CSE 6140

Assignment 1

Bhanu Verma - 903151012

**Question 1** - Simple Complexity



a. 443

b. 1002

c. 10002

d. n

e. n

f.

* 1. Time Complexity -
* Data Size - n
* Given code has two while loops, outer one iterates for n times. So, we get from there.
* Inner loop counts the number of powers of 3 in n, and hence time complexity for this step becomes 3 which is nothing but .
* Hence, total time complexity for the given algorithm becomes .
* Given algorithm outputs every number starting from 1 till n and the number of powers of 3 contained in that number in the form of a tuple. Here is the output for n = 9 - (1,0), (2,0), (3,1), (4,0), (5,0), (6,1) …. (9,2).

**Question 2** - Greedy - Tech Swim Run Bike

**Limitation** - Only one person can swim at a time, no such limitation on running or biking.

**Task** - to minimize the completion time

It might be very tempting to think that as the resource for swimming is available with a restriction so our solution might be devised in that direction. But if we think logically, we can see that no matter what order we use for athletes, total time required by all athletes for swimming will always be same. What will matter in the end is how much running and biking time does the last swimmer take in our schedule. If that time is more than the minimum running and biking time for any athlete, then we can see that there will be some idle time.

Hence, the solution that we propose is that combine and add the running and biking times to form a single variable and sort it in the decreasing order. Then, the order that we get will be the optimal solution.

**Our Claim** - Sorting the running and biking times in decreasing order will give us the optimal solution for minimizing completion time.

**Proof** - As we have ensured that last swimmer will have minimum running and biking time, we can guarantee that our solution will have no idle time.

1. ***Claim - There exists an optimal schedule with no idle time.***

Now, we need to prove that our schedule A is optimal i.e. it minimizes maximum lateness. Let’s define inversion as well. Inversion is when an athlete with less *running+biking* time is scheduled before an athlete with more *running+biking* time.

1. ***Claim - All schedules with no inversions and no idle time have the same finish time or maximum lateness.***

Two different schedules that no inversions and no idle time can only produce a different order of schedule if the running+biking times of more than one athletes are same. But even for such case completion time will not depend on the order of athletes with same times. Now, to prove that our algorithm is optimal, we need to prove that an optimal schedule has no inversions and no idle time.

1. ***Claim - There is an optimal schedule that has no inversions and no idle time****.*

We will prove this by taking an optimal schedule with no idle time and then change it into a schedule with no inversions without increasing the completion time. If an optimal schedule has an inversion then we have a case where an athlete ai , with less running+biking time is scheduled before another athlete ai+1 with more running+biking time. It is very easy to see that if there is only one such inversion in a schedule of decreasing order of *running+biking* time, then those two athletes will be placed consecutive to each other. We can just swap their positions in the schedule to remove this inversion and there will be no more inversions.

Now, we need to prove that this swapped schedule has maximum completion time no larger than the optimal schedule. Let’s say we swap ai with ai+1, one thing is certain that no two other athletes have changed their finish times. No, we see that ai+1 is completing earlier in new schedule. Crucial point here is that ai ***cannot be more late*** in the new schedule as ai+1 was in the previous schedule as completion time for ai < ai+1.

Optimality of our greedy algorithm can easily be deduced now. *We proved that an optimal schedule with no inversions exists and we also proved that all schedules with no inversion have the same maximum completion time. Hence, schedule generated by our algorithm is the optimal schedule.*

**Question 3** - Divide & Conquer

Let’s have a look at the data given in the question:

* Number of Tags - n
* Each tag can be represented as i where
* Rule: if then they light up on touching

Here is the algorithm that uses Divide & Conquer approach to solve this problem in :

* Keep on dividing the problem size into two equal halves, until problem size becomes 1.
* Use recursion for above step
* Base Case - when problem size becomes 1, return that tag
* Now for each level above base level, tap two tags i.e. check if their ids are same.
* If they light up i.e. if their ids are same return any tag to the upper level.
* If they don’t light up, check if any of them is in the majority by comparing each of these tags with the problem size at current level for that node.
* If in majority, return that tag. Else return Null or None.
* Intuition of finding majority is equivalent to finding Boyer Moore’s majority vote algorithm. [1]
* This step ensures that there will maximum of comparisons at each level and we have divided our problem into two equal halves at each level, depth of our tree or total number of levels are .
* Hence, total time complexity for our algorithm become .

Time complexity can also be deduced from Master’s theorem i.e.

where is a monotonically increasing

= number of sub-problems,

= size of each problem,

= time of combine step

For our solution, as we are dividing our problem into two sub-problems at each step and as well. for algorithm is as we are making comparisons for combining solutions.

Hence, using master’s theorem our time complexity becomes **.**

Here is the working code for this problem:



**References**:

1. <https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_majority_vote_algorithm>
2. <https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf>
3. Code Reference for UnionFind: <https://www.ics.uci.edu/~eppstein/PADS/UnionFind.py>