Design and Analysis of Algorithms
Homework 1

Clearly number your solution to each problem. Staple your solutions and bring them to class on the due date. Express your algorithms in pseudo-code when directed. Always provide justification for your answer when asked to give the running time of an algorithm. Be brief and concise, and draw pictures where appropriate.

1. (a) Show that $2(n+4)^6 + 2n^2 \log n$ is $O(n^6)$. *Hint:* Try applying the rules of Theorem 1.7.

   (b) Rank the following functions by order of growth. 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.

$$\begin{array}{ccccc} 6n\log n & 2^{100} & \log\log n & \log^2 n & 2^{\log n} \\ 3\sqrt{n} & n^{0.001} & n^2\log n & 5\log n & 5n \\ & n^3 & n^2 & n! & \end{array}$$

   (c) Algorithm A executes $2n\log_2 n$ primititve operations, while algorithm B executes n^2 operations. Determine the minimum integer value n_0 such that A executes fewer operations than B for $n \geq n_0$.

2. (a) What does the following algorithm **Foo** do? Analyze its worst-case running time, and express it using "Big-Oh" notation.

   (b) What does the following algorithm **Bar** do? Analyze its worst-case running time, and express it using "Big-Oh" notation.

Input: two integers, a and n
Output: ?

```
function Foo(a, n)
    k ← 0
    b ← 1
    while k < n do
        k ← k + 1
        b ← b * a
    return b
```

Input: two integers, a and n
Output: ?

```
function Bar(a, n)
    k ← n
    b ← 1
    c ← a
    while k > 0 do
        if k mod 2 = 0 then
            k ← k/2
            c ← c * c
        else
            k ← k − 1
            b ← b * c
    return b
```

3. (a) Describe the output of the following series of stack operations on a single, intially empty stack: *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*()

   (b) Describe the output of the following series of queue operations on a single, intiailly empty queue: *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*()

(c) 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.

(d) 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 should run in $O(1)$ time.

4. 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) 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.

```
function ENQUEUE(o)
    in_stack.push(o)

function DEQUEUE()
    while not in_stack.isEmpty() do
        out_stack.push(in_stack.pop())
    if out_stack.isEmpty() then
        throw QueueEmptyException
    return_obj ← out_stack.pop()
    while not out_stack.isEmpty() do
        in_stack.push(out_stack.pop())
    return return_obj
```

(b) 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 complexity 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.

```
function ENQUEUE(o)
    in_stack.push(o)

function DEQUEUE()
    if out_stack.isEmpty() then
        while not in_stack.isEmpty() do
            out_stack.push(in_stack.pop())
    if out_stack.isEmpty() then
        throw QueueEmptyException
    return out_stack.pop()
```

5. 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.