1) 1000c
2) 999c'+c
3) a) 1c b) 1
4) Same as 3
5) a) We're going to assume the relation R is sorted on the compound primary key in the order given (c,b,a) (a common assumption). Then, given a value of c, we have 20 blocks holding that value (1000000/50 tuples and 1000 tuples/block). Now here's the trick: in part b) we calculate that there can only be 100 tuples with a fixed value of both b and c. These can all fit within one block. Hence, given a *c* value and a *b* value, the B tree lookup will always yield the same block pointer, given any value of *a*, because these must necessarily be less than 100. Hence the answer to this also is 1! Apparently, having a compound primary key (c,b,a) vs. having a compound primary key like (c,b) doesn't have a performance impact for this type of query.
b) 1000000/(200*50)=100
6) a) 10,000/(20*10)=50
   b) 10,000*(1/20+1/10-1/(20*10))=10,000*29/200=1450
* I prefer to use the formula P(A or B)= P(A)+P(B)-P(A AND B). It's equivalent to the formula you got in the lecture notes (can you prove it? not required, but useful technique)

7) 10 million
8) 25 million
9) cross product: 10 million
10) 250 million: Be careful about this; the formula for calculating multi-join estimates is the product of all the T's, divided by the product of all the V's for the attributes that appear *at least* twice among all the relations **except the least V** for that attribute. Note that this is not the same as dividing by the max's of all the V's! It is the same in this case because attributes appear at most twice, but if any attribute had occurred more than twice (for example, if we had included W in the join as well) dividing by the max's would have given wrong answers (why is the formula above correct? think about it)
11) 10,000
12) 1 million