(a) The statement is false. The inequality $b_i > rt_i$ only means the stream cannot be first. For example, if $t_1 = t_2 = 1$ and $b_1 = r + 1$, and $b_2 = r - 1$, than stream 2 can go first, followed by stream 1.

(b) There are many correct ways to solve this. We will list here the main versions.

Version 1. Sort by "stream rate". Rate of stream i is $r_i = b_i/t_i$. Send the streams in increasing order of rates. Assume for simplicity of notation that the streams are given in this order.

Check if the total rate $\sum_{i=1}^{n} b_i \leq r \sum_{i=1}^{n} t_i$ holds. We claim the following

(1) *If this test fails than no ordering produces a feasible schedule. If the test succeeds, then we claim the above order gives a feasible schedule.*

Proof. If the test fails that no order can produce a feasible schedule, as in a total time of $\sum_{i=1}^{n} t_i$ we need to send $\sum_{i=1}^{n} b_i$ no matter what way we order it.

We claim that if the above sorted order sends too much for any initial time period $[0, t]$ than the above test will also fail. To see this consider a time t, and let i be the stream sent during the last time period. If the rate of stream i is at most r (i.e., $r_i \leq r$) then all streams sent so far have a rate at most r, so the total send is at most rt, contradicting the assumption that we sent too much in t time. So we must have $r_i > r$. However, streams are ordered in increasing rate, so in any time step after t we will also send at least r bits, and hence, the total rate at the end of all streams will also violate the "average rate at most r" rule. ■

The running time is $O(n)$. The problem only asked to decide if an ordering exists, and that is done by testing the if the one inequality $\sum_{i=1}^{n} b_i \leq r \sum_{i=1}^{n} t_i$ holds. If we also want to output the ordering we need to spend $O(n \log n)$ time sorting rates. It is also okay to test if the above ordering is feasible after each job (taking $O(n)$ time).

Version 2. Prove that sorting by rates is an optimal schedule by a "greedy stays ahead" argument. The key here is to state in what way the schedule is optimal, and in what way the greedy algorithm stays ahead.

(2) *In the above greedy schedule, after any time t the amount of data transmitted in the first t time steps is as low as possible.*

Proof. Each of the t_i seconds of the schedule transmitting the stream i will have transmission rate of r_i. For any t, the first t seconds are the t lowest transmission rates, so the first t seconds send the lowest total number of bits. If this schedule violates the bit-rate requirements after t seconds, then all other schedules will also violate the requirement as they send at least as many bits.

[1] ex232.234.783

An alternate way of phrasing this argument is to talk about the slack: allowed rate of rt minus the number of bits sent till time t. Using this notion, *the greedy schedule is ahead as for any t, the first t seconds have the highest total slack possible for any schedule.* This is true for the same reason as used above: the schedule sends the the t lowest transmission rates in the first t seconds. The schedule violates the bit-rate requirements after t seconds, if it has negative slack after t seconds, and then all other schedules will also violate the requirement as they have at most as much slack as this schedule. ■

Version 3. Prove that sorting by rates is an optimal schedule by an "exchange argument".

Assume there is a feasible schedule $\mathcal{O}$, and let the algorithm's schedule be $\mathcal{A}$. We say that two jobs are inverted if they occur in different order in $\mathcal{O}$ and in $\mathcal{A}$. As in earlier exchange arguments in the text, we know that if the two orders are different, then there are two adjacent jobs i and j in $\mathcal{O}$ (say i immediately follows j) that are inverted. We need to argue that if $\mathcal{O}$ is a feasible schedule, and i and j are inverted, than the schedule $\mathcal{O}'$ obtained by swapping i and j is also feasible. Let T be the time j starts in $\mathcal{O}$, and assume the schedule $\mathcal{O}$ sends B bits in the first T seconds. The only times when the total amount sent so far is affected by the swap are the times in the range $[T, T + t_j + t_i]$. Let $T + t$ be such a time. Assume that in the schedule $\mathcal{O}$ we send b_o bits during these t seconds. The schedule $\mathcal{O}$ is feasible, and so $B + b_o \leq r(T + t)$. By the ordering used of our algorithm (and the fact that the jobs i and j were inverted), the number of bits sent by the same t seconds in the swapped schedule $\mathcal{O}'$ is $b' \leq b_o$. Therefore, the new schedule satisfies $B + b' \leq B + b_o \leq r(T + t)$. Swapping a pair of adjacent inverted jobs decreases the number of inversions and keeps the schedule feasible. So we can repeatedly swap adjacent inverted jobs until the schedule $\mathcal{O}$ gets converted to the schedule of the greedy algorithm. This proves that the greedy algorithm produces a feasible schedule.

Version 4 In fact, we do not need to sort at all to produce a feasible schedule. For each stream i compute its slack $s_i = rt_i - b_i$. We claim that if a feasible schedule exists, then any order that starts will all steams that have positive slack is feasible. To see why, observe that an ordering is feasible if the sum of slacks is non-negative for any initial segment of the order. Starting with all streams of positive slack creates the highest possible total slack before we start adding the jobs with negative slacks. This observation allows us to create the feasible ordering in $O(n)$ time without sorting, by simply computing the sign of the slack for each stream.

Note that this argument also shows that sorting steams in decreasing in order of slacks works too, as this also orders streams with positive slack before those with negative slack.

Finally, note that one ordering that does not always work is to order steams in increasing order of bits b_i.