Q1

Denote that the length of a snake's DNA is n, and a snake with venom level x must have DNA length of $5x$. So with a n length of DNA, the maximum venom level is $\lfloor n/5 \rfloor$. Which means the time complexity is $O(n/5) * O(n) = O(n^2)$ when using brute force. That is check venom level from 1 to $\lfloor n/5 \rfloor$, and each round we use greedy search runs in $O(n)$.

We can use binary search to check the venom level so that we can run in $O(nlogn)$.

First, we count the numbers n_s, n_n, n_a, n_k, n_e of occurrences of each letters, 'S', 'N', 'A', 'K, 'E' in the given DNA sequence and denote left boundary $l = 0$, right boundary $r = min\{n_s, n_n, n_a, n_k, n_e\}$, which means we have a list contains range l to r stands for possible venom level. Apart from that, variable max_level to store the max venom level with initial value 0.

Then we do binary search, use while loop ,let $mid = (l + r)//2$ and use greedy algorithm to check wether the given DNA can derive level mid DNA sequence by deleting zero or more letters. If so, let $l = mid + 1$, $max\_level = mid$, otherwise, let $r = mid - 1$, and continue the search in order to find the max level. Until the loop ends($l = r$), and we can find the result.

For greedy method, since the format of venom DNA sequence is fixed, so for the given snake DNA we check from the start, is there have mid's 'S' followed by mid's 'N',mid's 'A',mid's 'K' and mid's 'E', If it is, so this DNA can have level mid, otherwise not and do next loop.

This algorithm runs in $O(logn)$ for binary search and $O(n)$ for greedy method each loop, so total time complexity is $O(nlogn)$.

Optimality: as I mentioned above, when use greedy method to traverse the given DNA sequence, first check wether exists mid's 'S', if not we can break this loop and do next, that is the result must have 'S'*mid – 'N'*mid – 'A'*mid – 'K'*mid – 'E'*mid order. we cannot find any optimal solution with the random order of those letters, which means given any optimal solution we cannot find difference, so greedy solution is the optimal solution.

Q2

According to the question, we already know that we know the exactly R_a, P_a and S_a number and $R_a + P_a + S_a = N$, same as R_b, P_b and S_b also $R_b + P_b + S_b = N$, however, we don't need to throw in the consecutive order.

The question is how should we play to maximise the number of points, which can be convert to that to minimise the loss round, as the total round is N, so win round plus loss round plus tie round is equal to N. Minimise loss round is equal to maximise the win round plus tie round.

Assume we have two arrays, array A with [R_a, P_a, S_a] and array B with [R_b, P_b, S_b], so the maximum win round is $win\_round = min(R_a, P_b) + min(P_a, S_b) + min(S_a, R_b)$, Denote $w_0 = min(R_a, P_b)$, $w_1 = min(P_a, S_b)$ and $w_2 = min(S_a, R_b)$, we throw Paper, Scissor and Rock in w_0, w_1 and w_2 rounds respectively, after that we can derive new array A_1 with $[R_a - w_0, P_a - w_1, S_a - w_2] = [R_a^{'}, P_a^{'}, S_a^{'}]$ and new array B_1 with $[R_b - w_2, P_b - w_0, S_b - w_1] = [R_b^{'}, P_b^{'}, S_b^{'}]$, then we can compute the maximum tie rounds is $tie\_round = min(R_a^{'}, R_b^{'}) + min(P_a^{'}, P_b^{'}) + min(S_a^{'}, S_b^{'})$ and same as above, for number of array A_1 and B_1, minus the minimum value at the same index, we finally get new array A_2 [$R_a^{''}$, $P_a^{''}$, $S_a^{''}$] and new array B_2 [$R_b^{''}$, $P_b^{''}$, $S_b^{''}$], it is clearly that some of those values are 0. Finally the rest of number of array A_2 and B_2, no matter how to throw are all loss so the loss round which is $loss\_round = sum(A_2) = sum(B_2)$ so the total score that by doing this method is $win\_round - loss\_round$ which is also the maximum score.

Optimality: assume we have another optimal solution with the same scores but not the same maximum rounds of win_round plus tie_round which is less than our algorithm. If so, this solution must have more loss rounds as the $win\_round + tie\_round + loss\_round = N$, in this situation, the scores must less than our's method, which is a contradictory. So my algorithm shows above can give the optimal solution.

Q3

According to this question, a train can arrive before the midnight and leave after midnight, so we first find how many trains followed this rule, those trains must be separate on different platforms.

Assume there are t trains that across midnight so the number of platforms must be greater or equal than t, which means $total \geq t$, now we sort these across midnight trains by departure time in increasing order. Denote list T.

For each trains in list T, we do a loop, we pick up in order and we can convert this problem to *Activity selection problem* talked on the lecture, we split the day as start at d_i and end at a_i. So the subquestion now is to find a maximum size subset of compatible trains. Note that trains here are non-across midnight trains. Like the answer of *Activity selection problem*, among the trains which do not conflict with the previously chosen trains always chose the one with the earliest end time. At the end of this loop, we delete those trains that picked in this loop and the rest trains as the next loop's non-across midnight trains.

After find all maximum size subset of trains for trains in list T, if there still have trains we continue do the above steps but the day has the normal start time and end time. Until there are no trains left. In this stage, when finishing one loop, means one more platform added. So finally we can find the number of platforms.

In this question, we followed the rules that find the maximum size subset of compatible trains and stay one platform, so when the total number of trains fixed, we can get the minimum number of platforms so that each train can stay without interfering with other trains.

Optimality: exactly like *Activity selection problem*, assume there is any different optimal solution, we find the first place where the chosen trains violates our algorithm choice. Show that replacing that trains with our method choice produces a non conflicting selection with the same number of trains. Continue in this manner till "morph" this optimal solution into our method solution, thus proving our method solution is also optimal.

Q4

According to this question, we should find the subset of jobs that maximises profit. So we first sort all jobs in decreasing order of profit and denoted as array p. This step runs in $O(nlogn)$ by using MERGE SORT.

Assume we have an empty list R that can be split into n's slots that can store jobs sequence.

Then, we use a single for loop and iterate on jobs in decreasing order of profit. And for each job i, we try to find an empty slot I in R that satisfy that $d_i > I$ and I is the greatest slot in range 1 to d_i. And put this job i in I's slot in list R and mark this slot is not empty. If there is no empty space for job i, which means no empty slot in range 1 to d_i, we will ignore this job and not consider this job.

Imagine the worst case, for n jobs, the first job we found slot in one times, second job we found in twice

..., nth job found slot in n times, so the time complexity is $O((n+1)n/2)$ which is $O(n^2)$. So the total time complexity of this algorithm is $O(n^2)$.

Optimality:

Assume there are two jobs i and j, $p_i < p_j$, but we consider job i first:

If list R have enough space for jobs, there is no difference with our greedy solution as we can simply swap these two jobs and obtain our greedy solution. Otherwise if there is only one room for a job, clearly we should consider job i first so that we can make more profit.

Assume there are two jobs i and j, $p_i = p_j$, and $d_i < d_j$, but we consider job j first:

If list R have enough space for jobs, there is no difference with our greedy solution as we can simply swap these two jobs and obtain our greedy solution. Otherwise if only job j in list, and in a place which is in range 1 to d_i and there will be no space for job i, so it's trivial that if we put job j more closer to its deadline, it may give place for i, so that our greedy solution will be obtained and increase the total profit.

Thus, our greedy method solution is optimal.

Q5

According to question, we know that each day chemicals will evaporate p percent of the amount you had at the end of previous day. So first we can determine how much of a chemical is left after k many days. We have N chemicals to produce so we need N days.

If we consider case that producing chemical in order. And according to my own understanding, I **assume the delivery is complete at the end of No.N day, and the last chemical produced doesn't evaporate but the previous produced chemicals will also evaporate**, so the total losing weight is

$$total = (W_1 - W_1(1-p\%)^n) + (W_2 - W_2(1-p\%)^{n-1}) + \cdots + (W_n - W_n(1-p\%)^0)$$

Consider $y = (1-p\%)^n$, when n decrease, the y will increase, and $1-(1-p\%)^n$ will decrease, so $lose = W(1-(1-p\%)^n) = W - W(1-p\%)^n$ will be decrease when n decrease which means one chemical will lose less if it be produced later. But according to question, chemicals have their own weight, what we should do is to arrange that those high-demand chemicals should be produced later and low-demand should be produced first.

Thus, sort chemicals in increasing-order of weight, and produce Ci in that order can minimise the total extra weight of all chemicals needed to produce.

Optimality: Assume we already sort the chemicals in increasing order of weight and denote the total loss is:

$total = (W_1 - W_1(1-p\%)^n) + (W_2 - W_2(1-p\%)^{n-1}) + \cdots + (W_k - W_k(1-p\%)^{n-k+1}) + (W_{k+1} - W_{k+1}(1-p\%)^{n-k}) + \cdots + (W_n - W_n(1-p\%)^0)$

(W_1, W_2, ... , W_n is already ordered). And let us see what happens if we swap to adjacent chemicals C_k and C_{k+1}, the total loss become:

$total' = (W_1 - W_1(1-p\%)^n) + (W_2 - W_2(1-p\%)^{n-1}) + \cdots + (W_{k+1} - W_{k+1}(1-p\%)^{n-k+1}) + (W_k - W_k(1-p\%)^{n-k}) + \cdots + (W_n - W_n(1-p\%)^0)$

Thus,

$$total - total^{'} = W_k(1-p\%)^{n-k} - W_k(1-p\%)^{n-k+1} + W_{k+1}(1-p\%)^{n-k+1} - W_{k+1}(1-p\%)^{n-k}$$

Denote $x = (1-p\%)$

$$\begin{aligned} total - total^{'} &= W_k x^{n-k}(1-x) + W_{k+1}x^{n-k}(x-1) \\ &= (1-x)(W_k - W_{k+1})x^{n-k} \end{aligned}$$

As $W_k < W_{k+1}$ and $1-p\% > 0$, so $total - total^{'} < 0$, so as long as there are two chemicals in decreasing order, means there are two chemicals produced in decreasing order and will produce more extra weight when producing. Thus the optimal solutions is the one that in increasing order of weight to produce.