



Kuwait University

**College Engineering and Petroleum
Computer Engineering Department**

CpE 0612300 : DES.&ANAL.OF ALGORITHMS

Semester: Spring 2021/2022

Section No. 03A/ATTEND

Project Algorithm

Student Name: < Waheed Dheef>

Student Id: <2191112333>

Student Name: <Yassine Nader Serrid>

Student Id: <2181156439>

Instructor Name: <Prof. Anastasios Dimitriou>

TA Name: <Eng. Maryam Aljame>

Date: <30/5/2022>

Contents

List of Figures:	2
Table Of Terminology And Definitions:.....	3
1. Introduction:	4
1.1 Problem Summary:	4
2. Specifications:	4
2.1 Design:.....	4
2.1.1 Formulation Of Problem Using DP Recurrences:	5
2.1.2 Correctness Of Recurrence:	5
2.2 Implementation:	6
3. Testing and screenshots:	8
3.1. Test Case 1: $N = M$:	8
3.2. Test Case 2: $N > M$:	9
3.3. Test Case 3: $N < M$:	9
3.4. Test Case 4: N & M Are Large:	10
4. Conclusion:.....	10
5. Work Distribution:	11
6. Source Code:	12
7. References:	15

List of Figures:

Figure 1. DAG (Directed Acyclic Graph) Of The Design.....	5
Figure 2. initializing Instances.	6
Figure 3. Insert Method.	6
Figure 4. Method OPS(Optimal Solution)	7
Figure 5. Printing Method.....	7
Figure 6. Main Program.	8
Figure 7. Test Case 1: $N = M$	8
Figure 8. Test Case 2: $N > M$	9
Figure 9. Test Case 3: $N < M$	9
Figure 10. Test Case 4: N & M Are Large.	10

Table Of Terminology And Definitions:

DP(Dynamic Programming)	An algorithmic strategy for solving an optimization issue by breaking it down into smaller subproblems and using the fact that the best solution to the overall problem is determined by the best solution to its subproblems.
1-D Arrays	The most basic type of Array, in which the items are stored linearly and may be accessed individually by supplying the index value of each element in the array.
Spaghetti Code	Spaghetti code is a pejorative term for source code that is unorganized and difficult to maintain.
Instance	A specific realization of any object

1. Introduction:

This project aims to design an efficient Dynamic programming (DP) Algorithm and come up with a recurrence that could solve the problem and prove its correctness for the required specifications of the algorithmic problem and implement it using python programming language, in advance we should take into consideration both of space requirements for the implemented program and time complexity and analyze the running time of the algorithm in the worst case scenarios both of these considerations helps us to design an algorithm that works rapidly while also conserving memory, in the process of implementing the program we are going to use a data structure that fits the requirements of the program and consumes less space. We are going to implement the code using class and object (OOP) concept instead of using a simple function implementation to avoid having a spaghetti code, also because a function represents behavior without state; a variable represents state without behavior; and a class lets you combine both and end up with something that has both state and behavior. State is what something knows or what it has, and behavior is what it can do.

1.1 Problem Summary:

In this project we are required to design a system for a new start-up that uses drones to deliver pharmaceutical products, using our knowledge of designing Dynamic Programming (DP) algorithms and python programming language, we are going to figure out an automated way that could help the start-up company to maximize the number of delivered objects which is the optimal solution that could help the start-up company to maximize its profits, considering the limitations applied on the drone. The company have a Schedule $S(i)$ which represents the necessary number of the object that we need to deliver during hour (i) , in addition drones also have a schedule $R(i)$ which represents the number of objects that could be delivered after recharging the drone for (i) hours since the last time it was used, and we could say that the actual number of objects that will be delivered in i hours depends on $R(i)$.

-The limitations and conditions applied on the drone:

- 1- The drone must be recharged regularly after each delivery.
- 2- The drone's battery in the beginning is discharged.
- 3- If the drone is operated for its first time in the k th hour, it can deliver up to $\min(S(k), R(k))$ products.

2. Specifications:

2.1 Design:

The program design consists of a class named Drones. The class initializes an array which is called schedule and have a size of N and the other array is recharge and have of size M . We also have, two pointers were initialized pointing at the first empty index of each array. Two additional arrays of size N were initialized that will store the optimal solution, for each hour and the hours when the drone was used to achieve the optimal solution that maximizes the number of objects that will be delivered.

In the drone class, we created three methods. The first method which is name is inserts OPS and the second one is optimal_for_n. The insert method inserts an array from the user input into the schedule array or recharge array. The OPS method recalculates the optimal solution for all existing hours in the schedule array and stores the hours when the drone was used in optimal_for_n that prints the maximum number of objects the drone can deliver at hour n and the hours the drone was used for this.

2.1.1 Formulation Of Problem Using DP Recurrences:

Let P = optimal solution, i = the hour that we are trying to find an optimal solution for it, j = the hours between 0 and i which we already have the optimal solution for it, q = the maximum amount for an hour (i) to find the optimal solution for an hour (i) We set $q = 0$.

$$P(i) = \max(q, P(j) + \min(\text{schedule}(i) + \text{recharge}(i-j)))$$

To find the optimal solution for any hour (i) we must solve all (j) between 0 and (i).

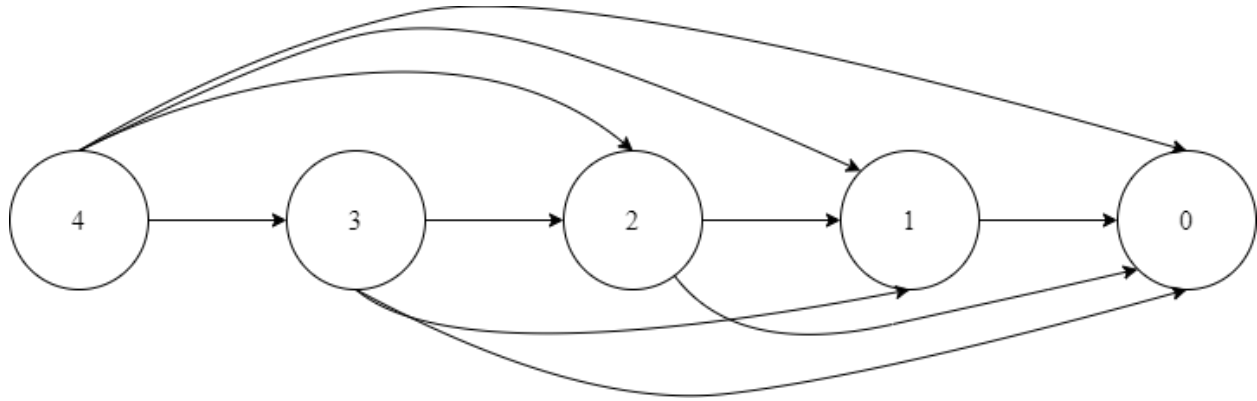


Figure 1. DAG (Directed Acyclic Graph) Of The Design.

2.1.2 Correctness Of Recurrence:

In an hour (i) the drone can only deliver the number of objects stored in array scheduled to allow even if recharge allows more. The drone can also only deliver the amount of recharge allowed even if the schedule allows more. For this, we take a minimum of the two .

To find the optimal solution for (i) we either take the minimum of both schedule [i] and recharge [i] or we send the drone at an hour (j) and let it charge ($i-j$) hours. We try this for all j in range 0 to (i) to find the maximum of them, which is the optimal solution for an hour (i).

schedule = [0 , 1 , 6 , 7 , 3], recharge = [0 , 1 , 2 , 5 , 6] (remember index = hour)

$P = [0 , 1 , 2 , 5 , ?]$

$$P(4) = P(0) + \min(\text{schedule}[4], \text{recharge}(4-0)) = 0 + \min(3, 6) = 0 + 3 = 3$$

$$P(4) = P(1) + \min(\text{schedule}[4], \text{recharge}(4-1)) = 1 + \min(3, 5) = 1 + 3 = 4$$

$$P(4) = P(2) + \min(\text{schedule}[4], \text{recharge}(4-2)) = 2 + \min(3, 2) = 2 + 2 = 4$$

$$P(4) = P(3) + \min(\text{schedule}[4], \text{recharge}(4-3)) = 5 + \min(3, 1) = 5 + 1 = 6$$

We take the max between these solutions which is when $P(4) = 6$.

2.2 Implementation:

The constructor in the class as shown below in Figure 2 initializes 6 difference instances 4 as an array and 2 as an integer. Instance schedules represent the amount the drone can deliver in a given hour. The index represents the hour. Where index 0 = hour 0, index 1 = hour 1. The index value is the amount that the drone can deliver in that hour. Schedule [2] = 6, which means the drone can deliver 6 objects in the second hour. The schedule_pointer is a pointer pointing right after the last inserted hour. By default, the last inserted hour is 0 with a 0-delivery amount. The pointer will give us huge advantages that will be shown later. recharge and recharge_pointer follow the same ideology as the previous two but for the amount the drone can deliver for M hours of recharging, recharge [2] = 2 means if the drone recharges for 2 hours, it can deliver 2 objects. As for the solution instance, we will save the maximum amount the drone can deliver in each hour. While the used array will tell us what hour the drone was used before hour N to achieve the maximum amount for hour N. The initializing instance process takes constant time, giving it time complexity. $O(1)$ Instance schedule, solution, and used arrays have a size of N while recharge has a size of M giving it a total space complexity of $3N + M$ or $O(N+M)$.

```
def __init__(self):
    self.schedule = [0] * 1000
    self.schedule_pointer = 1
    self.recharge = [0] * 100
    self.recharge_pointer = 1
    self.solution = [0] * 1000
    self.used = [0] * 1000
```

Figure 2. initializing Instances.

The first method in the class shown below in Figure 3 is called an insert. The method will take 2 arrays from the main input from the user. The array can be as big as N and M. arr1 is the array that represents the scheduled hour values. arr2 represents the recharge hour value. The array values will be entered starting from the pointer position. After entering them, the pointer will be moved by the length of the array. This way, the pointer gives us an advantage if we want to enter more hours in the array instead of starting over the method loop from the pointer to the size of the entered array. The average case should be small as we start looping from the pointer instead of the full array. However, if the user entered a large number for the first time, it could take a long time. This is the worst-case scenario of $N + M$ or $O(N+M)$. python does not make a copy array when you pass it into a function or method making the method does not consume extra space giving it space complexity of $O(1)$. At the end, the method will call the 2nd method OPS.

```
def insert(self, arr1, arr2):
    #add user input array into the right array
    if len(arr1) != 0 and self.schedule_pointer != 1000: #nothing will be added in case user array were empty or class array is full
        for M, i in zip(range(self.schedule_pointer, self.schedule_pointer + len(arr1)), range(0, len(arr1))):
            self.schedule[M] = arr1[i]
        self.schedule_pointer = self.schedule_pointer + len(arr1)

    if len(arr2) != 0 and self.recharge_pointer != 100:
        for M, j in zip(range(self.recharge_pointer, self.recharge_pointer + len(arr2)), range(0, len(arr2))):
            self.recharge[M] = arr2[j]
        self.recharge_pointer = self.recharge_pointer + len(arr2)

    self.OPS() #recalculate optimal solution
```

Figure 3. Insert Method.

From 0 to N, we then see if the max is less than the optimal solution for M + the minimum of both schedule N and recharge N – M in a case where N-M > recharge pointer – 1 means we will exceed the inserted recharge hour when we call recharge, which may result in either an error or an inaccurate number, so we call the last inserted hour in recharge. If N – M is not > recharge – 1, then we will call recharge [N-M]. Each time the check succeeds, we will save the new max and the M, which represents the hour the drone was used to achieve the current max for N. M for the max will be saved in the array used while max will be saved in array solution both in index N which represent the current hour, we are in ,after both loops finish, we should have all optimal solution for all hours in array schedule. The method does not create any lists of arrays, so it has a space complexity of O(1). Meanwhile, the inner loop is dependent on the outer loop increasing as the outer loop increases for time $\frac{N(N-1)}{2}$, which equals O(N²).

```
def OPS(self):
    for N in range(1,self.schedule_pointer): #loop from 1 to the schedule pointer to save time
        max = -9999999
        for M in range(N):
            if N - M > self.recharge_pointer-1: #N - M is greater than recharge pointer than we use the last inserted element in array recharge
                if max <= self.solution[M] + min(self.schedule[N], self.recharge[self.recharge_pointer-1]):
                    max = self.solution[M] + min(self.schedule[N], self.recharge[self.recharge_pointer-1])
                    self.used[N] = M
            else:
                #otherwise we look for the element N-M to know recharge hours
                if max <= self.solution[M] + min(self.schedule[N], self.recharge[N - M]):
                    max = self.solution[M] + min(self.schedule[N], self.recharge[N - M])
                    self.used[N] = M
        self.solution[N] = max #save the max number of object into array for each element N inside array schedule
```

Figure 4. Method OPS(Optimal Solution)

The last method in the class, as shown in Figure 5, prints the maximum number of objects from the array solution by calling solution [N]. It then saves the hour H the drone was used to achieve the optimal for N by calling used[N]. Then we find the hour when the drone was used to achieve the optimal for H by calling used [H]. And so on. When used[H] = 0, then the drone was only used for hour H to achieve optimal H. We save the used hours in a dummy array and then print them in reverse to match the test result given by the professor. The worst-case scenario (when the array schedule is full and consists of only 1), the dummy array can be as big as N, which will make its time complexity O(n). The running time in that case to find all H hours which is also take O(N).

```
def optimal_for_n(self,n):
    print("Maximum number of objects: {}".format(self.solution[n])) #print the maximum amount of object for hour n entered by the user that is saved in array solution
    print("Hours drone was used:{}_end=" " ")
    h = n
    dummy_list = []
    dummy_list.append(h)
    while self.used[h] != 0: #save all hours the drone was used to achieve the maximum for hour n in dummy list
        h = self.used[h]
        dummy_list.append(h)
    for i in range(len(dummy_list)-1,-1,-1): #print in reserve
        print(dummy_list[i], end=" ")
    print()
```

Figure 5. Printing Method.

The main program, as shown in Figure 6, will initialize the class to variable d and create an object from it. Then a while loop will ask the user to:

- 1-insert hours into the array schedule and recharge.
- 2-find the maximum number of objects as well as the hour to achieve it.
- 3-exit the program.

For 1, it will ask the user to input two arrays, one for schedule and the other one for recharge. 2 will ask the user to enter an hour to find the optimal solution. 3 will break the loop and exit the program. The main function will run if the user pleases, giving it no real running time. The first entry will create an array with no particular size entered by the user it can be small or big.

```
d = Drones() #intilize the class

while True:
    print("1- Enter 1 to insert hours")
    print("2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used")
    print("3- Enter 3 to exit the program")
    x = int(input("\nEnter your choice: "))
    if x == 1:
        #ask the user to enter array to insert the element inside to the corresponding array in the class
        print("The current hours entered in array schedule is {}".format(d.schedule_pointer - 1))
        N = list(map(int, input("Enter the amount of object schedule to deliver (remember your enteries will be saved in hour {} and onward)\nLeave space between your values: ".format(d.schedule_pointer - 1))
        print("The current hours entered in array recharge is {}".format(d.recharge_pointer - 1))
        M = list(map(int, input("Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour {} and onward)\nLeave space between yo
        d.insert(N,M)
    elif x == 2:
        #ask the user the hour he wish to find the maximum amount for
        print("The current amount of hours entered is {}".format(d.schedule_pointer-1))
        n = int(input("Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: "))
        d.optimal_for_n(n)
        print()
    elif x == 3:
        #exit the program
        break
    else:
        print("Invalid Input!\n")
```

Figure 6. Main Program.

3. Testing and screenshots:

3.1. Test Case 1: N = M:

```
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 1 2 3 4 5
The current hours entered in array recharge is 0
Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 1 2 3 4 5
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 2
the current amount of hours entered is 5
Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: 5
Maximum number of objects: 9
Hours drone was used: 2 4 5
```

Figure 7. Test Case 1: N = M.

3.2. Test Case 2: $N > M$:

It will take the greater M.

```
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 2 3 5 7 9 3 5 4 4 4
The current hours entered in array recharge is 0
Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 1 4 4 5 5
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 3
the current amount of hours entered is 10
Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: 10
Maximum number of objects: 19
Hours drone was used: 3 5 7 9 10
```

Figure 8. Test Case 2: $N > M$.

3.3. Test Case 3: $N < M$:

```
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 1 4 4 5 5
The current hours entered in array recharge is 0
Enter the amount the drone able to deliver for amount of hours the drone was charged (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 2 3 5 7 9 3 5 4 4 4
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 3
the current amount of hours entered is 5
Enter the hour you wish to find the maximum amount of delivery and the hours the drone operated to achieve it: 5
Maximum number of objects: 11
Hours drone was used: 1 4 5
```

Figure 9. Test Case 3: $N < M$.

3.4. Test Case 4: N & M Are Large:

```
C:\Users\user\AppData\Local\Programs\Python\Python310\python.exe C:/Users/user/PycharmProjects/Algo/main.py
1- Enter 1 to insert hours
2- Enter 2 to find the maximum delivery for a given hour and find the hours the drone was used
3- Enter 3 to exit the program

Enter your choice: 1
The current hours entered in array schedule is 0
Enter the amount of object schedule to deliver (remember your enteries will be saved in hour 1 and onward)
Leave space between your values: 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995 1000 1005 1010 1015 1020 1025 1030 1035 1040 1045 1050 1055 1060 1065 1070 1075 1080 1085 1090 1095 1100 1105 1110 1115 1120 1125 1130 1135 1140 1145 1150 1155 1160 1165 1170 1175 1180 1185 1190 1195 1200 1205 1210 1215 1220 1225 1230 1235 1240 1245 1250 1255 1260 1265 1270 1275 1280 1285 1290 1295 1300 1305 1310 1315 1320 1325 1330 1335 1340 1345 1350 1355 1360 1365 1370 1375 1380 1385 1390 1395 1400 1405 1410 1415 1420 1425 1430 1435 1440 1445 1450 1455 1460 1465 1470 1475 1480 1485 1490 1495 1500 1505 1510 1515 1520 1525 1530 1535 1540 1545 1550 1555 1560 1565 1570 1575 1580 1585 1590 1595 1600 1605 1610 1615 1620 1625 1630 1635 1640 1645 1650 1655 1660 1665 1670 1675 1680 1685 1690 1695 1700 1705 1710 1715 1720 1725 1730 1735 1740 1745 1750 1755 1760 1765 1770 1775 1780 1785 1790 1795 1800 1805 1810 1815 1820 1825 1830 1835 1840 1845 1850 1855 1860 1865 1870 1875 1880 1885 1890 1895 1900 1905 1910 1915 1920 1925 1930 1935 1940 1945 1950 1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025 2030 2035 2040 2045 2050 2055 2060 2065 2070 2075 2080 2085 2090 2095 2100 2105 2110 2115 2120 2125 2130 2135 2140 2145 2150 2155 2160 2165 2170 2175 2180 2185 2190 2195 2200 2205 2210 2215 2220 2225 2230 2235 2240 2245 2250 2255 2260 2265 2270 2275 2280 2285 2290 2295 2300 2305 2310 2315 2320 2325 2330 2335 2340 2345 2350 2355 2360 2365 2370 2375 2380 2385 2390 2395 2400 2405 2410 2415 2420 2425 2430 2435 2440 2445 2450 2455 2460 2465 2470 2475 2480 2485 2490 2495 2500 2505 2510 2515 2520 2525 2530 2535 2540 2545 2550 2555 2560 2565 2570 2575 2580 2585 2590 2595 2600 2605 2610 2615 2620 2625 2630 2635 2640 2645 2650 2655 2660 2665 2670 2675 2680 2685 2690 2695 2700 2705 2710 2715 2720 2725 2730 2735 2740 2745 2750 2755 2760 2765 2770 2775 2780 2785 2790 2795 2800 2805 2810 2815 2820 2825 2830 2835 2840 2845 2850 2855 2860 2865 2870 2875 2880 2885 2890 2895 2900 2905 2910 2915 2920 2925 2930 2935 2940 2945 2950 2955 2960 2965 2970 2975 2980 2985 2990 2995 3000 3005 3010 3015 3020 3025 3030 3035 3040 3045 3050 3055 3060 3065 3070 3075 3080 3085 3090 3095 3100 3105 3110 3115 3120 3125 3130 3135 3140 3145 3150 3155 3160 3165 3170 3175 3180 3185 3190 3195 3200 3205 3210 3215 3220 3225 3230 3235 3240 3245 3250 3255 3260 3265 3270 3275 3280 3285 3290 3295 3300 3305 3310 3315 3320 3325 3330 3335 3340 3345 3350 3355 3360 3365 3370 3375 3380 3385 3390 3395 3400 3405 3410 3415 3420 3425 3430 3435 3440 3445 3450 3455 3460 3465 3470 3475 3480 3485 3490 3495 3500 3505 3510 3515 3520 3525 3530 3535 3540 3545 3550 3555 3560 3565 3570 3575 3580 3585 3590 3595 3600 3605 3610 3615 3620 3625 3630 3635 3640 3645 3650 3655 3660 3665 3670 3675 3680 3685 3690 3695 3700 3705 3710 3715 3720 3725 3730 3735 3740 3745 3750 3755 3760 3765 3770 3775 3780 3785 3790 3795 3800 3805 3810 3815 3820 3825 3830 3835 3840 3845 3850 3855 3860 3865 3870 3875 3880 3885 3890 3895 3900 3905 3910 3915 3920 3925 3930 3935 3940 3945 3950 3955 3960 3965 3970 3975 3980 3985 3990 3995 4000 4005 4010 4015 4020 4025 4030 4035 4040 4045 4050 4055 4060 4065 4070 4075 4080 4085 4090 4095 4100 4105 4110 4115 4120 4125 4130 4135 4140 4145 4150 4155 4160 4165 4170 4175 4180 4185 4190 4195 4200 4205 4210 4215 4220 4225 4230 4235 4240 4245 4250 4255 4260 4265 4270 4275 4280 4285 4290 4295 4300 4305 4310 4315 4320 4325 4330 4335 4340 4345 4350 4355 4360 4365 4370 4375 4380 4385 4390 4395 4400 4405 4410 4415 4420 4425 4430 4435 4440 4445 4450 4455 4460 4465 4470 4475 4480 4485 4490 4495 4500 4505 4510 4515 4520 4525 4530 4535 4540 4545 4550 4555 4560 4565 4570 4575 4580 4585 4590 4595 4600 4605 4610 4615 4620 4625 4630 4635 4640 4645 4650 4655 4660 4665 4670 4675 4680 4685 4690 4695 4700 4705 4710 4715 4720 4725 4730 4735 4740 4745 4750 4755 4760 4765 4770 4775 4780 4785 4790 4795 4800 4805 4810 4815 4820 4825 4830 4835 4840 4845 4850 4855 4860 4865 4870 4875 4880 4885 4890 4895 4900 4905 4910 4915 4920 4925 4930 4935 4940 4945 4950 4955 4960 4965 4970 4975 4980 4985 4990 4995 5000 5005 5010 5015 5020 5025 5030 5035 5040 5045 5050 5055 5060 5065 5070 5075 5080 5085 5090 5095 5100 5105 5110 5115 5120 5125 5130 5135 5140 5145 5150 5155 5160 5165 5170 5175 5180 5185 5190 5195 5200 5205 5210 5215 5220 5225 5230 5235 5240 5245 5250 5255 5260 5265 5270 5275 5280 5285 5290 5295 5300 5305 5310 5315 5320 5325 5330 5335 5340 5345 5350 5355 5360 5365 5370 5375 5380 5385 5390 5395 5400 5405 5410 5415 5420 5425 5430 5435 5440 5445 5450 5455 5460 5465 5470 5475 5480 5485 5490 5495 5500 5505 5510 5515 5520 5525 5530 5535 5540 5545 5550 5555 5560 5565 5570 5575 5580 5585 5590 5595 5600 5605 5610 5615 5620 5625 5630 5635 5640 5645 5650 5655 5660 5665 5670 5675 5680 5685 5690 5695 5700 5705 5710 5715 5720 5725 5730 5735 5740 5745 5750 5755 5760 5765 5770 5775 5780 5785 5790 5795 5800 5805 5810 5815 5820 5825 5830 5835 5840 5845 5850 5855 5860 5865 5870 5875 5880 5885 5890 5895 5900 5905 5910 5915 5920 5925 5930 5935 5940 5945 5950 5955 5960 5965 5970 5975 5980 5985 5990 5995 6000 6005 6010 6015 6020 6025 6030 6035 6040 6045 6050 6055 6060 6065 6070 6075 6080 6085 6090 6095 6100 6105 6110 6115 6120 6125 6130 6135 6140 6145 6150 6155 6160 6165 6170 6175 6180 6185 6190 6195 6200 6205 6210 6215 6220 6225 6230 6235 6240 6245 6250 6255 6260 6265 6270 6275 6280 6285 6290 6295 6300 6305 6310 6315 6320 6325 6330 6335 6340 6345 6350 6355 6360 6365 6370 6375 6380 6385 6390 6395 6400 6405 6410 6415 6420 6425 6430 6435 6440 6445 6450 6455 6460 6465 6470 6475 6480 6485 6490 6495 6500 6505 6510 6515 6520 6525 6530 6535 6540 6545 6550 6555 6560 6565 6570 6575 6580 6585 6590 6595 6600 6605 6610 6615 6620 6625 6630 6635 6640 6645 6650 6655 6660 6665 6670 6675 6680 6685 6690 6695 6700 6705 6710 6715 6720 6725 6730 6735 6740 6745 6750 6755 6760 6765 6770 6775 6780 6785 6790 6795 6800 6805 6810 6815 6820 6825 6830 6835 6840 6845 6850 6855 6860 6865 6870 6875 6880 6885 6890 6895 6900 6905 6910 6915 6920 6925 6930 6935 6940 6945 6950 6955 6960 6965 6970 6975 6980 6985 6990 6995 7000 7005 7010 7015 7020 7025 7030 7035 7040 7045 7050 7055 7060 7065 7070 7075 7080 7085 7090 7095 7100 7105 7110 7115 7120 7125 7130 7135 7140 7145 7150 7155 7160 7165 7170 7175 7180 7185 7190 7195 7200 7205 7210 7215 7220 7225 7230 7235 7240 7245 7250 7255 7260 7265 7270 7275 7280 7285 7290 7295 7300 7305 7310 7315 7320 7325 7330 7335 7340 7345 7350 7355 7360 7365 7370 7375 7380 7385 7390 7395 7400 7405 7410 7415 7420 7425 7430 7435 7440 7445 7450 7455 7460 7465 7470 7475 7480 7485 7490 7495 7500 7505 7510 7515 7520 7525 7530 7535 7540 7545 7550 7555 7560 7565 7570 7575 7580 7585 7590 7595 7600 7605 7610 7615 7620 7625 7630 7635 7640 7645 7650 7655 7660 7665 7670 7675 7680 7685 7690 7695 7700 7705 7710 7715 7720 7725 7730 7735 7740 7745 7750 7755 7760 7765 7770 7775 7780 7785 7790 7795 7800 7805 7810 7815 7820 7825 7830 7835 7840 7845 7850 7855 7860 7865 7870 7875 7880 7885 7890 7895 7900 7905 7910 7915 7920 7925 7930 7935 7940 7945 7950 7955 7960 7965 7970 7975 7980 7985 7990 7995 8000 8005 8010 8015 8020 8025 8030 8035 8040 8045 8050 8055 8060 8065 8070 8075 8080 8085 8090 8095 8100 8105 8110 8115 8120 8125 8130 8135 8140 8145 8150 8155 8160 8165 8170 8175 8180 8185 8190 8195 8200 8205 8210 8215 8220 8225 8230 8235 8240 8245 8250 8255 8260 8265 8270 8275 8280 8285 8290 8295 8300 8305 8310 8315 8320 8325 8330 8335 8340 8345 8350 8355 8360 8365 8370 8375 8380 8385 8390 8395 8400 8405 8410 8415 8420 8425 8430 8435 8440 8445 8450 8455 8460 8465 8470 8475 8480 8485 8490 8495 8500 8505 8510 8515 8520 8525 8530 8535 8540 8545 8550 8555 8560 8565 8570 8575 8580 8585 8590 8595 8600 8605 8610 8615 8620 8625 8630 8635 8640 8645 8650 8655 8660 8665 8670 8675 8680 8685 8690 8695 8700 8705 8710 8715 8720 8725 8730 8735 8740 8745 8750 8755 8760 8765 8770 8775 8780 8785 8790 8795 8800 8805 8810 8815 8820 8825 8830 8835 8840 8845 8850 8855 8860 8865 8870 8875 8880 8885 8890 8895 8900 8905 8910 8915 8920 8925 8930 8935 8940 8945 8950 8955 8960 8965 8970 8975 8980 8985 8990 8995 9000 9005 9010 9015 9020 9025 9030 9035 9040 9045 9050 9055 9060 9065 9070 9075 9080 9085 9090 9095 9100 9105 9110 9115 9120 9125 9130 9135 9140 9145 9150 9155 9160 9165 9170 9175 9180 9185 9190 9195 9200 9205 9210 9215 9220 9225 9230 9235 9240 9245 9250 9255 9260 9265 9270 9275 9280 9285 9290 9295 9300 9305 9310 9315 9320 9325 9330 9335 9340 9345 9350 9355 9360 9365 9370 9375 9380 9385 9390 9395 9400 9405 9410 9415 9420 9425 9430 9435 9440 9445 9450 9455 9460 9465 9470 9475 9480 9485 9490 9495 9500 9505 9510 9515 9520 9525 9530 9535 9540 9545 9550 9555 9560 9565 9570 9575 9580 9585 9590 9595 9600 9605 9610 9615 9620 9625 9630 9635 9640 9645 9650 9655 9660 9665 9670 9675 9680 9685 9690 9695 9700 9705 9710 9715 9720 9725 9730 9735 9740 9745 9750 9755 9760 9765 9770 9775 9780 9785 9790 9795 9800 9805 9810 9815 9820 9825 9830 9835 9840 9845 9850 9855 9860 9865 9870 9875 9880 9885 9890 9895 9900 9905 9910 9915 9920 9925 9930 9935 9940 9945 9950 9955 9960 9965 9970 9975 9980 9985 9990 9995 10000 10005 10010 10015 10020 10025 10030 10035 10040 10045 10050 10055 10060 10065 10070 10075 10080 10085 10090 10095 10100 10105 10110 10115 10120 10125 10130 10135 10140 10145 10150 10155 10160 10165 10170 10175 10180 10185 10190 10195 10200 10205 10210 10215 10220 10225 10230 10235 10240 10245 10250 10255 10260 10265 10270 10275 10280 10285 10290 10295 10300 10305 10310 10315 10320 10325 10330 10335 10340 10345 10350 10355 10360 10365 10370 10375 10380 10385 10390 10395 10400 10405 10410 10415 10420 10425 10430 10435 10440 10445 10450 10455 10460 10465 10470 10475 10480 10485 10490 10495 10500 10505 10510 10515 10520 10525 10530 10535 10540 10545 10550 10555 10560 10565 10570 10575 10580 10585 10590 10595 10600 10605 10610 10615 10620 10625 10630 10635 10640 10645 10650 10655 10660 10665 10670 10675 10680 10685 10690 10695 10700 10705 10710 10715 10720 10725 10730 10735 10740 10745 10750 10755 10760 10765 10770 10775 10780 10785 10790 10795 10800 10805 10810 10815 10820 10825 10830 10835 10840 10845 10850 10855 10860 10865 10870 10875 10880 10885 10890 10895 10900 10905 10910 10915 10920 10925 10930 10935 10940 10945 10950 10955 10960 10965 10970 10975 10980 10985 10990 10995 11000 11005 11010 11015 11020 11025 11030 11035 11040 11045 11050 11055 11060 11065 11070 11075 11080 11085 11090 11095 11100 11105 11110 11115 11120 11125 11130 11135 11140 11145 11150 11155 11160 11165 11170 11175 11180 11185 11190 11195 11200 11205 11210 11215 11220 11225 11230 11235 11240 11245 11250 11255 11260 11265 11270 11275 11280 11285 11290 11295 11300 11305 11310 11315 11320 11325 11330 11335 11340 11345 11350 11355 11360 11365 11370 11375 11380 11385 11390 11395 11400 11405 11410 11415 11420 11425 11430 11435 11440 11445 11450 11455 11460 11465 11470 11475 11480 11485 11490 11495 11500 11505 11510 11515 11520 11525 11530 11535 11540 11545 11550 11555 11560 11565 11570 11575 11580 11585 11590 11595 11600 11605 11610 11615 11620 11625 11630 11635 11640 11645 11650 11655 11660 11665 11670 11675 11680 11685 11690 11695 11700 11705 11710 11715 11720 11725 11730 11735 11740 11745 11750 11755 11760 11765 11770 11775 11780 11785 11790 11795 11800 11805 11810 11815 11820 11825 11830 11835 11840 11845 11850 11855 11860 11865 11870 11875 11880 11885 11890 11895 11900 11905 11910 11915 11920 11925 11930 11935 11940 11945 11950 11955 11960 11965 11970 11975 11980 11985 11990 11995 12000
```

5. Work Distribution:

Data & Time	26/May/2022 – 4:30 PM
Meeting Held By	Yassin Nader – Waheed Dheef
<ul style="list-style-type: none">-Figuring out how to get the array as an input from the user.-Analyzing and reading the problem description.-Finding the Dynamic Programming (DP) Recurrence and test its correctness.-Decision and Dividing the work.	

Data & Time	27/May/2022 – 11:30 PM
Meeting Held By	Yassin Nader – Waheed Dheef
<ul style="list-style-type: none">-Figuring out the best options we could use to implement the program.-Implementing the program and testing it and analyzing its time and space complexity.	

Data & Time	28-29-30/May/2022 – 3:30 PM
Meeting Held By	Yassin Nader – Waheed Dheef
<ul style="list-style-type: none">-Writing the report.	

6. Source Code:

Source Code

```
class Drones():
    def __init__(self):
        #intilize all requested array and
        needed array and pointers
        self.schedule = [0] * 1000
        self.schedule_pointer = 1 #will always point at the last
        inserted element + 1
        self.recharge = [0] * 100
        self.recharge_pointer = 1
        self.solution = [0] * 1000
        self.used = [0] * 1000

    def insert(self,arr1,arr2):
        #add user
        input array into the right array
        if len(arr1) != 0 and self.schedule_pointer != 1000: #nothing
        will be added in case user array were empty or class array is full
            for N , i in
            zip(range(self.schedule_pointer,self.schedule_pointer+len(arr1)),range
            (0,len(arr1))):
                self.schedule[N]= arr1[i]
                self.schedule_pointer = self.schedule_pointer + len(arr1)

            if len(arr2) != 0 and self.recharge_pointer != 100:
                for M , j in
                zip(range(self.recharge_pointer,self.recharge_pointer +
                len(arr2)),range(0,len(arr2))):
                    self.recharge[M]=arr2[j]
                    self.recharge_pointer = self.recharge_pointer + len(arr2)

            self.OPS() #recalculate optimal solution

    def OPS(self):
        for N in range(1,self.schedule_pointer):
            #loop from 1 to
            the schedule pointer to save time
            max = -9999999
            for M in range(N):
                if N - M > self.recharge_pointer-1: #N - M is greater
                than recharge pointer than we use the last inserted element in array
                recharge
                    if max <= self.solution[M] + min(self.schedule[N],
                    self.recharge[self.recharge_pointer 1]):
                        max = self.solution[M] + min(self.schedule[N],
                        self.recharge[self.recharge_pointer 1])
                        self.used[N] = M
                else:
                    #otherwise we look
                    for the element N-M to know recharge hours
```

```

        if max <= self.solution[M] + min(self.schedule[N],
self.recharge[N - M]):
            max = self.solution[M] + min(self.schedule[N],
self.recharge[N - M])
            self.used[N] = M

        self.solution[N] =max                #save the max
number of objects into array for each element N inside array schedule

    def optimal_for_n(self,n):
        print("Maximum number of objects:
{}".format(self.solution[n]))                #print the maximum amount of
object for hour n entered by the user that is saved in array solution
        print("Hours drone was used:",end=" ")
        h = n
        dummy_list = []
        dummy_list.append(h)
        while self.used[h] != 0:                #save all hours
the drone was used to achieve the maximum for hour n in dummy list
            h = self.used[h]
            dummy_list.append(h)
        for i in range(len(dummy_list)-1,-1,-1):    #print in reserve
            print(dummy_list[i], end = " ")

        print()

#after we finish the class, we will create the main

d = Drones()    #intlize the class

while True:
    print("1- Enter 1 to insert hours")
    print("2- Enter 2 to find the maximum delivery for a given hour
and find the hours the drone was used")
    print("3- Enter 3 to exit the program")
    x = int(input("\nEnter your choice: "))
    if x == 1:                #ask the user to enter
array to insert the element inside to the corosponding array in the
class
        print("The current hours entered in array schedule is
{}".format(d.schedule_pointer - 1))
        N = list(map(int, input("Enter the amount of object schedule
to deliver (remember your enteries will be saved in hour {} and
onward)\nLeave space between your values:
".format(d.schedule_pointer)).strip().split())) # Use Space to enter
your values

```

```

        print("The current hours entered in array recharge is
{}".format(d.recharge_pointer - 1))
        M = list(map(int, input("Enter the amount the drone able to
deliver for amount of hours the drone was charged (remember your
enteries will be saved in hour {} and onward)\nLeave space between
your values: ".format(d.recharge_pointer)).strip().split()))
        d.insert(N,M)
    elif x == 2:                                     #ask the user the hour he
wish to find the maximum amount for
        print("the current amount of hours entered is {}".
format(d.schedule_pointer-1))
        n = int(input("Enter the hour you wish to find the maximum
amount of delivery and the hours the drone operated to achieve it: "))
        d.optimal_for_n(n)
        print()
    elif x == 3:                                     #exit the program
        break
    else:
        print("Invalid Input!\n")

```

7. References:

- 1- What is Dynamic Programming? - Grokking Dynamic Programming Patterns for Coding Interviews. (2022). Retrieved 27 May 2022, from <https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2G1pAq0000>
- 2- One Dimensional Array in C++. (2021, May 24). Toppr-Guides. <https://www.toppr.com/guides/computer-science/programming-in-c-/structured-data-type/one-dimensional-array/#:~:text=A%20One%2DDimensional%20Array%20is,element%20stored%20in%20the%20array>
- 3- Upadhyay, S. (2021). Time and Space complexity in Data Structure | Simplilearn. Retrieved from https://www.simplilearn.com/tutorials/data-structure-tutorial/time-and-space-complexity#what_is_time_complexityA
- 4- (2022). Retrieved 27 May 2022, from https://www.reddit.com/r/learnpython/comments/1mc8ih/why_should_you_use_classes_instead_of_functions/
- 5- Spaghetti code - Wikipedia. (2022). Retrieved 27 May 2022, from https://en.wikipedia.org/wiki/Spaghetti_code
- 6- What is the meaning of instance?. (2022). Retrieved 28 May 2022, from <https://www.techtarget.com/whatis/definition/instance#:~:text=What%20is%20an%20instance%3F,realized%20instance%20is%20called%20instantiation.>