Name: **Student ID:**

Sample questions from previous open-book midterm exams

I. A linked list is different from an array, how? (select all that apply)

- Linked list can store more types of data than an array can.
- Deletion in an array is slower than in a linked list.
- You can sort a linked list, but you cannot sort an array.
- Insertion in a linked list is slower than in an array.
- A linked list is dynamically sizable, while an array is fixed in size.

II. What is the most accurate running time in big-O of: 55 log n^2 + 20 n^{10} log n ?

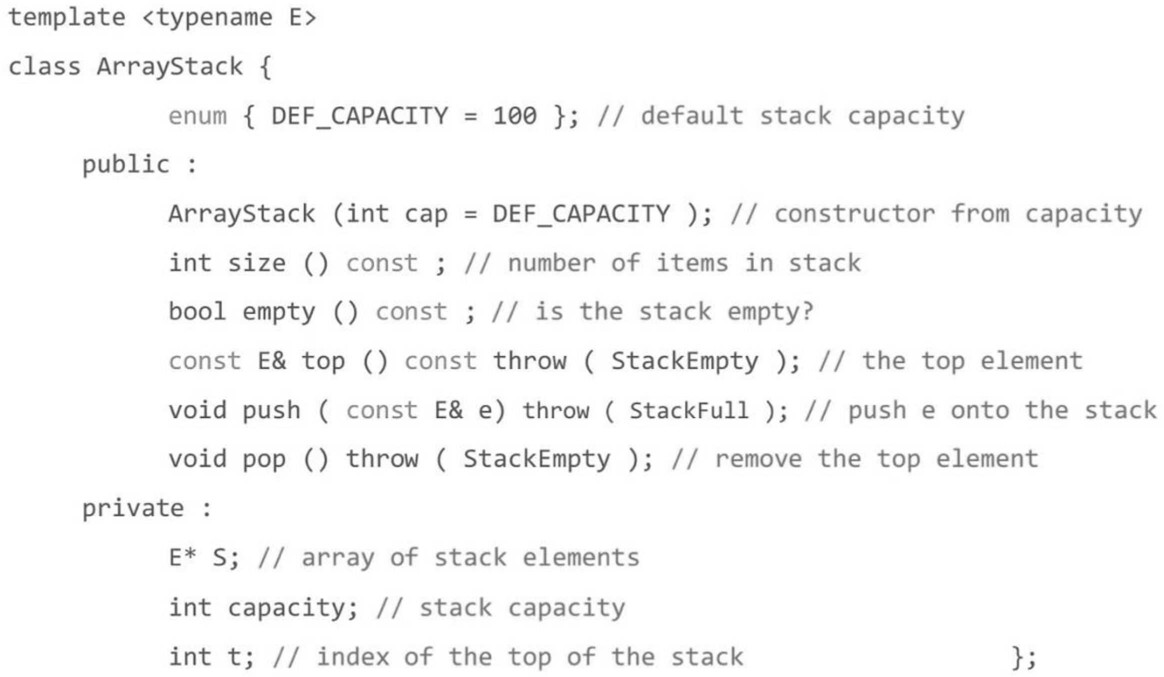
- $O(n^{10})$
- $O(\log n^2)$
- $O(\log n)$
- $O(n^{10} \log n)$

III. What is the running time of the following function? Justify your answer (show your work)

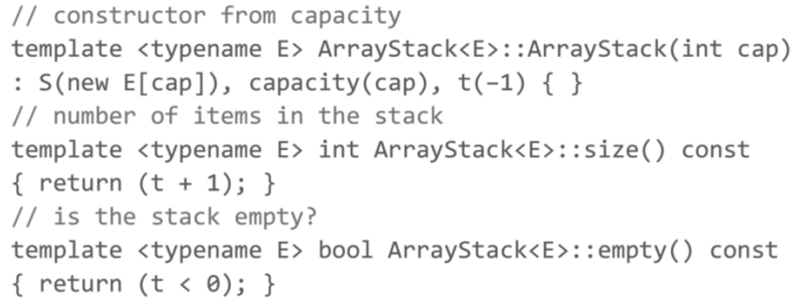
```
int example(int n) {
    if (n<=0){
       return n;}
    for (int i=1; i<=n; i++) {
        for (int j=1; j<=n; j++) {
            cout << i << "\n";
            break;
         }
     }
}
```

Name: **Student ID:**

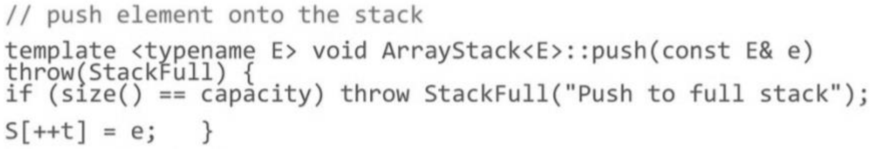
IV. Consider the following *array-based stack implementation:*



```
template <typename E>

class ArrayStack {

          enum { DEF_CAPACITY = 100 }; // default stack capacity

     public :

          ArrayStack (int cap = DEF_CAPACITY ); // constructor from capacity

          int size () const ; // number of items in stack

          bool empty () const ; // is the stack empty?

          const E& top () const throw ( StackEmpty ); // the top element

          void push ( const E& e) throw ( StackFull ); // push e onto the stack

          void pop () throw ( StackEmpty ); // remove the top element

     private :

          E* S; // array of stack elements

          int capacity; // stack capacity

          int t; // index of the top of the stack                   };
```



```
// constructor from capacity
template <typename E> ArrayStack<E>::ArrayStack(int cap)
: S(new E[cap]), capacity(cap), t(-1) { }
// number of items in the stack
template <typename E> int ArrayStack<E>::size() const
{ return (t + 1); }
// is the stack empty?
template <typename E> bool ArrayStack<E>::empty() const
{ return (t < 0); }
```



```
// push element onto the stack
template <typename E> void ArrayStack<E>::push(const E& e)
throw(StackFull) {
if (size() == capacity) throw StackFull("Push to full stack");
S[++t] = e;   }
```

a) ***Adjust the implementation*** *of* `push(const E& e)` *method so that it makes the capacity of the stack the double when it is found full and adds the element.*

b) ***Write the implementation*** *of* `reverseStack()` *that reverses the order of the elements within the stack.* ***Compute it's running time*** *in terms of big-O (justify).*

```
class ArrayStack {
      public:
            …
            void reverseStack() throw (StakeEmpty);
            …
};

// code here
```

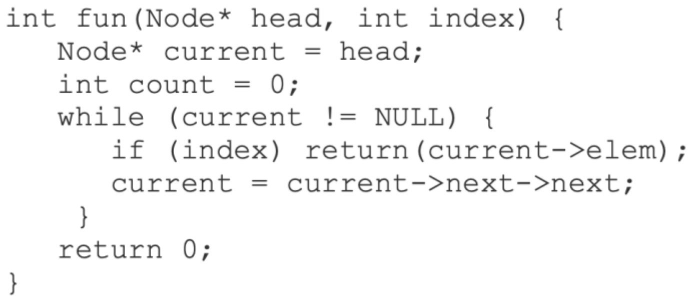
V. ***Consider the following Doubly linked list implementation:***

```
typedef string Elem;
class DNode {                           // doubly linked list node
  private:
    Elem elem;
    DNode* prev;
    DNode* next;
    friend class DLinkedList;
};

class DLinkedList {                     // doubly linked list
  public:
    DLinkedList();
    ~DLinkedList();
    bool empty() const;                 // is list empty?
    bool searchDL(DNode* headNode, string term);
  private:
    DNode* header;
    DNode* trailer;
};
```

Write the implementation of `searchDL`*, which checks whether a given term exists in the linked list or not <u>via recursion</u>.* ***What type*** *of recursion did you apply?* ***Compute it's running time*** *in terms of big-O (justify).*

```
// code here
```

VI. ***What is wrong with the following implementation of a function that returns the integer value saved within the Nth node in a linked list (if the index is for a non-existent node element, the function will return zero)? Identify the problem(s) and fix the code.***



```
int fun(Node* head, int index) {
   Node* current = head;
   int count = 0;
   while (current != NULL) {
      if (index) return(current->elem);
      current = current->next->next;
    }
   return 0;
}
```