Make sure you understand the following argument, and you should be able to give similar arguments for other combination of at-most/at-least, combined with and/or and big/not-big and sour/not-sour.

Let B = subset of big fruits, S = subset of sour fruits, b = |B|, and s = |S|.
Let -S denote the complement of S (i.e., subset of fruits that are not sour),
B ^ -S denote the intersection of B and -S corresponding to the fruits that
are big and non-sour, and B U -S denote the union of B and -S corresponding
to the fruits that are big or non-sour.

Maximizing B ^ -S means maximizing overlap of B and -S. Because |B| = b and
|-S| = f-s, maximum overlap is of size min(b, f-s). Clearly, it cannot be more
than either of b and f-s and hence cannot be more than min(b, f-s).

For f = 10, b = 6, and s = 7, we get max |B ^ -S| = min(6, 10-7) = 3. If we had
s = 2, then max |B ^ -S| would be min(6, 10-2) = 6. Indeed, we could fit all 6
big fruits within 8 non-sour fruits.

In conclusion, there are at most min(b, f-s) fruits that are big and not sour.

What is the minimum size of B ^ -S? This requires keeping B as much as possible
inside S, that is maximizing B ^ S. From above, this is min(b, s). Note that
by subtracting the size of B ^ S from that of B we get the size of B ^ -S. Thus,
there are at least b - min(b, s) = b + max(-b, -s) = max(b-b, b-s) = max(0, b-s)
fruits that big and not sour.

For f = 10, b = 6, and s = 7, we have at least max(0, 6-7) = 0 fruits that are
big and not sour. Because we could fit all big fruits within the sour fruits, in this case
there are no fruits that are big and not sour. If s were 2, then there would be
at least max(0, 6-2) = 4 fruits that are big and not sour. Indeed, we could have
at most 2 fruits that are big and sour and hence at least 4 big and non-sour
fruits.
~