CSE548 Fall 2019 Analysis of Algorithms Homework3

Tianao Wang 112819772

October 15, 2019

1. Sort everyone's biking time + running time in a decresing order. Then send the contestants in this order.
   Suppose there is a solution which is not in this order. There must be two contestants i and j which $b_i + r_i < b_j + r_j$ and j begins earlier than i. If we change the order of i and j. In this case, j will finish earlier than before and i will start after j finish swimming. Due to $b_i + r_i < b_j + r_j$, i will finish earlier than j before. So, we can change the schedule and let the larger $bikingtime + runningtime$ to start earlier. Finally, we can rebuild the order into a new one which is sorted by $bikingtime + runningtime$ in a decresing order.

2. (a) False.
   Consider there are two streams. $(b1, t1) = (2000, 1), (b2, t2) = (6000, 1)$. We can send stream1 first and then send stream2 which is a valid schedule.

   (b) Sort the streams by b_i/t_i in an incresing order. Check whether it is a valid schedule in this order. If $\sum_{i=1}^{n} b_i < r \sum_{i=1}^{n} t_i$, we can say it has a valid schedule. If not, there is no valid schedule.
   In this case, we send the lowest increasing rate stream first. Make sure that there is biggest valid data space for the rest streams. If, in this case, the schedule is still invalid, we can say there is no valid schedule.
   The running time of sorting is $O(nlg\ n)$, checking is $O(n)$. So the total running time is $O(nlg\ n)$.

3. (a) Sort the processes by their finish time. Invoke status check program at the finish time of the first uncovered process until all the processes are covered.
   Proof by induction.
   Base case: If these is one processes, we need invoke status check program for one time.
   Inductive hypothesis: If there are k processes, we need invoke status check program for q times.
   Inductive step: If there are k+1 processes, there are two situations. One is we don't need to invoke the program because the finish time of new process is earlier than the former total finish time. The other is that we need to invoke the programe for one more time because the finish time of the new program is later than the former total finish time.
   In conclusion, this algorithm can find the least time to invoke status check program.

   (b) True.
   Consider there are k processes with k* individual processes and we start k* times status

check program.
Suppose there is one process in k processes which need to invoke status check program for one more time. It can't be intersected with other process. So there are k*+1 individual processes and the suppose is wrong. In conclusion, individual process is the only thing forcing us to need invocations of status check.

4. Algorithm: Let x_i, i from 1 to n, matches the earliest end t_j. If all the x_i can be matched as $|t_j - x_i| < e_j$, there exists an association. If not, there is no association.
   Proof: Suppose there is an association and this algorithm can't match. There is a x_a which matches t_m in the association. But the algorithm calculate that x_a should match t_n, which $n < m$. Then we can change the matching pair into x_a with t_n. This will not change the accuracy of the association. After changing every situation like this, we can obtain a matching list same with the algorithm.

5. Search the (n/2)th element of A and B. Then compare A(n/2) with B(n/2). If $A(n/2) < B(n/2)$, second part of B is larger than first part of A and first part of B(totally $> n$), so the second part of B can be ignored. In addition, first part of A is less than second part of A and second part of B(totally $> n$), so the first part A can be ignored. In this way, we can ignore n/2 elements each time. Then recurse this process, and decrase the value of n until n = 1 and then calculate the average of the final two value which is the median of the 2n numbers.
   Due to I can't delete the elements, so I just abandon the smaller part and save the larger part. So each time, I can ignore n/4 elements in current data, the total running time is $O(2lg\ n)$

6. 1) Divide the chessboard into 4 parts by n/2. Compare the points on the divide lines.
   2) If minimun points is on the border, we recurse the step1.
   3) if minimun points is on the edges connecting to more than 2 parts, we compare the connected points with this point. If the middle point is the minimum, return this point. If not, select the minimum point and compare the connected points with this point. If it still the minimun point, then return. If not, select the area of this point and recurse step1.
   Follow these three step, we can find the local minimum.