Q1

(a)

First, we use two *for* loops in order to go through $n(n-1)/2$ pairs $(A[k], A[M]), k < m$, and store $sum(A[k]^2, A[M]^2)$ for all $1 \leq k \leq m \leq n$, in a new array B. Eventually, array B has the size of $n(n-1)/2$.
Next, we sort the array B – we can do this in $O(N \log N) N = n(n-1)/2$ which is equal to $O(n^2 \log n)$ in worst case, for example, using Merge Sort. After that, traverse the sorted array and we can determine whether there exists such a number.

(b)

In this case, we take a similar approach as in (a), except using a hash map to check, each insertion and lookup takes $O(1)$ expected time.
Similar to (a), we replace the array B to a hash map where the key is the $sum(A[k]^2, A[M]^2)$ and the corresponding value is the time the sum appears.
After that, we check whether there exists a key with value 2 in $O(1)$, if so this key is the result, otherwise, there exists not such a number.
In conclusion, this algorithm runs in the expected time of $O(n^2)$.

Q2

This is done by a "double QuickSort" as follows. Pick a nut adn use it as a pivot to split the bolts into three groups: those for the nuts was too large, those for the nuts was fitted and those for the nuts was too small. Then pick a bolt which is fitted and let this bolt try all the nuts, splitting the nuts in three groups as well: nuts that are too small, nuts that fit this bolt and the nuts which were too large. Continue this process with the first group of nuts and first group of bolts and then also the third group of nuts with the third group of bolts.
This algorithm runs in the expected time $O(n \log n)$.

Q3

As the question mentioned, 1024 apples which equals to 2^{10}, we can assume we have a complete binary tree with 2^{10} leaves and $2^{10} - 1$ internal nodes and of depth 10. We place all apples at the leaves, compare each pair and "promote" the heavier apple to the upper level and proceed in such a way till you reach the root of the tree, which will contain the heaviest apple. Clearly, each internal node is a result of one comparison and there are $2^{10} - 1$ many nodes thus also the same number of comparison so far.

And the second heaviest apple must be among in the apples which were compared with the heaviest apple along the way. There are 10 apples so finding the heaviest among them will take $10 - 1 = 9$comparisons by brute force. Note that the worst case is that at the leave level the first and the second heaviest apples were compared, so the second heaviest had been left on the bottom level. In total this is exactly at most $2^{10} + 10 - 2 = 1032$ weightings.

Q4

As the question mentioned, each square has n^2 trees and total $16n^2$ trees in orchard. And we already have a map with the number of apples on each tree.
First, assume we have a Sum Matrix which $sum[i][j]$ equals to the sum of apples from left top$(0,0)$ to right bottom (i,j) trees which called 2-dimension prefix sum. We can use 2 *for* loops to handle this in $O(n^2)$.
Then, we have $(3n+1)^2$ possible squares and for each square we compute the number of apples in this square by $sum[i][j] - sum[i-n][j] - sum[i][j-n] + sum[i-n][j-n]$ in $O(n^2)$.
Finally, we can find the square contains the largest number of apples in $O(n^2) + O(n^2) = O(n^2)$.

Q5

(a)

First, we can get $g(n) = 2\log_2^{(n^{\log_2^n})} = 2\log_2^n * \log_2^n = 2(\log_2^n)^2$
Second, taking $c = 1$: $0 \le (\log_2^n)^2 \le 2(\log_2^n)^2$ for all $n \ge 1$. Which is $f(n) = O(g(n))$
Also, taking $c = \frac{1}{2}$, $c = 1$: $0 \le c2(\log_2^n)^2 \le (\log_2^n)^2$ for all $n \ge 1$. Which is $f(n) = \Omega(g(n))$
Therefore, $f(n) = \theta(g(n))$

(b)

$f(n) = n^{10}$ $\quad$ $g(n) = 2^{\sqrt[10]{n}}$
We want to show that $f(n) = O(g(n))$, which means that we have to show that $f(n) \le cg(n)$ for some positive c and all sufficiently large n. But, since the log function is monotonically increasing, this will hold just in case

$$10\log n \le \log c + \sqrt[10]{n}$$

We now taking $c = 1$ then it is enough to show that

$$\frac{10\log n}{\sqrt[10]{n}} \le 1$$

for sufficiently large n. To this end we use the L'Hoˆpital's to compute the limit

$$\lim_{n\to\infty} \frac{10\log n}{\sqrt[10]{n}} = \lim_{n\to\infty} \frac{(10\log n)'}{(\sqrt[10]{n})'} = \frac{100}{\ln 2} \lim_{n\to\infty} \frac{1}{\sqrt[10]{n}} = 0$$

Since $\lim_{n\to\infty} \frac{10\log n}{\sqrt[10]{n}} = 0$ then, for sufficiently large n we will have $\frac{10\log n}{\sqrt[10]{n}} \le 1$.
So, $f(n) = O(g(n))$.

(c)

Just note that $1 + (-1)^n$ cycles with one period equal to $\{2, 0\}$. Thus, for all even numbers for n we have $1+(-1)^n = 2$ and for all odd numbers for n we have $1+(-1)^n = 0$. Thus for any fixed constant $c > 0$ for all even numbers n eventually $n^{1+(-1)^n} = n^2 > n$ and for all odd numbers n eventually $n^{1+(-1)^n} = 1 < c * n$ when $n > 0$.
Thus, neither $f(n) = O(g(n))$ nor $f(n) = \Omega(g(n))$.