

1. Consider the following algorithm, which determines whether $1 + 2 + \dots + n$ is a perfect square:

```
bool test(int n)
{
    for(int i = 1; i <= n; i++)
    {
        // Compute sum = 1 + 2 + ... + n
        int sum = 0;
        for(int k = 1; k <= n; k++)
            sum = sum + k;

        // Compare sum and i^2
        if (sum == i*i)
            return true;
    }
    return false;
}
```

(a) (10 Points) In terms of n , what is the worst case time complexity of this algorithm? (Justify your answer!)

(b) (5 Points) Modify the algorithm to improve its complexity (it still has to compute the same thing!) and analyze the complexity of your improved algorithm.

2. (5 Points) MIDN Jones analyzes an algorithm and determines that its best case complexity is $O(n^2)$. MIDN Smith analyzes the same algorithm and determines that its best case complexity is $\Theta(n \lg n)$. Could they both be right, or must at least one of them be wrong? Justify your answer!

3. (10 Points) Suppose A is an array of n positive integers the largest of which is k , and suppose that `mystery(M)`, where M is an integer, is a function with worst case time complexity $O(M^2)$ and $\Omega(M \lg M)$. Consider the following algorithm:

```
int sum = 0;
for(int i = 0; i < n; i++)
{
    int x = mystery(A[i]);
    sum += x;
}
cout << sum << endl;
```

What can you tell me about the worst case time required by the following algorithm in terms of n and k ? I'll start you off: The worst-case time is $O(2^{nk})$ and $\Omega(1)$. Improve on this by giving me the most precise upper and lower bounds you can! Justify your answers!

4. (10 Points) Give a recurrence relation for the worst case time complexity for the algorithm below. Justify your answer!

```
int num_zeros(array A, int i, int j)
{
    if (i > j) return 0;
    if (i == j)
        if (A[i] == 0) return 1;
        else return 0;

    int s1 = (j - i)/3, s2 = (2*(j - i))/3;

    int z1 = num_zeros(A,i,s1);
    int z2 = num_zeros(A,s1+1,s2);
    int z3 = num_zeros(A,s2+1,j);

    return z1 + z2 + z3;
}
```

5. (10 Points) The Berlekamp-Massey algorithm for polynomial factorization has worst case $O(2^n)$, where n is the degree of the polynomial to factor. The LLL algorithm has worst case $\Theta(n^5)$. Now, n^5 grows *much* more slowly than 2^n , yet most any system that needs polynomial factorization uses the Berlekamp-Massey algorithm. Why? Give three possible good reasons.

6. (10 Points) Consider the following algorithm, which takes a size n array A containing the numbers $1, \dots, n$, but not necessarily in sorted order.

```

double S = 0;
for(int i = 0; i < n; ++i)
{
    // Compute A[i]th harmonic number, i.e. 1/1 + 1/2 + ... + 1/A[i]
    double H=0;
    for(int k = 1; k <= A[i]; ++k)
        H += 1.0/double(k);

    // Compute floor(lg(i+1))
    int L = 0;
    for(int m = i + 1; m > 1; m = m/2)
        L++;

    // Add floor(lg(i+1))/A[i]th harmonic number
    S += L/H;
}

```

- (a) (10 Points) Analyze the worst case complexity of this algorithm, and justify your answer thoroughly! The more precise the answer you're able to give, the more credit you'll get.

- (b) (5 Points) Improve this algorithm and give the worst case complexity for your improved algorithm. Your improved algorithm must compute the the same thing! I.e.

$$\sum_{i=0}^{n-1} \frac{\lfloor \lg(i+1) \rfloor}{1/1 + 1/2 + \dots + 1/A[i]}$$