Question 3

Algorithm idea:

Assume I have a dictionary input of n contestants (keys) and their swimming time, biking time and running time (values). As question description says, we can only one contestant in pool throughout the game, the total time spent in swimming had to be fixed which I just ignore this fact while scheduling. Unlike swimming, there can be more than 1 contestant simultaneously in biking and running. So in order to minimize the completion time. First, I generate a new dictionary such that for each contestant (key) compute the sum of his/her biking time and running time (value). Second, sort the new dictionary by value from large to small number. Lastly just output the key as answer schedule.

Algorithm detail:

Assume I have a dictionary in a format as contestant1: (swim, bike, run)

newDict = { }

answer = [ ]

For contestant in givenDict:

bpr = givenDict[contestant][1] + givenDict[contestant][2] # bike+run

newDict[contestant] = bpr

#above I obtain a dictionary of contestant and its corresponding biking+running

newDict = sorted(newDict sort by value)

reverse(newDict) get a dictionary sorted by value from large number to small

for contestant in newDict:

answer.append(contestant)

return answer

Proof of Correctness idea:

Assume I schedule n contestants from lowest time spend in biking and running to highest. I will always need to wait until the last person finish all three because the last person is the last person begin bike and run and he spent the most amount of time in biking and running. But with my algorithm, the person who spent the most amount of time can be finish before other contestants since this person can finish his/her running and biking while other contestant swimming. Following this idea, all people who has more biking and running time would able to finish while other contestant swim. Therefore, in order to minimize the completion time, I put all contestants in descending order.

Proof detail:

If I schedule n contestants in increasing order, the total completion time will be ∑ swimming time + (waitTime(time spent for contestant before him/her to finish swim) of the last person + biking+running of last person). This is obviously a schedule that takes a lot of time. Unlike the ascending order, as my proof idea says above, the person who spent the most amount of time can be finish before other contestants since this person can finish his/her running and biking while other contestant swimming. Therefore, the descending order allow all contestant who has a longer running and biking time to finish their mission whiling other contestant still in swimming. So the contestants who have the faster biking and running time can finish their mission fast which implies the completion time will be minimized.

Runtime Analysis:

newDict and answer initialization: O(1) times each

obtain newDict with givenDict: O(n) times there are n contestants

sort and reverse the newDict by value: O(nlog(n)) time

obtain answer list: O(n) times

Overall: O(n) + O(n) + O(1) + O(nlog(n)) = O(nlog(n))