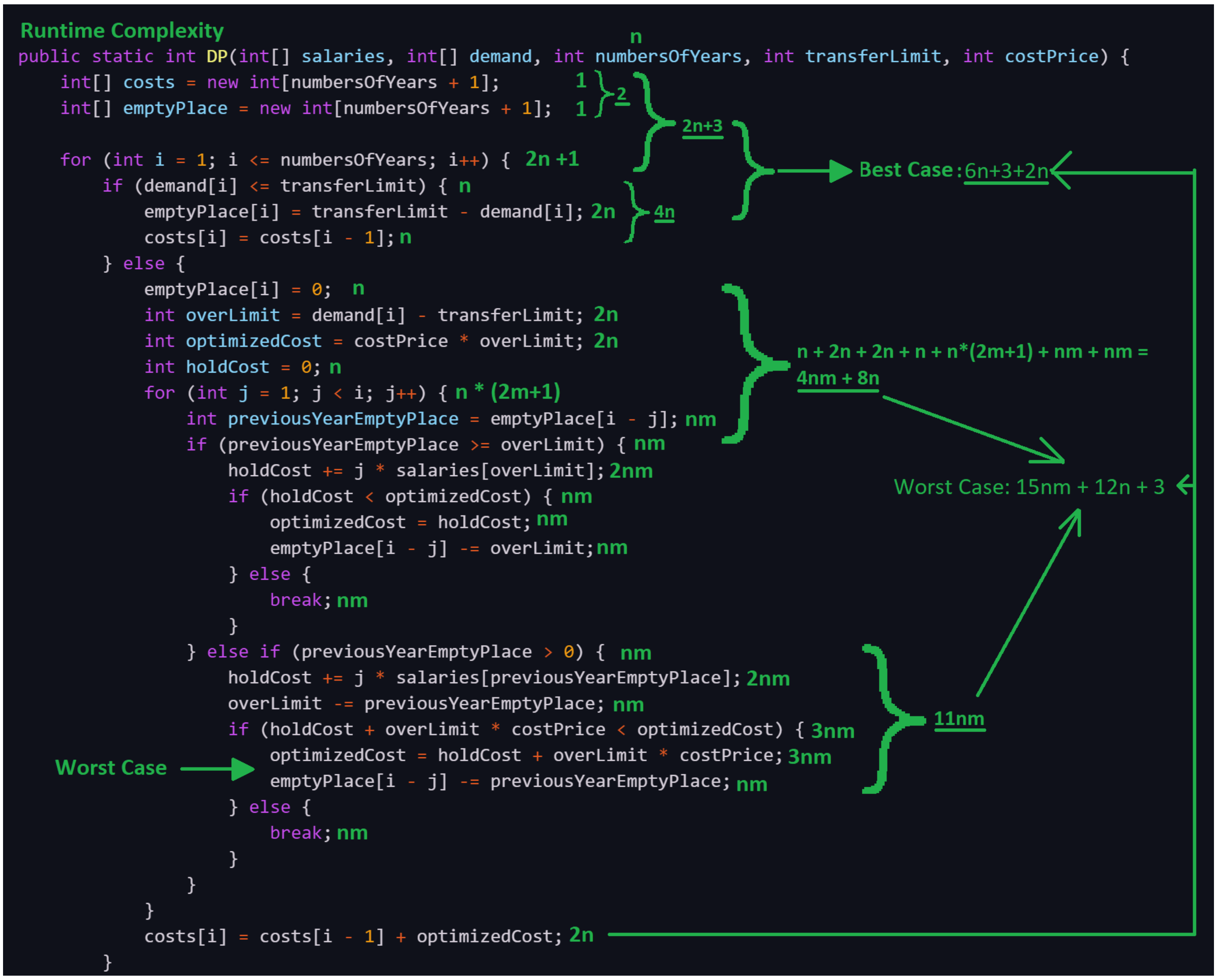
REPORT

- **Code summary:**

  In the DP function, we take information as parameters such as transferLimit, numberOfYears, costPrice, and also demand and salary table as array indexing from 1. We create a cost array to keep the optimized price for that number of year duration and an emptPlace array to track empty years to find possible optimization. The algorithm does not use the 0 index. The algorithm loops through until reaches the numberOfYear. In the loop, compared if the limit is enough. If it is, the algorithm assigns the previous year's cost and remaining places to relevant arrays. If demand is bigger than the limit, first assign 0 to the empty place for the current year, calculate player excess to overLimit, coach expense to optimizedCost, and for the calculation 0 for the holding cost. Second, it starts loops to find if there is an empty year. Then, if that year has provides player excess, holdCost is calculated. It is compared with the optimized cost. Assigned the minimum one. If there is not enough space for that year took all the empty places, and rent coaches for the remaining. If it is profitable, decrease empty places of that year and go one more previous year, and continue the same steps until the holding cost gets bigger than the optimized cost. After all these steps, the minimum cost for that year's period of time will be found and assigned it plus the previous year's cost to the cost array index. Eventually returning the last year's cost. In summary, the program calculates each cost for the current year by getting the previous cost information from the costs array plus optimized cost. That applied dynamic programming sense.

- **Runtime Complexity:**



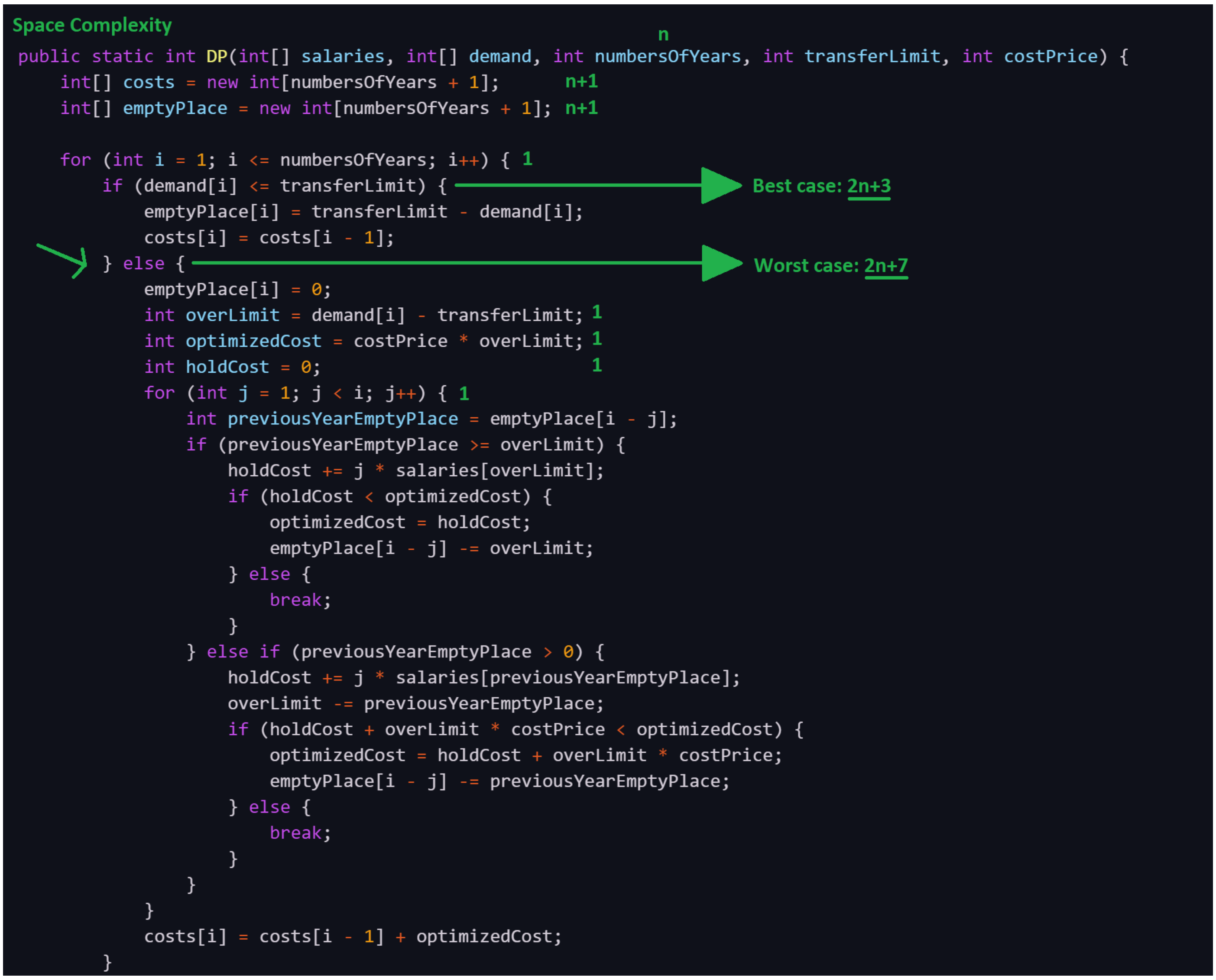
Runtime Complexity

```
public static int DP(int[] salaries, int[] demand, int numbersOfYears, int transferLimit, int costPrice) {
    int[] costs = new int[numbersOfYears + 1];
    int[] emptyPlace = new int[numbersOfYears + 1];

    for (int i = 1; i <= numbersOfYears; i++) {
        if (demand[i] <= transferLimit) {
            emptyPlace[i] = transferLimit - demand[i];
            costs[i] = costs[i - 1];
        } else {
            emptyPlace[i] = 0;
            int overLimit = demand[i] - transferLimit;
            int optimizedCost = costPrice * overLimit;
            int holdCost = 0;
            for (int j = 1; j < i; j++) {
                int previousYearEmptyPlace = emptyPlace[i - j];
                if (previousYearEmptyPlace >= overLimit) {
                    holdCost += j * salaries[overLimit];
                    if (holdCost < optimizedCost) {
                        optimizedCost = holdCost;
                        emptyPlace[i - j] -= overLimit;
                    } else {
                        break;
                    }
                } else if (previousYearEmptyPlace > 0) {
                    holdCost += j * salaries[previousYearEmptyPlace];
                    overLimit -= previousYearEmptyPlace;
                    if (holdCost + overLimit * costPrice < optimizedCost) {
                        optimizedCost = holdCost + overLimit * costPrice;
                        emptyPlace[i - j] -= previousYearEmptyPlace;
                    } else {
                        break;
                    }
                }
            }
            costs[i] = costs[i - 1] + optimizedCost;
    }
```

As shown in the figure, the best-case scenario is 8n+3 and the worst-case scenario is 15nm+12n+3 . Considering that 15nm+12n+3 < O (), we can say the runtime complexity of this function is Ω (8n+3) as Ω (n) in the best case and O(15nm+12n+3) as O(n^2) in the worst case.

- **Space Complexity:**



```java
public static int DP(int[] salaries, int[] demand, int numbersOfYears, int transferLimit, int costPrice) {
    int[] costs = new int[numbersOfYears + 1];
    int[] emptyPlace = new int[numbersOfYears + 1];

    for (int i = 1; i <= numbersOfYears; i++) {
        if (demand[i] <= transferLimit) {
            emptyPlace[i] = transferLimit - demand[i];
            costs[i] = costs[i - 1];
        } else {
            emptyPlace[i] = 0;
            int overLimit = demand[i] - transferLimit;
            int optimizedCost = costPrice * overLimit;
            int holdCost = 0;
            for (int j = 1; j < i; j++) {
                int previousYearEmptyPlace = emptyPlace[i - j];
                if (previousYearEmptyPlace >= overLimit) {
                    holdCost += j * salaries[overLimit];
                    if (holdCost < optimizedCost) {
                        optimizedCost = holdCost;
                        emptyPlace[i - j] -= overLimit;
                    } else {
                        break;
                    }
                } else if (previousYearEmptyPlace > 0) {
                    holdCost += j * salaries[previousYearEmptyPlace];
                    overLimit -= previousYearEmptyPlace;
                    if (holdCost + overLimit * costPrice < optimizedCost) {
                        optimizedCost = holdCost + overLimit * costPrice;
                        emptyPlace[i - j] -= previousYearEmptyPlace;
                    } else {
                        break;
                    }
                }
            }
            costs[i] = costs[i - 1] + optimizedCost;
        }
```

As shown in the figure, the best-case scenario is 2n+3, and the worst-case scenario is 2n+7. Considering that 2n+3 < Θ () < 2n+7, we can say the space complexity of this function is Θ (n) in the average case.

Gürkan Bıyık 2020510019