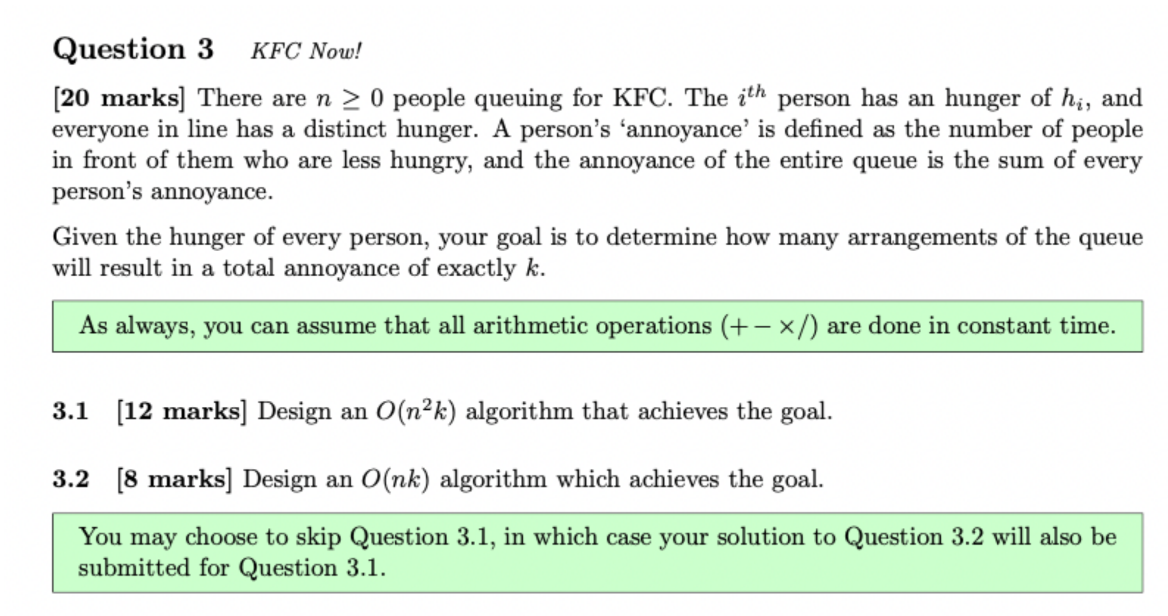
Comp3121 assignment 4 Question 3
Chen En (z5335039



Question 3 *KFC Now!*

[20 marks] There are $n \geq 0$ people queuing for KFC. The i^{th} person has an hunger of h_i, and everyone in line has a distinct hunger. A person's 'annoyance' is defined as the number of people in front of them who are less hungry, and the annoyance of the entire queue is the sum of every person's annoyance.

Given the hunger of every person, your goal is to determine how many arrangements of the queue will result in a total annoyance of exactly k.

As always, you can assume that all arithmetic operations $(+-\times/)$ are done in constant time.

3.1 **[12 marks]** Design an $O(n^2k)$ algorithm that achieves the goal.

3.2 **[8 marks]** Design an $O(nk)$ algorithm which achieves the goal.

You may choose to skip Question 3.1, in which case your solution to Question 3.2 will also be submitted for Question 3.1.

3.2

Base case: $dp[i][0]$ (All cases when annoyance = 0.)

Recursion:

Case 1: $j < i \Rightarrow dp[i][j] = dp[i][j-1] + dp[i-1][j]$

Case2: $j \geq i \Rightarrow dp[i][j] = dp[i][j-1] + dp[i-1][j] - dp[i][j-i]$

Subproblem: For each $1 \leq i \leq n$ and each $0 \leq j \leq k$, let $Q(i,j)$ let the number of total arrangements of annoyance j of i people be $dp[i][j]$.

Answer: $dp[n][k]$

Justification:

Since people's hunger is distinct, we define the people's hunger as different number 1…n. That is, when there are i people we present their hunger as $\{1 \ldots i\}$. And the sequence shows the different arrangements of the people waiting in line. And the number behind each list shows the total annoyance. For instance, when there are two people waiting in line there are only two cases of arrangement, which is $\{1,2\}$ with the total annoyance of 1. And $\{2,1\}$ with total annoyance of 0. We present those two cases as $\{1,2\}$: 1 *and* $\{2,1\}$: 0.

number of people	1 person	2 people	3 people	i people
arrangement of people waiting in line	1	{1,2}: 1 {2,1}: 0	{3 ,1, 2}: 1 {1, 3 ,2}: 2 {1, 2, 3}: 3 {2, 1, 3}: 0 {2, 3, 1}: 1 {2, 1, 3}: 2	…..

To show all the arrangements of 3 people cases, we can derive it from the previous 2 people cases and insert 3 into different indexes in the list. (We present the 2 people cases in [ ] .)

$$\{3\,[2,1]\} : 0$$
$$\{[2]3\,[1]\} : 1$$
$$\{[2,1]3\} : 2$$
$$\{3\,[1,2]\} : 1$$
$$\{[1]\,3\,[2]\} : 2$$
$$\{[1,2]3\} : 3$$

From the above listing, we notice that the total annoyance of each case can be calculated by the index where we insert 3 plus the total annoyance from 2 people cases. For instance, as the case $\{3,[2,1]\}$ the total annoyance is the index of 3 which is 0, plus the total annoyance of $\{2,1\}$ which is 0. As a result, the total annoyance of $\{3,2,1\}$ is 0. And we can conclude this into the recursion case of

$$\boldsymbol{total\ annoyance\ of\ arrangement\ (i) =}$$
$$\boldsymbol{idx_{insert}(i) + total\ annoyance\ of\ arrangement\ (i-1)},$$

that $0 \leq index\_insert \leq i-1$. That is, the number of arrangements of i people with total j annoyance can be calculated by summing up the number of arrangements of i-1 people with total $j, j-1, \ldots, j-i$ annoyance.

Let the number of total arrangements of annoyance j of i people be $dp[i][j]$

Case 1: $j < i$

$$dp[i][j] = dp[i-1][0] + \dots + dp[i-1][j]$$
$$\Rightarrow dp[i][j] = dp[i][j-1] + dp[i-1][j]$$

Case 2: $j \geq i$

$$dp[i][j] = dp[i-1][j-i] + \dots + dp[i-1][j]$$
$$\Rightarrow dp[i][j] = dp[i][j-1] + dp[i-1][j] - dp[i][j-i]$$

$$Since\ dp[i][j+1] = dp[i-1][j-i+1] + \dots + dp[i-1][j+1]$$
$$= (dp[i-1][j-i] + \dots + dp[i-1][j]) + dp[i-1][j+1] - dp[i-1][j-i]$$
$$= dp[i][j] + dp[i-1][j+1] - dp[i-1][j-i]$$

We use the above formula to calculate the amount of total arrangement of annoyance, which we then take those numbers to fill in the array $dp[i][j]$.

Time complexity

Since we create an n*k array $dp[i][j]$ to store the arrangements of annoyance when there are different numbers of people, which the time complexity takes $O(n * k)$. As we search up the annoyance k from the table, it takes $O(1)$. Thus, the total time complexity is $O(nk)$.