Sicheng Chu schu37@wisc.edu Lec 001
Hiroshi Shu hshu4@wisc.edu Lec 001
Hw 05

1.
Define boarding time interval as 15 min before boarding time to 15 mins after boarding time. For time intervals $t_i(S_i, E_i)$ belongs to All boarding time interval T, sort them in a non decreasing order. Assign a staff for $t_1(S_1, E_1)$ and let that staff work for 2 hours. When the staff is finished for the shift, check if at the time there is a boarding interval contains the finishing time for the shift. If so, immediately assign another staff for a 2 hours shift, else, wait until the next starting intervals to assign the staff.

Assume there is an ideal algorithm O that solves the problem. We can compare it with our greedy algorithm G. By default we need staff to cover all time interval for the week. Thus for $t_1(S_1, E_1)$, we must assign a staff to cover it. The Greedy Algorithm shift's end time for the first interval is no less than O because we use the maximize time. If there is an interval for end time in algorithm O, we still need a second staff to cover the shift and the start time of the second staff is no sooner than G's algorithm. If there is no shift, then G's algorithm assign the second staff no sooner than O's algorithm because it waits until the next interval that requires us we must assign an staff. So for every assignment, G';s algorithm's start time is no sooner than optimized algorithm O. Thus G is the optimized solution.

Sorting the time interval take nlogn. Checking all intervals for ending shift time cost n as long as we traverse the sorted time interval and keep tracking the current time intervals the shift is covering. So the whole algorithm cost O(n) = nlogn.

Sicheng Chu schu37@wisc.edu Lec 001
Hiroshi Shu hshu4@wisc.edu Lec 001
Hw 05

Sicheng Chu schu37@wisc.edu Lec 001
Hiroshi Shu hshu4@wisc.edu Lec 001
Hw 05

2.

(1)Let A be the array that stores the values of all cards with responding ordering. Then $A[1] \cdots A[n]$ are the value of card $1 \cdots n$. Define $Val(i, j)$ as the maximum total value player 1 get when both players perform optimally where i represent the leftmost index of array A and j represents the rightmost index of array A, then

$$Val(i, j) = max((A[i] + min(Val(i + 1, j), Val(i + 1, j - 1)), (A[j] + min(Val(i + 1, j - 1), Val(i, j - 2)))$$

.

The base cases are either $i = j$ or $i = j - 1$ since the number of cards is either odd or even. So for $i = j$, $Val(i, j) = A[i]$; For $i = j - 1$, $Val(i, j) = max(A[i], A[i + 1])$

(2) Because the player 2 will also using optimal strategy, after play 1 pick a card, player 2 will try to minimize the remaining total card value. This leave the player 1 to pick the value that can maximize the rest value, which is minimized by player 2.

There are two cases for player 1 to pick cards:
If player 1 pick the leftmost card, then value of player 1
$C1 = A[i] + min(Val(i + 1, j), Val(i + 1, j - 1))$.
If player 1 pick the rightmost card, then value of player 1
$C2 = A[j] + min(Val(i + 1, j - 1), Val(i, j - 2))$.
Then pick $Val(i, j) = max(C1, C2)$ would give player 1 the optimal strategy with maximized total value.

Since the players are pick from both side and each time each play only pick one cards, at the end of the game, the players will end up with either remain 1 card for n is odd or 2 card for n is even. Then trivially, player 1 either pick the remaining 1 card or pick the maximum of the 2 card. This is shown by the base cases in (1)

(3) Algorithm: let the total number of car be N
For each card from left to right, store the card value in array A, where $A[i]$ represents the i^{th} from left to right card's value.$\cdots\cdots O(N)$.

```
For (int L= 0; L < N; L++) { //Line (1)
        For (int i = 0, int j = L; j<N; j++, i++){ /Line (2)
                C1 = A[i] + min(Val(i + 1, j), Val(i + 1, j - 1))
                C2 = A[j] + min(Val(i + 1, j - 1), Val(i, j - 2))
                Val(i, j) = max(C1, C2)
        }
}
```

Sicheng Chu schu37@wisc.edu Lec 001
Hiroshi Shu hshu4@wisc.edu Lec 001
Hw 05

(4) Line (1) and LIne (2) form an nested loop with complexity of $O(N^2)$. Lines below Line (2) can be done in constant time because access and retrieve value of an element in an array with index are constant time. So the overall complexity is $O(N^2)$