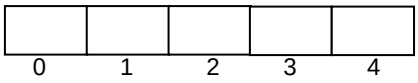
Dynamic Array

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.

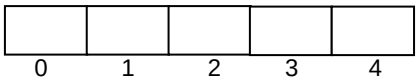
```
   struct dynArrDeque  d;
1)  initDynArrDeque(&d, 5);
2)  addBackArrDeque(&d, 3.0);
3)  addBackArrDeque(&d, 5.0);
4)  addBackArrDeque(&d, 1.0);
5)  removeFrontArrDeque(&d);
6)  addBackArrDeque(&d, 2.0);
7)  addFrontArrDeque(&d, 10.0);
8)  removeBackArrDeque(&d);
9)  removeFront(ArrDeque &d);
```

1 

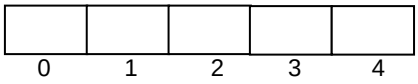
cap = 5 cnt = 0 beg= 0

2 

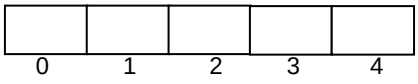
cap = cnt = beg=

3 

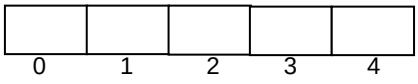
cap = cnt = beg=

4 

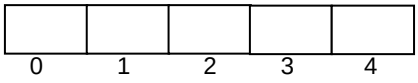
cap = cnt = beg=

5 

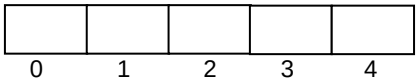
cap = cnt = beg=

6 

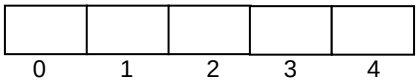
cap = cnt = beg=

7 

cap = cnt = beg=

8 

cap = cnt = beg=

9 

cap = cnt = beg=

Answer provided in the attached file.

2.Consider an empty dynamic array container with initial capacity 5. Suppose writing a new element to the array costs 1 unit, and copying a single element during reallocation also costs 1 unit. How many total units will it cost to call 11 consecutive push operations on this stack?

Add operation	***Cost***
Add 1	**1**
Add 2	**1**
Add 3	**1**
Add 4	**1**
Add 5	**1**
Add 6	**5 (copy) + 1**
Add 7	**1**
Add 8	**1**
Add 9	**1**
Add 10	**1**
Add 11	**10 (copy) + 1**
	Total cost : 26

2. Write the following functions _dynArrayRemoveAt(…) which removes an element at a particular index in a dynamic array and _dynArraySwap(….) which swaps two specified elements in the dynamic array . Do not call any other function with these functions. You must use assert() functions if required.

```
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--;

 }
```

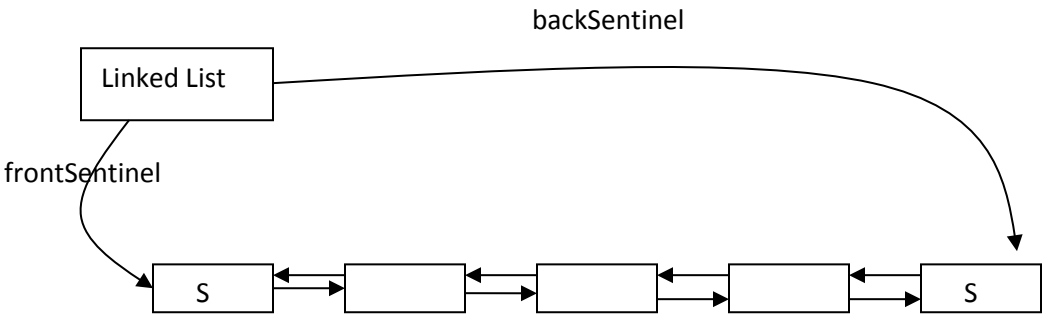
```
 void swapDynArr(struct DynArr * da, int i, int  j){
          TYPE temp;
       assert(i < da->size);
       assert(j < da->size);
       assert(i >= 0);
       assert(j >= 0);

       temp  = da->data[i];
       da->data[i] = da->data[j];
       da->data[j] = temp;
 }
```

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

O(n)

Linked List:

1. Below is a depiction of a deque ADT with 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(…) that prints the values of the links in the deque from front to back.

```
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;
 }
 }
```

b) Write a function _addBackLinkedList(……) which adds a new link with the given value to the back of the deque. Do not redirect to any other function.

```
static void _addBackLinkedList(struct linkedList* list, TYPE value){



}
```

```
struct DLink *newLink;
    newLink = malloc(sizeof(struct DLink) );
    assert(newLink != NULL);

    newLink->value = value;
    newLink->prev = list->backSentinel->prev;
    newLink->next = list->backSentinel;

    list->backSentinel->prev->next = newLink;
    list->backSentinel->prev = newLink;

    list->size++;
```

2.What is the big-O execution time of the print() function for a linked list (as described in the previous question)? O(n)

3.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

	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

4. Write down the worst-case computational complexity for adding to the back of a queue using a linkedlist with a first-link pointer and a last-link pointer, assuming the first link is the front. O(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).

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

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

int linkedListIteratorHasNext (struct linkedListIterator *itr)

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

}

TYPE linkedListIteratorNext (struct linkedListIterator *itr) {


          itr->cur = itr->cur->next;
        TYPE val = itr->cur->value;


               return val;

 }
```

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.

```
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);
}
```