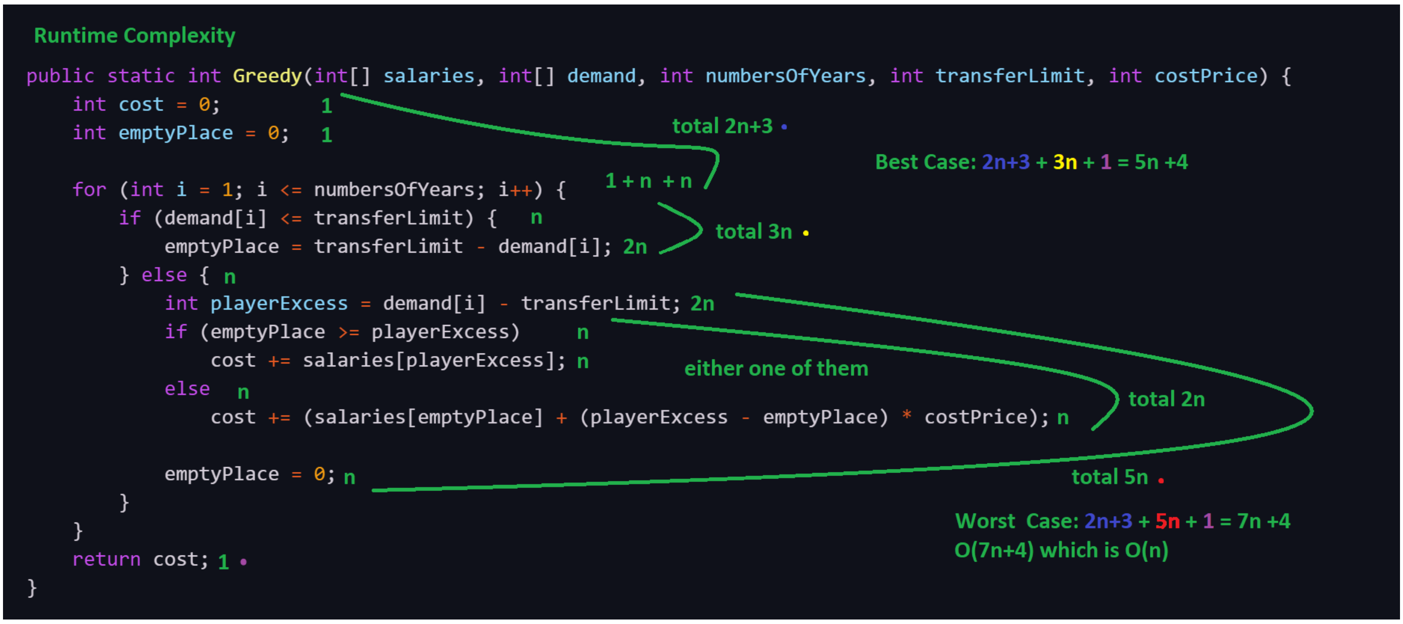
# REPORT

- **Code summary:**

  In the Greedy function, we take information as parameters such as transferLimit, numberOfYears, costPrice, and also demand and salary table as array indexing from 1. There is two variable to calculate the cost and track the previous year's unused place, both assigned 0. The algorithm starts calculation from the first year. If demand is lower or equal to the limit. It assigns unused place to emptyPlace variable. Else, first calculates the player excess. Then, check if the previous year's unused place is enough for player excess. If it is, add salary to the cost. If not add all unused place salary and coach price for remainings. Lastly, assigns 0 to emptyPlace to indicate that there is no unused place for the next years. In summary, the program calculates the minimum cost by salary prioritized because in general coach cost is higher than salary cost. That applied greedy approach.

- **Runtime Complexity:**



Runtime Complexity

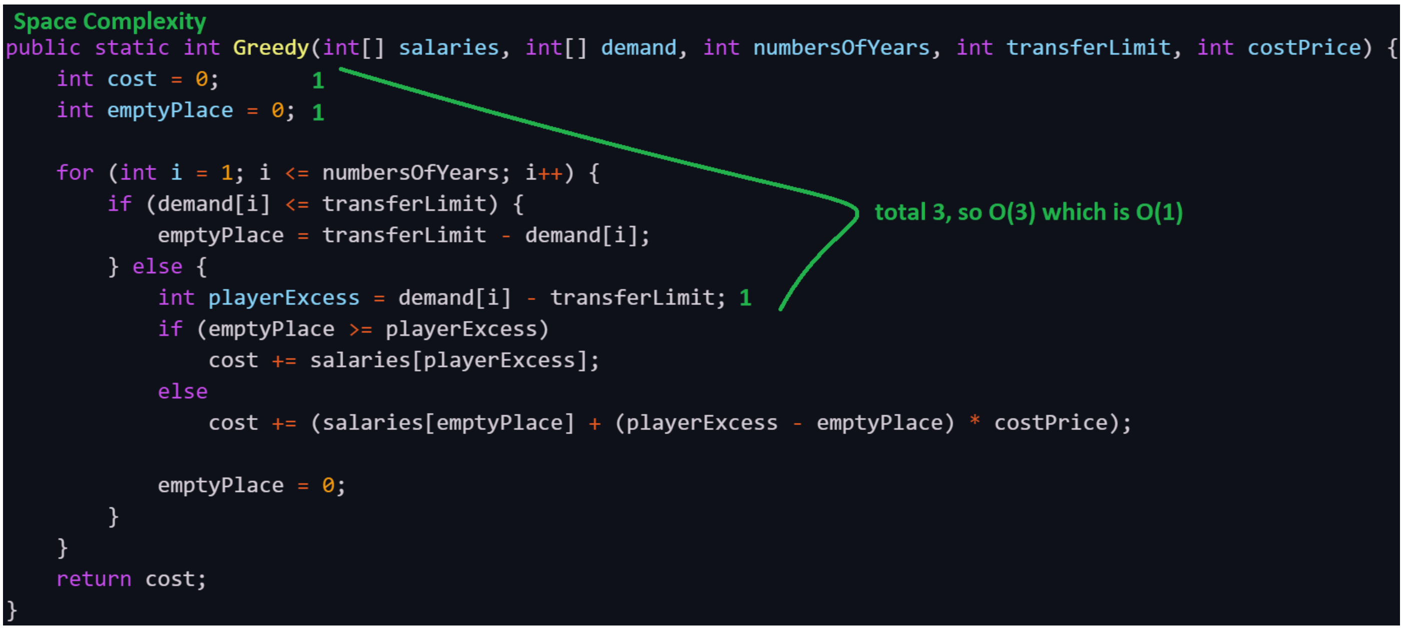
```
public static int Greedy(int[] salaries, int[] demand, int numbersOfYears, int transferLimit, int costPrice) {
    int cost = 0;
    int emptyPlace = 0;

    for (int i = 1; i <= numbersOfYears; i++) {
        if (demand[i] <= transferLimit) {
            emptyPlace = transferLimit - demand[i];
        } else {
            int playerExcess = demand[i] - transferLimit;
            if (emptyPlace >= playerExcess)
                cost += salaries[playerExcess];
            else
                cost += (salaries[emptyPlace] + (playerExcess - emptyPlace) * costPrice);

            emptyPlace = 0;
        }
    }
    return cost;
}
```

As shown in the figure, the best-case scenario is 5n+4 and the worst-case scenario is 7n+4. Considering that 5n+4 < Ω () < 7n+4, we can say the runtime complexity of this function is Ω (n) in the average case. In big-Oh, O(7n+4) = O(n).

- **Space Complexity:**



Space Complexity

```
public static int Greedy(int[] salaries, int[] demand, int numbersOfYears, int transferLimit, int costPrice) {
    int cost = 0;        1
    int emptyPlace = 0;  1

    for (int i = 1; i <= numbersOfYears; i++) {
        if (demand[i] <= transferLimit) {
            emptyPlace = transferLimit - demand[i];
        } else {
            int playerExcess = demand[i] - transferLimit; 1
            if (emptyPlace >= playerExcess)
                cost += salaries[playerExcess];
            else
                cost += (salaries[emptyPlace] + (playerExcess - emptyPlace) * costPrice);

            emptyPlace = 0;
        }
    }
    return cost;
}
```

total 3, so O(3) which is O(1)

As shown in the figure, the best-case scenario is 2, and the worst-case scenario is 3. Considering that 2 < Θ () < 3, we can say the space complexity of this function is Θ (1) in the average case. In big-Oh, O(3) = O(1).

Gürkan Bıyık 2020510019