Q1:

a)

The statement is false.

since

consider when , function , the statement is false.

b)

The statement is true.

Assume satisfies , then

for some positive constants and . Then clearly

Hence

Where , .

By definition, .

c)

The statement is false.

Assume satisfies , then

So

Q2:

a)

Since , by master theorem,

b)

By drawing the recursion tree,

At the same time,

Therefore .

c)

So

d)

Where is infinitely large.

Q3:

The worst case is that it is not a prime, so it goes all of the loops.

For the outer loop, starts from 2, increment by 1 each time, and ends when .

So , outer loop runs times.

For the inner loop, starts from 2, increment by 1 each time, and ends when .

The worst case runtime

Q4:

1: For all , compute and store the sum into a tree T.

2: For all , compute , and search the sum in tree T. Return true if the sum is found. If all sums are not found in the tree, return false.

COMMON\_SUM(A, B, n){

//A, B are int arrays of size n.

//T is an empty tree.

Tree T;

for (int x = 0; i < n; ++x){

for (int y = 0; i < n; ++y){

T.insert(A[x] + A[y]);

}

}

for (int x = 0; i < n; ++x){

for (int y = 0; i < n; ++y){

if (T.search(A[x] + A[y])){

return true;

}

}

}

return false;

}

This is a brute force method. The first loop saves every possible sum of elements form array A. In the second loop, for each sum we search it in the previous saved data base: if it is a match, the sum appeared before, so there are two numbers in A sum to the same number as the sum from array B.

The first big loop runs times. Inside the loop, the insert takes time, so the insertion takes time. Similarly for the second loop, it runs times, each takes time for searching in a tree. In total the algorithm takes time.