CS3230 WA2

Yadunand Prem, A0253252M

Question 1

Since there is a favourable seating constraint, and f_i is distinct set of numbers, we can sort the guests in a decreasing order of favour score. This can be done in $O(n \log n)$ timing

We can think about this problem as picking a guest from the sorted list and adding them to either the left of the host or the right of the host. Brute forcing this would run in $O(2^n)$ time complexity. We can reframe this as such, `DP(num_left, num_right, last_left, last_right)`. This keeps track of the number of people on the left, number of people on the right, and the index of the person last placed on the left and right. last_left and last_right is used to calculate the hatred when a new person is added to the corresponding side.

To show that there are overlapping subproblems, we need to show that 2 different paths then result in solving a similar problem. We can generate them as such: With `num_left == num_right`, and the last `x` moves are the same, and they contain at least one `L` and one `R`, they are overlapping. For example, if the pathing taken was `LRLRLR` and `LLRRLR`, both would end up with `DP(3, 3, 6, 7)`. Thus, there are overlapping subproblems.

At the kth iteration, the kth most favourable guest must be added to either the left end or the right end. Suppose not, that there is a more optimal solution where another guest is added instead. This would form a contradiction of the favourability constraint. Therefore, the kth most favourable guest must be added to either the left end or the right end.

Base Case: `dp(n, n, last_left, last_right) = 0`

This occurs when this has reached the last node of the recursion tree, when there are n guests on both the left and right.

General Case: Let $\text{last\_left}, \text{last\_right}, \text{num\_left}, \text{num\_right}$ be l_l, l_r, n_l, n_r, and curr = `max(l_l, l_r)+1`

$$\text{dp} = \min\begin{cases}\text{hatred}[l_l][\text{curr}] + \text{dp}(n_l+1, n_r, \text{curr}, l_r) \\ \text{hatred}[l_r][\text{curr}] + \text{dp}(n_l, n_r+1, l_l, \text{curr})\end{cases}$$

The solution is the minimum of *going* both left and right, and then calculating the min hatred from that. We can form the memoization by indexing the table by (num_left, num_right, last_left, last_right). Since each value ranges from $0 - n$, the total number of subproblems is $O(n^4)$. Each subproblem only has addition and comparsions, all of which run in $O(1)$ timing. Therefore, this algorithm runs in $O(n^4)$.

We can optimise this by deriving `num_right = max(last_left, last_right) - num_left`. This reduces the number of subproblems to $O(n^3)$, and thus, the total time reduces to $O(n^3)$.

```
dp(num_left, num_right, last_left, last_right):
  next = max(last_left, last_right) + 1

  if num_left == n and num_right == n:
    return 0

  if num_left != n:
    left = h[last_left][next] + dp(num_left + 1, num_right, next, last_right)
  if num_right != n:
```

```
  right = h[last_right][next] + dp(num_left, num_right + 1, last_left, next)
return min(left, right)
```

Question 2

2bi)

$P(\text{any random element from T has T-rank in} [k, k+r)) = \frac{r}{n}$
$P(\text{any random element from T does not have T-rank in} [k, k+r)) = 1 - \frac{r}{n}$ #

2bii)

Show that $P(\text{none of elements in S has T rank in } [k, k+r))$ is bounded from above by $\frac{1}{n^3}$

$P(\text{none of elements in S has T rank in } [k, k+r)) = \left(1 - \frac{r}{n}\right)^{2n^{\frac{2}{3}}} = \left(1 - \frac{5n^{\frac{1}{3}}\ln(n)}{n}\right)^{2n^{\frac{2}{3}}} = \left(1 - \frac{5\ln(n)}{n^{\frac{2}{3}}}\right)^{2n^{\frac{2}{3}}}$

let $x = \frac{n^{\frac{2}{3}}}{5\ln(n)}$, it can be seen that $\forall n > 1, x > 0$.

$\left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e}$(Using Lemma 1b)

$\left(1 - \frac{1}{x}\right)^{\frac{n^{\frac{2}{3}}}{5\ln(n)}} \leq \frac{1}{e} \equiv \left(1 - \frac{1}{x}\right)^{n^{\frac{2}{3}}} \leq \left(\frac{1}{e}\right)^{5\ln(n)}$

$\left(1 - \frac{1}{x}\right)^{2n^{\frac{2}{3}}} \leq \left(\frac{1}{e}\right)^{10\ln(n)} \equiv \left(1 - \frac{1}{x}\right)^{2n^{\frac{2}{3}}} \leq \left(\frac{1}{n^{10}}\right)$

$\left(1 - \frac{5\ln(n)}{n^{\frac{2}{3}}}\right)^{2n^{\frac{2}{3}}} \leq \left(\frac{1}{n^{10}}\right) \leq \left(\frac{1}{n^3}\right)$

2c)

Let the interval r_i, r_{i+1} be called a range.

Show that the $P(\text{S contains at least 1 element from each range}) \geq 1 - \frac{1}{n^2}$. This is the same as $P(\text{S contains no elements from each range}) < \frac{1}{n^2}$

$P(\text{none from each range}) < \sum P(\text{none from range i})$ (Union Bound)

$P(\text{none from range i}) \leq \frac{1}{n^3}, \text{num of ranges} = \frac{n}{5n^{\frac{1}{3}}\ln(n)}$

$\sum P(\text{none from range i}) \leq \frac{\frac{n}{5n^{\frac{1}{3}}\ln(n)}}{\frac{1}{n^3}} = \frac{1}{n^2 \cdot 5n^{\frac{1}{3}}\ln(n)}$

$\forall n > 1, \ln(n) > 0$

$\therefore, \frac{1}{n^2 \cdot 5n^{\frac{1}{3}}\ln(n)} < \frac{1}{n^2}$

$P(\text{S contains no elements from each range}) < \frac{1}{n^2}$

$P(\text{S contains at least 1 element from each range}) \geq 1 - \frac{1}{n^2}$ #

2d)

Let the event where $a \leq M$ be A and $b \geq M$ be B.

$P(A) = P(B) \geq 1 - \frac{2}{n^3}$
$P(A \cap B) = P\left(\overline{\overline{A} \cup \overline{B}}\right) = 1 - P\left(\overline{A} \cup \overline{B}\right)$
$P\left(\overline{A} \cup \overline{B}\right) \leq P\left(\overline{A}\right) + P\left(\overline{B}\right) = \frac{4}{n^3}$
$1 - P\left(\overline{A} \cup \overline{B}\right) \geq 1 - \frac{4}{n^3}$

$P(A \cap B) \geq 1 - \frac{4}{n^3} \geq 1 - \frac{1}{n}$ #

Between a and b, there are $2 * 5n^{\frac{1}{3}} \ln(n)$ elements in S. With a $1 - \frac{1}{n^2}$ probability, we can say that each element must at least be from 1 range in S'. Therefore, there are at most $10n^{\frac{1}{3}} \ln(n) \cdot 5n^{\frac{1}{3}} \ln(n) = 50n^{\frac{2}{3}}(\ln(n)^2)$ elements between a and b in S'.

$P(\text{S contains at least 1 element of T rank from each range}) =$

$P\Big(\text{S' contains at most } 50n^{\frac{2}{3}}(\ln(n))^2\Big) = 1 - \frac{1}{n^2} < 1 - \frac{1}{n}$ #

2e)

To prove that the algorithm runs in linear time, we need to show that each step of the algorithm runs in at most $O(n)$.

1. Randomly selecting $2n^{\frac{2}{3}}$ elements take $O(n)$ timing
2. Sorting S of $2n^{\frac{2}{3}}$ elements takes $2n^{\frac{2}{3}} \log\Big(2n^{\frac{2}{3}}\Big) = O\Big(n^{\frac{2}{3}} \log(n)\Big) < O(n)$ using a comparison based sorting algorithm
3. This can be calculated in $O(n)$ by counting the number of elements lesser than a and b in T

4-6. $O(1)$

7. This can also be done in $O(n)$, by iterating through T and selecting the valid values that are $a \leq n \leq b$
8. S can have at most $1000n^{\frac{2}{3}}(\ln n)^2$ elements, and sorting them takes $1000n^{\frac{2}{3}}(\ln n)^2 \log\Big(1000n^{\frac{2}{3}}(\ln n)^2\Big)$. $1000n^{\frac{2}{3}}(\ln n)^2 < O(n)$. $\log\Big(1000n^{\frac{2}{3}}(\ln n)^2\Big) < O(n)$. $\therefore$. This runs in at most $O(n)$

Step 9: Indexing a sorted array can be done in $O(1)$

Therefore, since each operation runs in at most $O(n)$, this algorithm runs in linear time

From part d, $P(a \leq M \leq b) \geq 1 - \frac{1}{n}$. So With probability of $1 - \frac{1}{n}$, M lies in S'. Since S' is sorted and a portion of T, the median would be in the center of T. It then gets the index of the element at middle of T, the Median, and by subtracting R_a, it gets the index of the median with respect to S'. Therefore, the algorithm would return the median with $1 - \frac{1}{n}$ probability.