Design and Analysis of Algorithms: Homework 1

1. (10 points) Rank the following functions by order of growth, from slowest-growing to fastest-growing. That is, find an arrangement $f_1, f_2, ...f_{13}$ of the following functions satisfying $f_1 \in O(f_2)$, $f_2 \in O(f_3)$, etc. *Hint:* Try applying the rules of Theorem 1.7.

$6n \log n$	2^{100}	$\log \log n$	$\log^2 n$	$2^{\log n}$
$3\sqrt{n}$	$n^{0.001}$	$n^2 \log n$	$5 \log n$	$5n$
	n^3	n^2	$n!$	

2. (10 points) Describe in a few words what the algorithm **Foo** does and what the algorithm **Bar** does. Analyze the worst-case running time of each algorithm and express it using "Big-Oh" notation.

Input: two integers, a and n
Output: ?

```
1: function FOO(a, n)
2:     k ← 0
3:     b ← 1
4:     while k < n do
5:         k ← k + 1
6:         b ← b * a
7:     return b
```

Input: two integers, a and n
Output: ?

```
1: function BAR(a, n)
2:     k ← n
3:     b ← 1
4:     c ← a
5:     while k > 0 do
6:         if k mod 2 = 0 then
7:             k ← k/2
8:             c ← c * c
9:         else
10:            k ← k − 1
11:            b ← b * c
12:    return b
```

3. (a) (4 points) You are given an initially empty stack and perform the following operations on it: *push*(B), *push*(A), *push*(T), *push*(I), *pop*(), *pop*(), *push*(Z), *push*(A), *pop*(), *push*(I), *push*(N), *push*(L), *pop*(), *push*(G), *push*(A), *push*(R), *push*(F), *pop*(), *pop*(). Show the contents of the stack after all operations have been performed and indicate where the top of the stack is.

   (b) (6 points) Describe how to implement two stacks using one array. The total number of elements in both stacks is limited by the array length; all stack operations (push, pop, size) should run in $O(1)$ time.

4. (a) (4 points) You are given an initially empty queue and perform the following operations on it: *enqueue*(B), *enqueue*(A), *enqueue*(T), *enqueue*(I), *dequeue*(), *dequeue*(), *enqueue*(Z), *enqueue*(A), *dequeue*(), *enqueue*(I), *enqueue*(N), *enqueue*(L), *dequeue*(), *enqueue*(G), *enqueue*(A), *enqueue*(R), *enqueue*(F), *dequeue*(), *dequeue*(). Show the contents of the queue after all operations have been performed and indicate where the front and end of the queue are.

   (b) (6 points) Describe in pseudo-code a linear-time algorithm for reversing a queue Q. To access the queue, you are only allowed to use the methods of a queue ADT. *Hint:* Consider using an auxiliary data structure.

5. In year 2264 the twenty-third starship came off the assembly lines at NASA. This starship was called the USS Enterprise. Unfortunately, the core libraries of the Enterprise were corrupted during an exploration mission. The only uncorrupted data structure left was a simple stack. A team of engineers set out to reimplement all other data structures in terms of stacks, and they started out with queues.

(a) (5 points) The following are parts of their original implementation of queue using two stacks (in_stack and out_stack). Analyze the worst-case running times of its **enqueue** and **dequeue** methods, and express them using "Big-Oh" notation.

```
1: function ENQUEUE(o)
2:     in_stack.push(o)
3:
4: function DEQUEUE()
5:     while not in_stack.isEmpty() do
6:         out_stack.push(in_stack.pop())
7:     if  out_stack.isEmpty() then
8:         throw QueueEmptyException
9:     return_obj ← out_stack.pop()
10:    while not out_stack.isEmpty() do
11:        in_stack.push(out_stack.pop())
12:    return return_obj
```

(b) (5 points) Later in the 23rd century, a new chief engineering officer named Montgomery Scott took over. He set out to optimize the old code. Thus a new implementation of a queue (still using two stacks) was born. What is the worst-case total running time of performing a series of $2n$ **enqueue** operations and n **dequeue** operations in an unspecified order? Express this using "Big-Oh" notation. *Hint:* Try using techniques presented in section 1.5.

```
1: function ENQUEUE(o)
2:     in_stack.push(o)
3:
4: function DEQUEUE()
5:     if out_stack.isEmpty() then
6:         while not in_stack.isEmpty() do
7:             out_stack.push(in_stack.pop())
8:     if out_stack.isEmpty() then
9:         throw QueueEmptyException
10:    return out_stack.pop()
```

6. (5 points) A program written by a graduate student uses an implementation of the sequence ADT as its main component. It uses only **atRank**, **insertAtRank**, and **remove** operations in some unspecified order. It is known that this program performs n^2 **atRank** operations, $5n$ **insertAtRank** operations, and n **remove** operations. Which implementation of the sequence ADT should the student use in the interest of efficiency: the array-based one or the one that uses a doubly-linked list? Explain.

7. (a) (4 points) Draw a single binary tree T such that each of the following properties holds:
   - each **internal** node of T stores a single character
   - a **preorder** traversal of T yields COMPILE, and
   - a **inorder** traversal of T yields PMIOLCE.

   (b) (6 points) Give an $O(n)$-time algorithm for computing the depth of each node of a tree T, where n is the number of nodes of T. Assume the existence of methods setDepth(v,d) and getDepth(v) that run in $O(1)$-time.

8. (5 points) Design an algorithm, inorderNext(v), which returns the node visited after node v in an inorder traversal of binary tree T of size n. Analyze its worst-case running time. Your algorithm should avoid performing traversals of the entire tree.