QUEENS COLLEGE
CSCI 313
Instructor: Krishna Mahavadi

Department of Computer Science
Practice problems on O, Omega, Theta and the Master theorem

These problems were given on exams for this course. Some older problems did not make use of generics in Java, but generic implementations are now required in this course.

Problem 1 Give useful Θ estimates for the following functions $t(n)$.

(a) $t(n) = 5log_2(n^2) + (log_2(n))^2 + log_4(n) + (log_2(100))^3$.

(b) $t(n)$ satisfies $t(n) = 2t(n/2) + n$.

(c) $t(n)$ satisfies $t(n) = 4t(n/3) + n$.

(d) $t(n)$ is the running time of the following function:

```
public static void shuffle(int []x, int a, int b, int n) {
   for (int i = 0; i < n; i+=2) {
      int temp = x[a + i];
      x[a + i] = x[b + i];
      x[b + i] = temp;
   }
}
```

(e) $t(n)$ is the running time of the following function that calls shuffle from (d):

```
public static void multiShuffle(int []x, int a, int n) {
   if (n == 0) return;
   multiShuffle(x, a, n/2);
   multiShuffle(x, a + n/4, n/2);
   multiShuffle(x, a + n/2, n/2);
   shuffle(x, a, a + n/2, n/2);
}
```

Problem 2 Give useful O-estimates of the run times of the following methods:

(a) The method $addHead$ for a singly linked list that has size n.

(b) An efficient method to calculate the power x^n (consider the run time as a function of n, the time should be considered as being proportional to the total number of additions, subtractions, multiplications, and divisions performed).

(c) An efficient method to sort an array of n numbers into order.

For (d) and (e), consider the following recursive function, in which A represents an integer constant:

```
int f(int n) {
   if (n <= 0) return 1;
   int ans = f(n/2) * 2;
   for (int i = 1; i<= n; i++)
      for (int j = 1; j <= n; j++)
         ans += i / j;
   for (int k = 1; k < A; k++)
         ans -= f(n/2 - k);
   return ans;
}
```

(d) In the case where $A = 3$ estimate the run time of $f(n)$.

(e) In the case where $A = 4$ estimate the run time of $f(n)$.

Problem 3 Give useful O estimates for the run times of the following methods.

(a) *removeMin* for a *PriorityQueue* storing n items in a heap implementation.

(b) *preOrder* for a general *Tree* storing n items.

(c) *get* for a chained *HashTable* storing n items with load factor λ.

(d) A recursive method f that processes n input items by: sorting the items (efficiently), makes two recursive calls to process $n/2$ items, computes the products of all pairs of input items and finally makes two further recursive calls to process $n/2$ items.