1. **[5 points]** When analyzing the performance of algorithms, what is the *doubling hypothesis*? Estimate the runtime growth rate function form and the specific values for:

Problem Size	Time (in arbitrary units)
16	33
32	132
64	396
128	1584

   1) Doubling hypothesis is a quick to estimate runtime growth rate, specifically the base b in a power-law relationship (3 points). It is obtained by calculating the ratio between $a(2N)^b$ and $a(N)^b$ and then compute $b = log_2 ratio$. While it can not identify the logarithmic factor a.
   2) Ratio1=(132/33)=4; ratio2=(396/132)=3; ratio3=(1583/396)=4 (1 point), so the ratio is converged at 4. $b = log_2 ratio = log_2 4$=2. Then the growth rate function can be denoted as $O(n^2)$ (1 point).

2. **[3+1 points]** Propose (i.e., write pseudo-code) an "efficient" solution for the 3-sum problem? What is the best case computational complexity and compare it to the worst case computational complexity?
   1) (3 points) Sort the N numbers, and then for each pair of number a[i] and a[j], binary search (2 points) for (a[i]+a[j])
   2) (1 point) The key is to identify the best case and worst case for the binary search in the algorithm. For the binary search, worst case is $O(\log n)$, and best case is $O(1)$. Therefore, for the 3-sum problem, the worst case is $O(n^2 \log n)$, and best case is $O(n^2)$.

3. **[5 points]** Here is the Pseudocode for the solution to the Towers of Hanoi problem. Trace the (**order and value**) function calls.

```
solveTowers(count, source, destination, spare)
   if (count is 1)
      Move a disk directly from source to destination
   else
   {
      solveTowers(count - 1, source, spare, destination)
      solveTowers(1, source, destination, spare)
      solveTowers(count - 1, spare, destination, source)
   }
```

   1) Label the pegs A, B, C; let n be the total number of disks; number the disks from 1 (smallest, topmost) to n (largest, bottom-most)
   2) Move n-1 disks from source to auxiliary, so they are out of the way; move the nth disk from source to target; move the n-1 disks so that we left on auxiliary onto target.
   3) http://interactivepython.org/runestone/static/pythonds/Recursion/TowerofHanoi.html
   4)
```
1. count 1 moving disk from A to B
2. count 2 moving disk from A to C
3. count 1 moving disk from B to C
4. count 3 moving disk from A to B
5. count 1 moving disk from C to A
6. count 2 moving disk from C to B
7. count 1 moving disk from A to B
```

4. **[1+1+1+3 points]** **(i)** If you are told that for A(x), if you increase x by a factor of 10, the value of A goes up by a factor of 1000, what can you say about the functional form of A(x)? (**ii**) Can low-order terms in an algorithm's growth-rate function can be ignored? Why? (**iii**) What is the runtime in the worst case, for binary search? (**iv**) Arrange the following functions based upon increasing growth-rate function (i.e. f(A) > f(B), implies A has a higher growth-rate function). NOTE: There maybe functions that are equal to each other.

(i) N^2 (ii) 2^N (iii) $N*\log_2 N$ (iv) $N*\log_3 N$ (v) $\log_3 N$ (vi) N (vii) N^3

1) (1 point) $A(x) = x^3$
2) (1 point) Intuitively, the lower-order terms of an asymptotically positive function can be ignored in determining asymptotically tight bounds because they are insignificant for large n. A tiny fraction of the highest-order term is enough to dominate the lower-order terms.
3) (1 point) Log(N)
4) (3 points) $2^N > N^3 > N^2 > N * log_2 N > N * log_3 N > N > log_3 N$