HW3 Rubric

1. Exercise 10 Page 110 (15 points)

In an undirected Graph G=(V, E), given two nodes v and w, we want to find the number of all the shortest paths from v to w.

Algorithm (7 points)

- Run BFS from v with a slight modification. Store a counter for each node, initialized to 0, except for root node v, which is initialized to 1. We also need to store another variable in the nodes, denoting the layer they are located in, it is initialized to -1 except for the root (0), and gets updated when we discover nodes.
- Once we discovered a node y when exploring the neighbors of another node x, we do counter[y]+=counter[x], **only if y is in the next layer of x**.
  - If y was a node in the upper layer of x or its sibling, we shouldn't update counter[y]
- While running BFS, check if we have discovered the target node w. whenever a node at the layer of w was selected for exploration, we can terminate the algorithm.

Runtime analysis (3 points)

- Incrementing the counter and checking the layer can be done in O(1). Hence, all we have is the normal BFS algorithm, which is of O(m+n), plus some additional operations at each step, which could be done in O(1), and don't change the overall runtime of our algorithm.

Proof (5 points)

- Use induction
  - Our algorithm correctly counts the number of paths for all the nodes in layers 1 to k.
- Base case: k=1: # of all the paths from v to v is 1
- Induction step: we have found all the paths from root (v) to layer k-1.
- For each node t in layer k:
  - In all the shortest paths from v to t, the node before t must be located in (k-1)th layer. So, we have:

$$Number\ of\ paths\ ending\ in\ t = \sum_{\substack{z\,\epsilon\ layer\ (k-1) \\ and\ edge\ (z,t)\ exists}} number\ of\ paths\ ending\ in\ z$$

2. Exercise 6 on page 108 (15 points)

Proof: by contradiction: Assume there is an edge e=(x, y) in G that does not exist in T. These cases might arise:

- x and y are more than 1 layers apart in T.
  - Contradiction: T is a BFS tree. When BFS is exploring one of the nodes (whichever it selects first) the other will be added to the next layer.
- x and y are exactly one layer apart
  - T is a DFS tree. without loss of generality, assume DFS selects x for exploration before y. x should be the ancestor of y, and because they are one layer apart, x is parent of y and e exists in T.
- X and y are siblings
  - Same as the previous case. Without loss of generality, assume x is explored first. y should be in the subtree of x, because T is a DFS tree. Can't be siblings. Contradiction.

All cases ran into a contradiction. Thus, T=G.

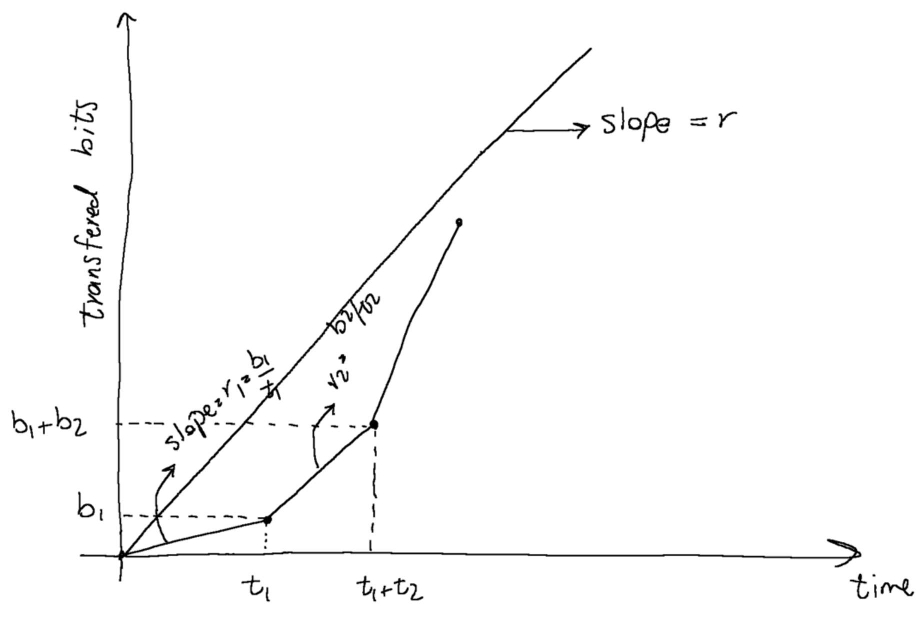
3. Exercise 12 on page 193 (20 points)

a) (5 points) Counterexample: suppose link parameter is r=5000, and we have (b1, t1)=(1000,1) and (b2, t2) = (6000, 1). Although $b_2 \leq rt_2$ does not hold here, running these two streams in the order of 1,2 is valid.

b) (15 points)

Claim: Given any n streams, the total number of transferred bits would be minimum at each time if we sort the video streams based on their bit rate $r_i = \frac{b_i}{t_i}$ and send them in non-decreasing order.

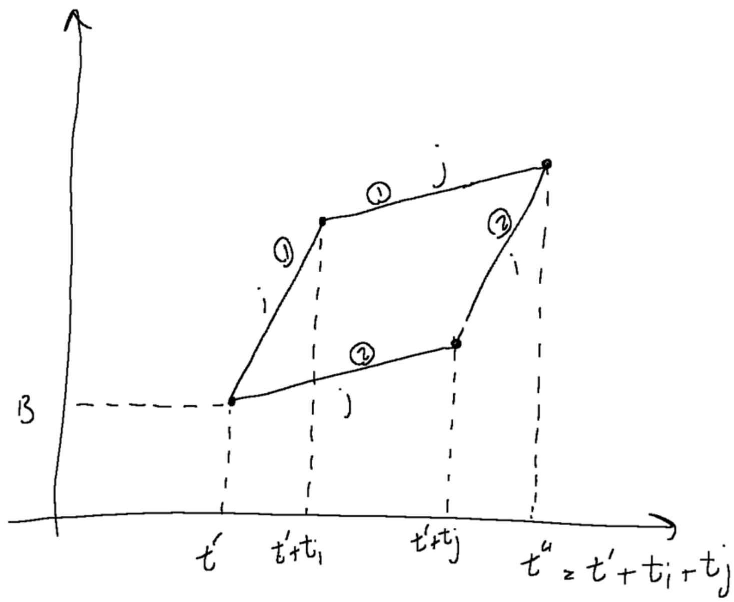
For better visualization, look at the following figure. The x and y axis are time and the total number of transferred bits respectively. the slope would be $m = {b_2 - b_1}/{t_2 - t_1}$ which has the same definition as bit transfer rate r. The question asks us to keep the diagram under the line with slope r.



Proof: by contradiction.

- assume the optimal ordering has an **inversion**. We define an inversion as two subsequent streams i, j for which $r_i > r_j$ (i is sent exactly before j). If the ordering is not non-decreasing, it is guaranteed that an inversion can be found.
  - Notation: the start time of sending i is $\boldsymbol{t'}$ and the time by which both i and j are sent is denoted by **t"** = $t' + t_i + t_j$.

- Reverse the order of i and j. We claim that the resulting ordering would have a lower transfer rate at all times, except for the beginning and the end of the period.
- Look at the following figure. The lines marked with 1 (on the top) are regarding to when we send i before j. (2) is for the exact opposite. B is the total number of bits that have been sent by t' (before we start transferring i and j).



- Note we calculating overall bit rate by $r(t) = {\# \, of \, bits \, sent \, by \, t}/{t}$
  - For times $t < t'$ and $t > t"$, the bit transfer rate would be the same for both orderings.
    - for $t < t'$: Nothing has changed
    - for $t > t"$: We send $b_i + b_{i+1}$ bits in $[t', t"]$ in both cases: No difference in calculating r(t)
  - For $t \in [t', t"]$ the modified ordering has lower bit transfer rate, i.e., the total number of bits for each t in the mentioned period would be lower in path 2 in comparison with path 1 (in the diagram)
  - It can be easily shown by using the fact that $r_i > r_j$.

We improved the optimal algorithm and this is a contradiction. An inversion cannot exist and the optimal ordering must be non-decreasing.

In the first figure, it is clear that if the lines are sorted in a non-decreasing order, the whole figure would be under the line connecting the origin to the end point. Furthermore, we know that the slope of this line is ${\sum_{k=1}^{n} b_i}/{\sum_{l=1}^{n} t_l}$. Thus, to solve the problem it suffices to calculate this fraction, which can be done in **O(n)**. Note that giving the exact order of sending video streams still needs O(nlog(n)) time, but it is not requested by the question.

Detailed rubric for part b: definition of algorithm: 5pts, proof: 5pts, Runtime: 5pts

4. Exercise 3 on page 189 (15 points)

We want to prove the greedy algorithm always stays ahead. Same as what we did in the interval scheduling problem, we use proof by contradiction: assume greedy algorithm is not optimal and another algorithm is doing better than it. Use induction for comparison:

- The greedy algorithm sends more or the same number of boxes in the first k trucks.
- Base case is obvious.
- Suppose the statement is correct for k-1.
    - Greedy algorithm have sent $b_1, \ldots, b_i$, and the other one $b_1, \ldots, b_j$, where $i \geq j$
- For kth round:
    - The other algorithm sends $b_{j+1}, \ldots, b_l$. We know that the sum of the weights of $b_{i+1}, \ldots, b_l$ is less than W because $i \geq j$. So we can first put the packages that the other algorithm has sent in the kth truck, and load the remaining space with other packages.
        - So, our algorithm is ahead after k rounds.

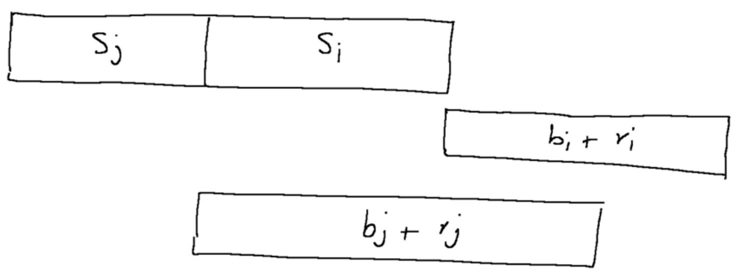
Overall, the greedy approach uses less number of trucks than other algorithms, or at worst case, equal.

5. Exercise 6 on page 191 (20 points)

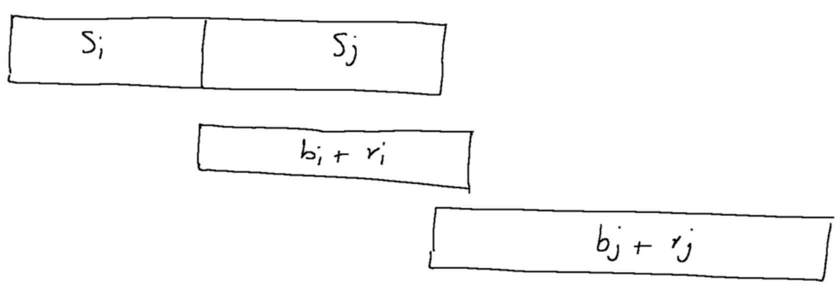
Algorithm (7 points)
Suppose s_i, b_i, r_i are projected swimming, biking, and running time for contestant i. We sort all contestants based on $b_i + r_i$ in non-increasing order and claim this is the optimal order.

Proof (10 points)
Imagine the optimal order is not what we suggested. The optimal algorithm does not follow the non-increasing order in terms of $b + r$, so we can find two subsequent contestants i, j (j starts immediately after i) for which $b_j + r_j > b_i + r_i$. We claim if we swap their position, the resulting order would be better.

- The pool is in use for $s_i + s_j$ in both orders.
- The finishing time of i and j in the modified order w.r.t. the time they started using the pool is:
    - $\max(s_j + b_j + r_j,\ s_j + s_i + b_i + r_i)$



- in the original order, it was:
    - $s_i + s_j + b_j + r_j$



Because $b_i + r_i > b_j + r_j$, the finishing time of the modified algorithm is better, and this is a contradiction to the original assumption: it must be impossible to improve the optimal algorithm.
Hence, an algorithm with an inversion cannot be optimal, and our claim was correct.

Time complexity (3 points)
Calculating r+b for all contestants is O(n). Sorting could be done in O(nlog(n)). Overall, O(nlog(n)).

6. The rotting orange problem (15 points)

{2, 1, 0, 2, 1}
{1, 0, 1, 2, 1}
{1, 0, 0, 2, 1}

Algorithm (7 points)

The solution would be similar to running several BFS algorithms simultaneously.

In the beginning, put the indexes of all the 2s in a queue and call it q[0].

- q[0]=[(0, 0), (0, 3), (1, 3), (2, 3)].
- q[i] is the set of all oranges that rot in timestamp i

Moreover, count the total number of ones in the matrix and call it cnt1.

Follow this algorithm:

- Pop the first element of q[i]
- Add the indexes of all the 1s adjacent to it to q[i+1]
- Change those 1s to 2
- Subtract the number of added elements from cnt1
- Repeat the following steps until q[i] is empty, then If q[i+1] is empty
    - If cnt1=0, it means that all the oranges are rotten at the end. Return i
    - If cnt1>0: it is impossible for every orange to be rotten. Return -1

Time complexity analysis (5 points)

If we take each index as a node, the number of nodes is NM. We select each node for exploration at most once and explore 4 directions. The overall runtime would be O(MN).

Proof (3 points)

We should prove if there exists a shortest path p with length m from a rotten orange to a fresh orange:

- The fresh orange will definitely rot
- It will rot at time m

Both are obvious based on how BFS works. Moreover, it is obvious that if there is no path to a fresh orange, the algorithm will return -1.