Final Project

1. Implement the algorithm:

Added the code to GitHub repository. My GitHub username: JoyeeBan

2. Five Test Cases of varying sizes:

I implemented the test cases in my actual code. My test cases are of size 10,100,1000,10000,100000. When you run the program, it asks to input the size of the list.

For Brute Force:

n=10

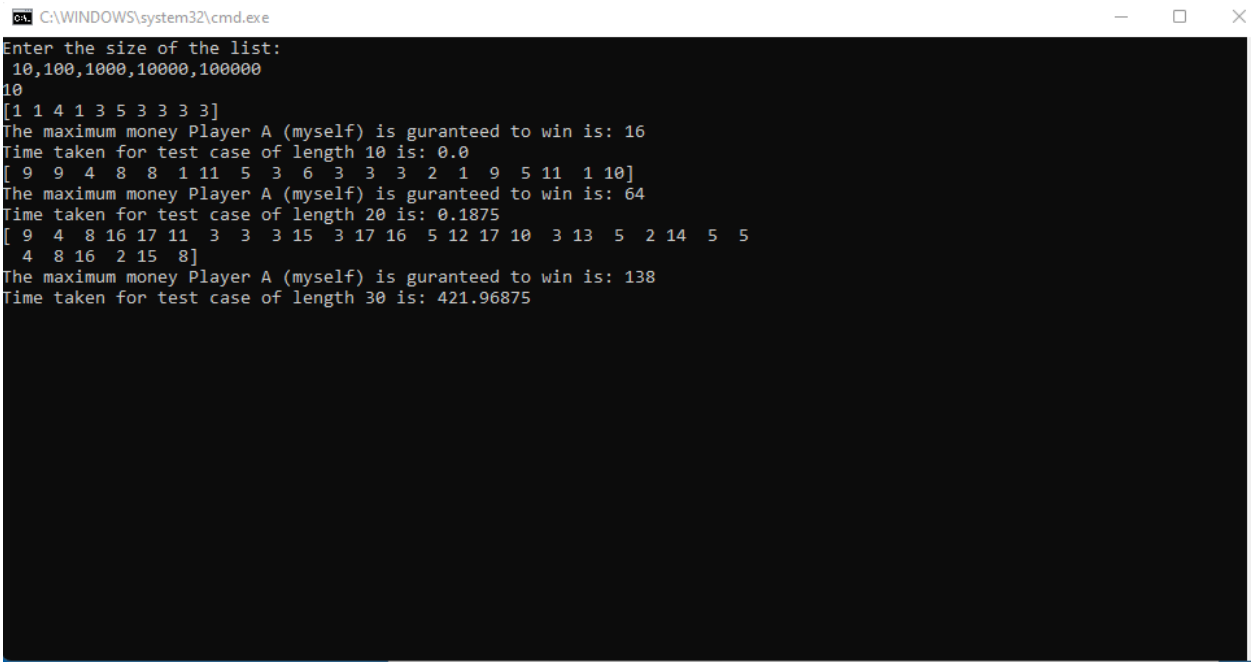

Fig 1

For the other test sizes, n=100,1000,10000,100000, Brute Force takes a huge amount of time to run as the time complexity of the algorithm is exponential. Hence, I have just attached my screenshot for test case n=10.

For Greedy Approach:

n=10:

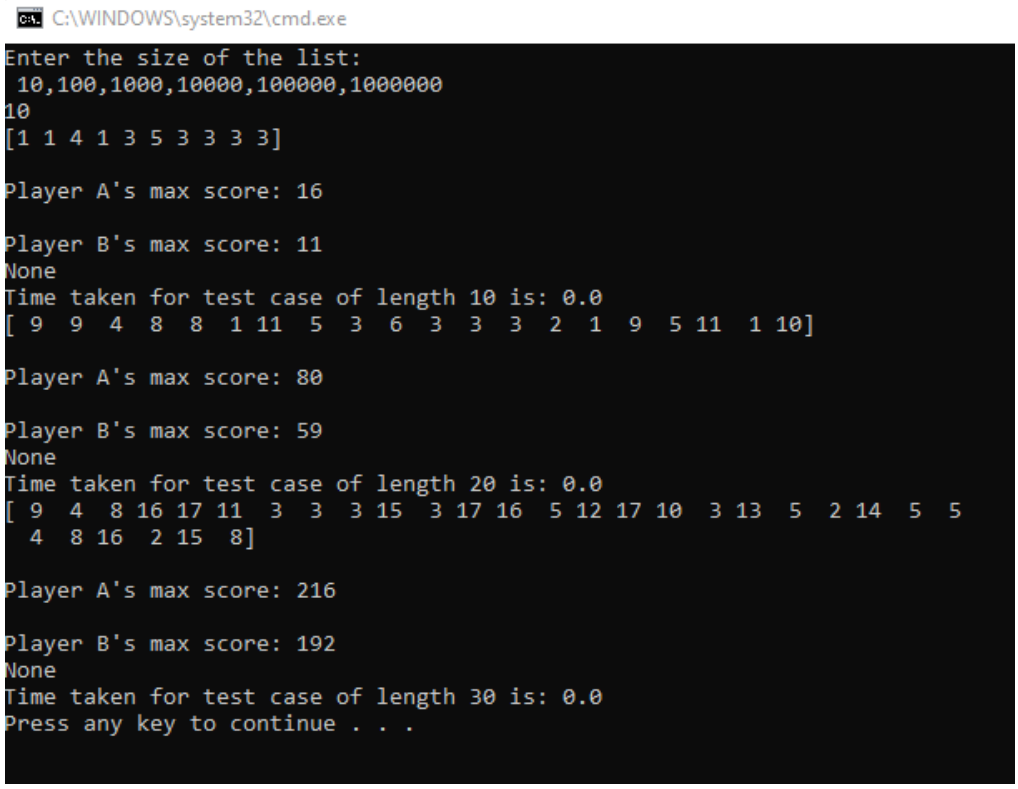

Fig 2

n=100:

```
 10,100,1000,10000,100000,1000000
100
[1 1 4 1 3 5 3 3 3 3 2 1 1 5 4 5 3 1 4 2 3 4 5 5 2 4 5 5 4 4 4 2 2 4 1 3 2
 2 4 3 4 1 2 1 5 3 1 1 3 3 2 1 3 2 5 3 1 4 4 4 1 2 5 3 4 4 5 3 5 4 2 1 5 4
 5 3 3 5 3 5 2 3 3 1 3 3 4 5 3 5 4 2 1 1 2 3 5 3 4 4]

Player A's max score: 158

Player B's max score: 151
None
Time taken for test case of length 100 is: 0.0
[ 9  9  4  8  8  1 11  5  3  6  3  3  3  2  1  9  5 11  1 10  7  3  5  2
  6  4  5  5  4  8  2  2  8  8  1  3 10 10  4  3  6  9  2  1  8  7  3  1
  9  3  6 11  2  9 11  2  6  5  3  9  4  6  1 10 11  4  7  4  5 11  8  7
  4 10  1  5  5  6  8  7  7  3 11  5  3  8  2 11  7 11  7  1 11  8  3  4
  6  5  3  5  4  8 10  1  1  6 10  7  7  6  7 11  5 11  8  4 10  3  4  9
  8  2  6 10  4  1  7  3  4  5 11  9 10  9  6  3  8  6 10  1 10  9  7  3
  1  6  4 11  3  4  7  5  2  4  2  5  9  9 11  3  3  8  3  2  3  8  2 11
 11  1 11  6  4  6  3  7  2 11  2  6 10  3 11  6  7  5  7  8 10  8  4  1
  3 11  6  2  2 11  1  4]

Player A's max score: 757

Player B's max score: 722
None
Time taken for test case of length 200 is: 0.0
[ 9  4  8 16 17 11  3  3  3 15  3 17 16  5 12 17 10  3 13  5  2 14  5  5
  4  8 16  2 15  8 17  3 10  3 15 17 16  8 14  7 13  1  3 11  9 14 11  5
  9 15  1 14 13 11  4  7  4 16 11 16  4 10 17 12  5  6  8  7  3 11 14 13
  2  7 11  1  3  5  5  4 10 17 17  7  6  7  8 12  3 14  6 14  4 17 13 14
 13  4 11  9  9 13 13  1 10  9  3 17  6  3  4 15 14  2 15  2  9  8  3  3
 14  1 11  4  6  7 11 14  3 16 13 12  6 13  7 17  2 11 13 17  5 11  6 17
 17  4  3  4 17  7  3  1 16 17 10 16  3  4  7 12  5  4  5 10  5  8  5 11
 14 17  1  7  9 15  7  6  1  9  9  6 10 13 13  3 14  2  5  1 11 10  2  3
 13  8  5  4  4  5  8 11 16 13 14  2 16 11  9  9 14 16 14 17 11  7 17  6
 16  4  6 11 12 17 17  7 10 13  9  1 15 11  7  6  8  1 14  8 15 17  6 17
  3 17 16  4 13  6  3  1 14 10 14  4 16 16 14 12  6  8  7  6  1 10 14  2
  5  2  1  7  4  4 10 15 15  8  9 14 13  4  3 12  4  3  1  7 12 13  2  3
 12  7 13  3  5  6  8  6  5 17  2 14]

Player A's max score: 2092

Player B's max score: 2070
```

Fig 3

n=1000:

```
Enter the size of the list:
 10,100,1000,10000,100000,1000000
1000
[1 1 4 1 3 5 3 3 3 3 2 1 1 5 4 5 3 1 4 2 3 4 5 5 2 4 5 5 4 4 4 2 2 4 1 3 2
 2 4 3 4 1 2 1 5 3 1 1 3 3 2 1 3 2 5 3 1 4 4 4 1 2 5 3 4 4 5 3 5 4 2 1 5 4
 5 3 3 5 3 5 2 3 3 1 3 3 4 5 3 5 4 2 1 1 2 3 5 3 4 4 2 3 4 1 2 2 4 1 5 3 5
 4 5 5 3 1 2 1 5 3 5 5 2 1 2 1 5 3 1 4 3 3 4 5 2 4 2 5 1 1 3 5 3 3 3 2 3 2
 3 3 1 3 4 5 3 2 3 2 2 3 5 3 4 5 5 2 4 1 3 3 2 2 3 5 1 4 4 5 3 1 1 1 2 4 3
 4 2 1 3 1 1 2 2 3 4 4 2 5 4 3 2 5 5 2 5 4 5 4 3 5 4 3 1 3 1 1 2 5 1 1 4 1
 2 5 4 5 3 1 2 5 1 4 3 2 2 3 5 5 4 4 3 4 5 3 5 5 5 2 2 1 3 4 1 1 1 2 1 3 1
 5 5 2 4 5 4 3 4 4 1 1 3 2 2 5 3 1 4 1 4 2 4 3 4 3 5 4 1 1 3 5 1 1 2 3 2 1
 4 4 5 2 3 1 4 2 5 4 3 5 4 2 1 2 5 5 1 4 4 1 2 2 5 4 2 1 2 3 2 4 5 4 5 4 4
 3 5 2 3 3 2 3 1 3 5 4 3 2 4 5 3 4 3 1 2 4 1 3 3 5 5 2 2 3 4 3 5 3 5 2 5 1
 2 2 5 1 2 1 5 4 3 3 4 3 4 5 4 1 2 4 3 5 4 2 1 2 4 2 3 3 3 2 3 2 1 2 1 3 2
 2 3 3 1 4 4 3 5 3 1 2 4 5 3 2 5 1 5 2 4 4 1 1 1 1 5 1 2 2 4 5 4 1 1 1 4 2
 3 5 2 5 2 3 5 3 4 3 3 5 4 5 4 4 2 2 4 2 4 3 4 1 2 5 3 4 2 2 1 3 3 4 5 2 5
 2 2 3 4 2 5 5 1 5 1 3 5 5 1 3 4 1 3 5 4 5 2 1 1 4 3 4 1 1 3 4 3 3 4 5 5 4
 5 4 3 2 4 2 4 3 5 4 3 5 1 2 3 5 1 4 1 4 5 4 5 1 3 4 4 5 4 4 4 2 3 5 4 5 1
 2 4 4 3 4 2 1 5 1 4 4 3 5 1 2 4 1 1 1 5 1 1 4 2 3 3 4 1 5 4 5 2 5 5 5 4 5
 3 2 2 4 4 4 4 3 3 5 1 2 2 2 4 5 2 2 4 4 2 3 5 3 3 5 4 5 4 1 1 4 5 3 4 3 5
 1 1 1 2 4 2 4 2 5 5 2 2 1 2 2 4 2 2 4 4 5 4 1 4 4 4 1 5 1 5 5 3 3 5 4 4 3
 5 5 1 3 4 5 3 3 5 2 1 2 5 1 1 2 5 4 5 2 2 5 2 5 2 3 2 3 5 1 2 3 2 4 1 5 3
 1 1 5 1 5 2 1 3 1 5 3 3 2 1 3 3 5 2 2 3 3 4 4 2 4 1 2 5 3 5 1 2 4 5 4 5 2
 5 3 3 4 3 1 4 2 1 2 3 4 5 1 3 1 5 2 2 4 2 4 5 2 5 3 4 5 2 4 1 5 1 2 1 4 1
 2 4 3 5 2 5 4 4 5 2 3 2 5 3 3 1 3 5 1 5 3 2 4 5 5 4 2 3 2 4 4 3 1 3 5 3 4
 3 1 5 5 2 5 5 5 4 2 1 1 5 1 1 4 1 4 5 1 1 4 1 3 3 1 4 2 4 2 3 5 3 4 5 3 2
 4 5 1 2 5 5 3 3 4 4 1 1 4 1 5 2 2 1 5 3 1 3 5 2 4 4 1 5 1 3 2 3 3 1 5 1 1
 4 5 2 4 4 5 5 2 1 1 4 3 1 5 5 2 4 3 5 2 1 2 5 3 1 5 3 5 4 5 5 5 1 2 1 2 2
 3 2 3 4 5 4 4 2 5 2 4 1 4 1 5 5 1 2 1 1 5 1 4 4 3 4 1 2 3 2 1 3 3 5 2 2 3
 3 1 4 4 5 1 1 5 3 4 5 4 4 5 5 4 1 2 2 2 1 5 1 3 4 4 5 1 1 3 3 3 4 3 3 5 4
 3]

Player A's max score: 1528

Player B's max score: 1543
None
Time taken for test case of length 1000 is: 0.0
[9 9 4 ... 8 9 4]

Player A's max score: 7576

Player B's max score: 7601
```

Fig 4.1

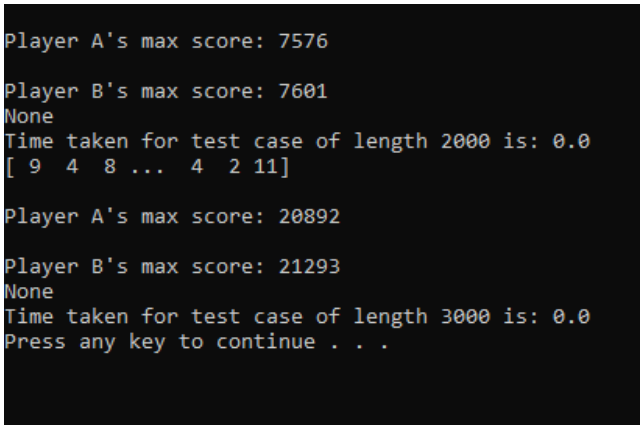

Fig 4.2

n=10000:

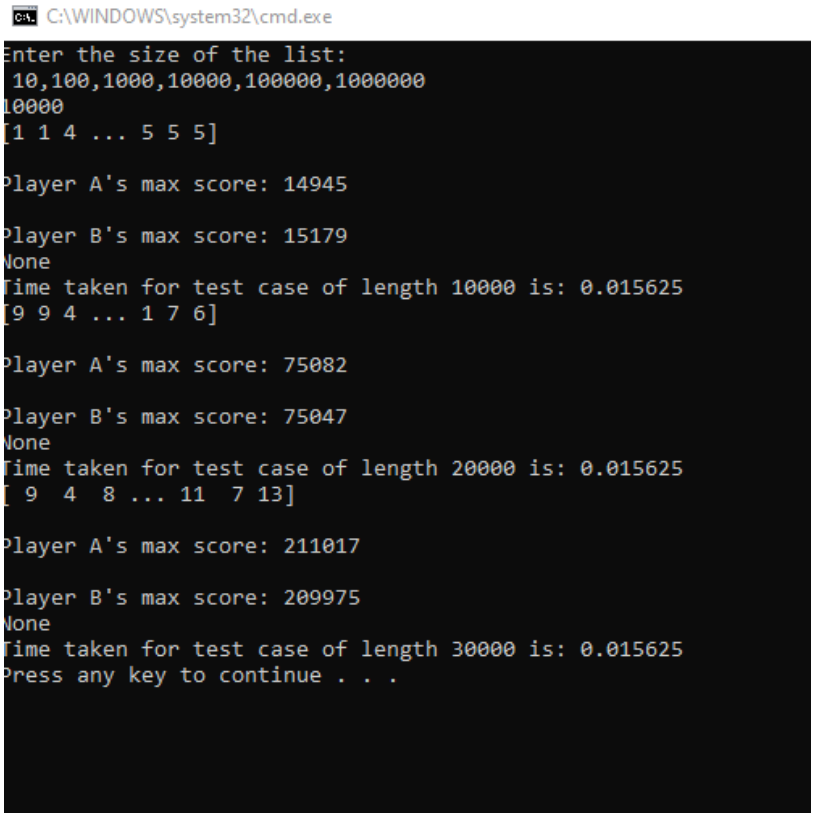

Fig 5

n=100000:

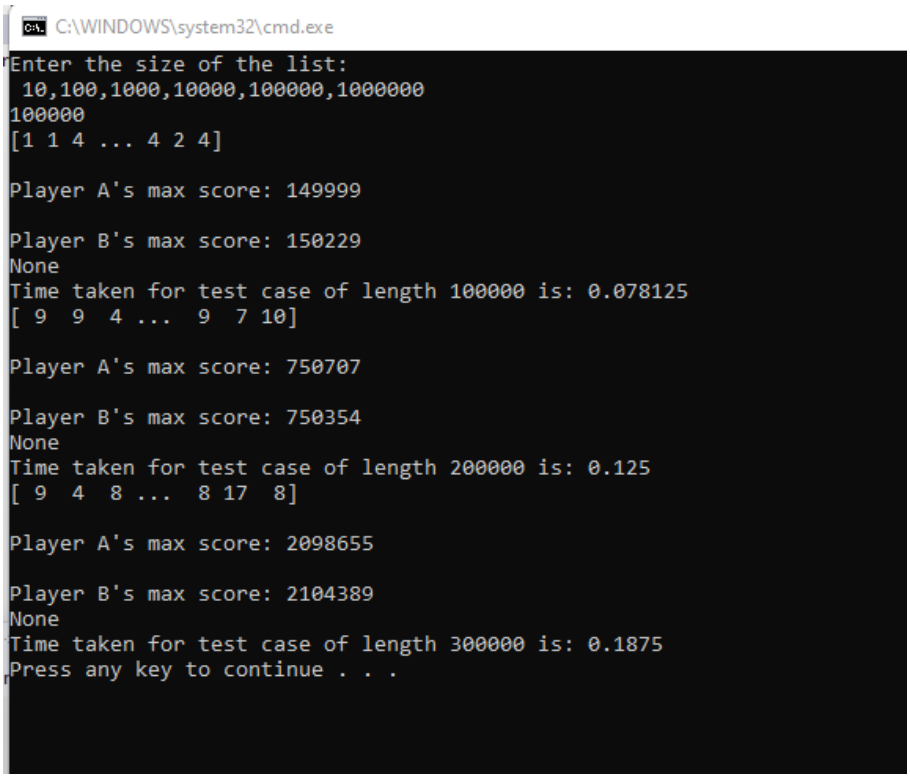

Fig 6

As the time complexity of the algorithm is O(n), linear, so it does not take that much time to run the algorithm on all the five test cases. However, this algorithm does not always return the same answer as the other (Brute Force and Dynamic) would return on the same list of coins (shown in section 2a).

For Dynamic Approach:

n=10:

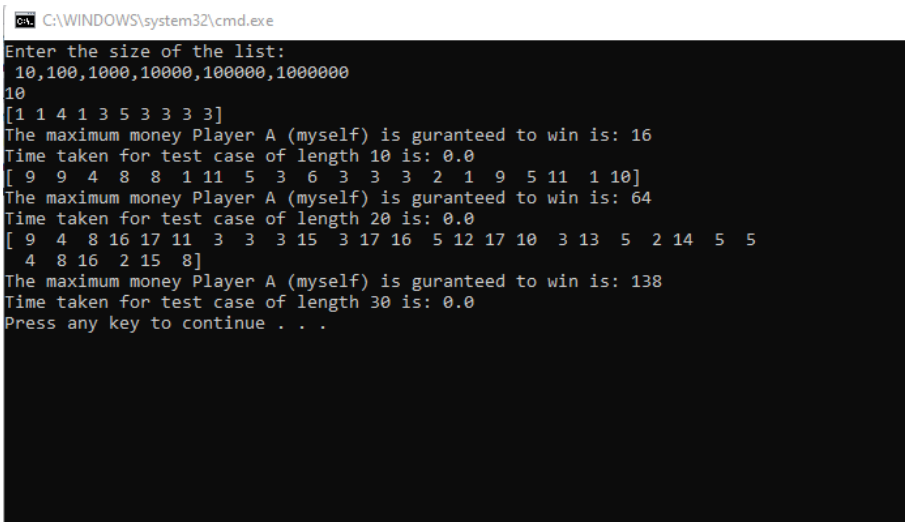

Fig 7

n=100:

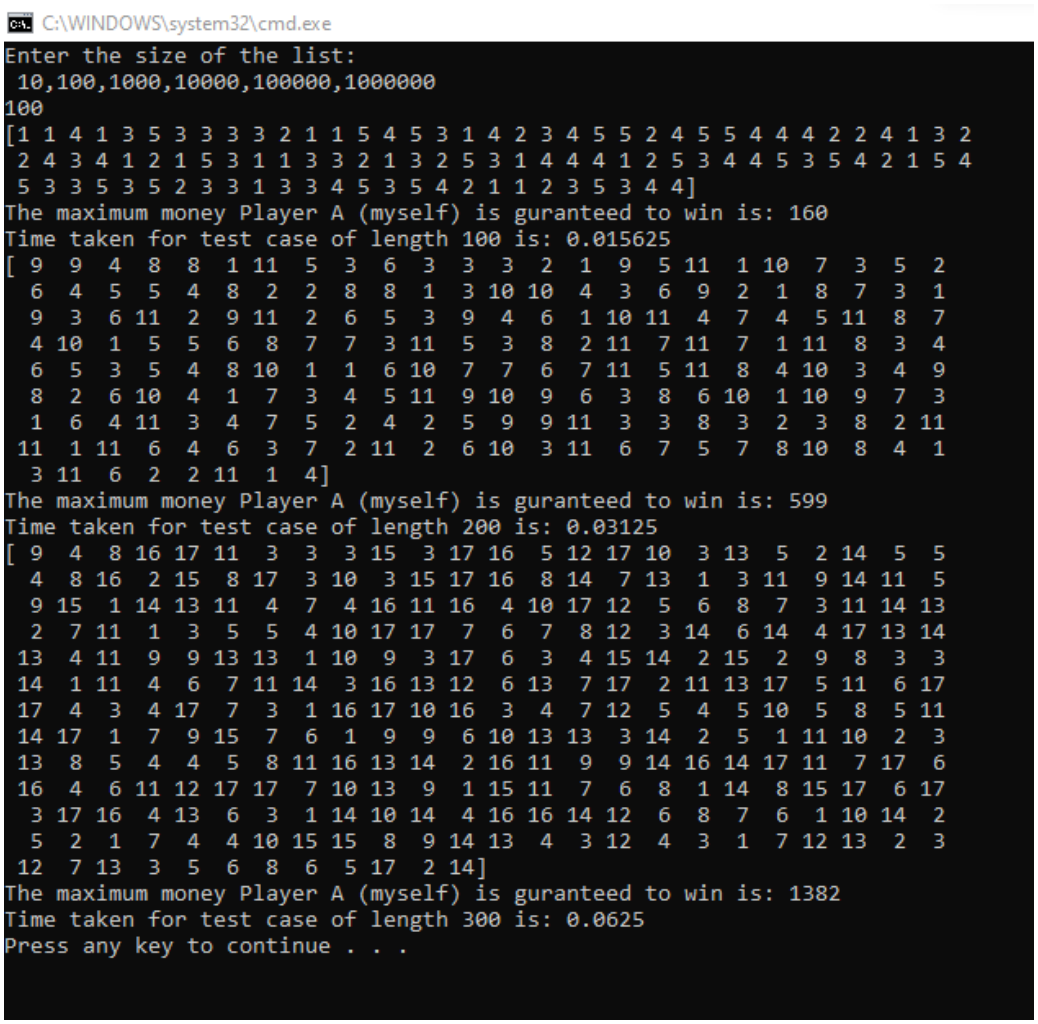

Fig 8

n=1000:

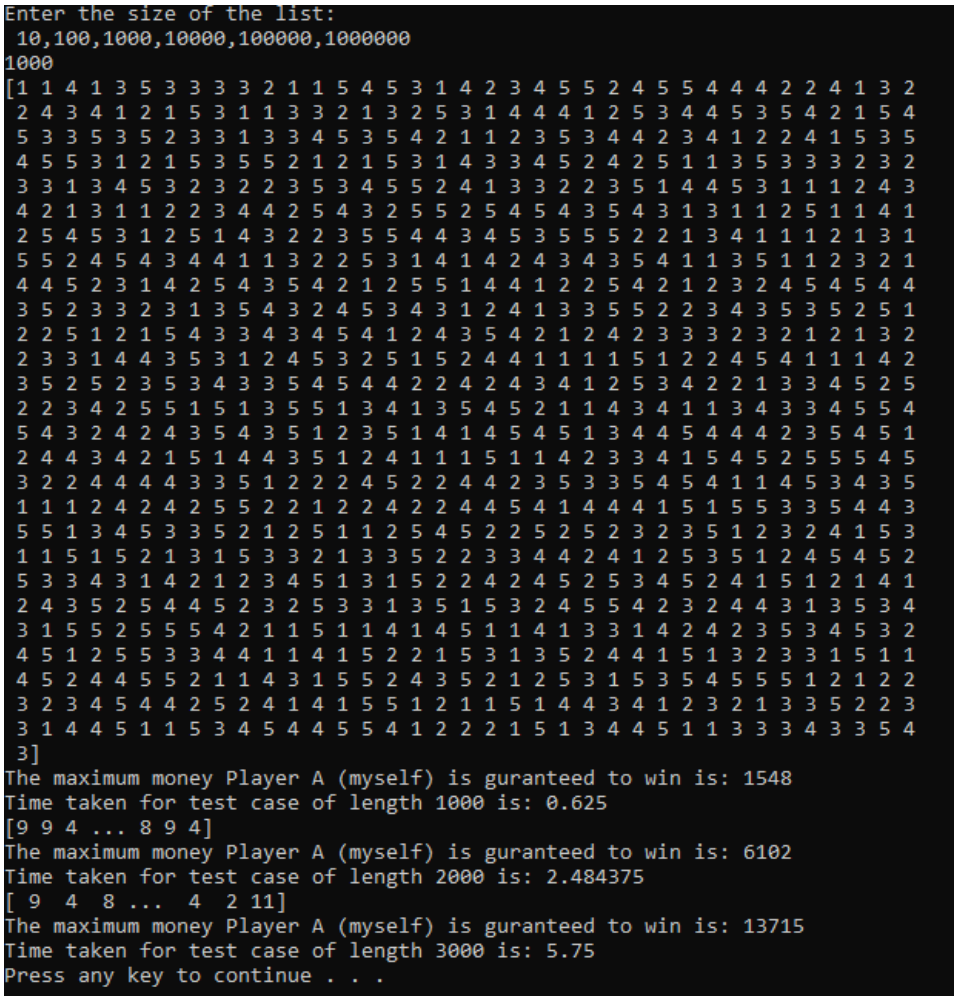

Fig 9

n=10000 & 100000:

The algorithm is taking large time to run for test case of 10000 & 100000, as the time complexity would be in this case 10000^2 & 100000^2 respectively.

2.a. Check each algorithm returns the same answer and discuss why greedy algorithm returns a different answer than others.

n=10

Brute

```
C:\WINDOWS\system32\cmd.exe
Enter the size of the list:
 10,100,1000,10000,100000
10
[1 1 4 1 3 5 3 3 3 3]
The maximum money Player A (myself) is guranteed to win is: 16
Time taken for test case of length 10 is: 0.0
[ 9  9  4  8  8  1 11  5  3  6  3  3  3  2  1  9  5 11  1 10]
The maximum money Player A (myself) is guranteed to win is: 64
Time taken for test case of length 20 is: 0.1875
[ 9  4  8 16 17 11  3  3  3 15  3 17 16  5 12 17 10  3 13  5  2 14  5  5
  4  8 16  2 15  8]
The maximum money Player A (myself) is guranteed to win is: 138
Time taken for test case of length 30 is: 421.96875
```

Greedy

```
Enter the size of the list:
 10,100,1000,10000,100000,1000000
10
[1 1 4 1 3 5 3 3 3 3]

Player A's max score: 16

Player B's max score: 11
None
Time taken for test case of length 10 is: 0.0
[ 9  9  4  8  8  1 11  5  3  6  3  3  3  2  1  9  5 11  1 10]

Player A's max score: 80

Player B's max score: 59
None
Time taken for test case of length 20 is: 0.0
[ 9  4  8 16 17 11  3  3  3 15  3 17 16  5 12 17 10  3 13  5  2 14  5
  4  8 16  2 15  8]

Player A's max score: 216

Player B's max score: 192
None
Time taken for test case of length 30 is: 0.0
Press any key to continue . . .
```

Dynamic

```
C:\WINDOWS\system32\cmd.exe
Enter the size of the list:
 10,100,1000,10000,100000,1000000
10
[1 1 4 1 3 5 3 3 3 3]
The maximum money Player A (myself) is guranteed to win is: 16
Time taken for test case of length 10 is: 0.0
[ 9  9  4  8  8  1 11  5  3  6  3  3  3  2  1  9  5 11  1 10]
The maximum money Player A (myself) is guranteed to win is: 64
Time taken for test case of length 20 is: 0.0
[ 9  4  8 16 17 11  3  3  3 15  3 17 16  5 12 17 10  3 13  5  2 14  5  5
  4  8 16  2 15  8]
The maximum money Player A (myself) is guranteed to win is: 138
Time taken for test case of length 30 is: 0.0
Press any key to continue . . .
```

Table 1

As we can see, Brute Force and Dynamic approach returns the same value as the output amount for Player A for test case of size n=10, however, Greedy approach returns a different value as the output amount that Player A wins for test case of size n=10. Since the Brute Force can only be run on a small test case of size 10, so just compared the value with Dynamic approach and it turns out to be same.

The greedy approach does not take care of anything else other than the fact that it just looks at current value and ignore the value which could be available in next move. So, it does not always produce optimal solutions whereas in other two approaches it is also taken into consideration that the opponent is also playing optimally. Brute Force and Dynamic both make sure that Player A wins with the maximum possible value when Player B is playing optimally.

To show, when greedy approach does not produce an optimal solution and makes Player B win the game, here is a snapshot of it:

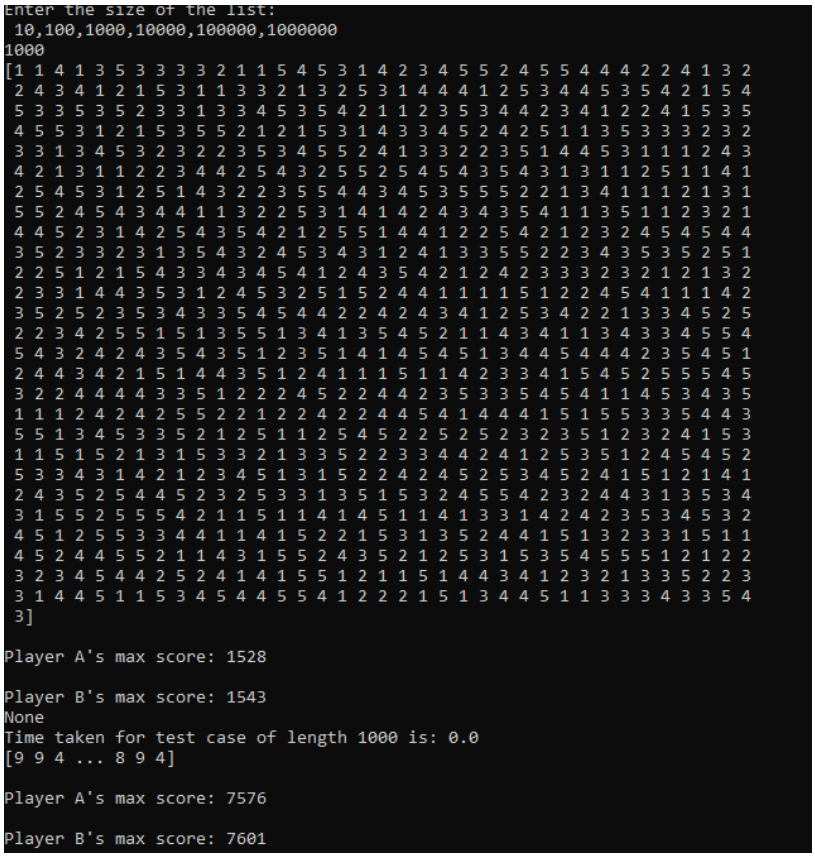

Fig 10.1

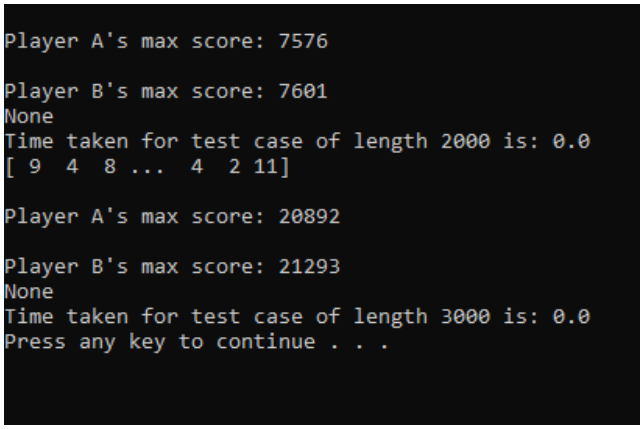

Fig 10.2

The score of Player B (amount won by Player B) is more than Player A. Hence, we can not rely on greedy approach to produce optimal solution and win our game.

2.b. Time each of your algorithm on each test case and report their runtimes.

Brute Force (O(2^n))	Greedy (O(n))	Dynamic (O(n^2))
For n=10: Test case of length 10: 0.0 Test case of length 20:0.1875 Test case of length 30:421.96875 *(refer fig 1.)*	For n: 10, 100,1000: 0.0 *(refer fig 2, fig 3, fig 4.1 & 4.2)*	For n=10: 0.0 *(refer fig 7.)*
For other test cases, 100,1000,10000,100000: The algorithm takes huge amount of time to run as the time complexity is exponential.	For n= 10000: Test case of length 10000: 0.015 Test case of length 20000: 0.015 Test case of length 30000: 0.015 *(refer fig 5)*	For n=100: Test case of length 100: 0.015 Test case of length 200: 0.031 Test case of length 300: 0.06 *(refer fig 8)*
	For n= 100000: Test case of length 100000: 0.078 Test case of length 200000: 0.125 Test case of length 300000: 0.1875 *(refer fig 6)*	For n=1000: Test case of length 1000: 0.625 Test case of length 2000: 2.484 Test case of length 3000: 5.75 *(refer fig 9)* For n=10000 & 100000, the algorithm takes huge time to run as the time complexity is n^2, so in this case it would be 10000^2 & 100000^2, which is quite high.

Table 2

2.c. Plot with size of test cases on x-axis and time on y-axis.

Brute Force:

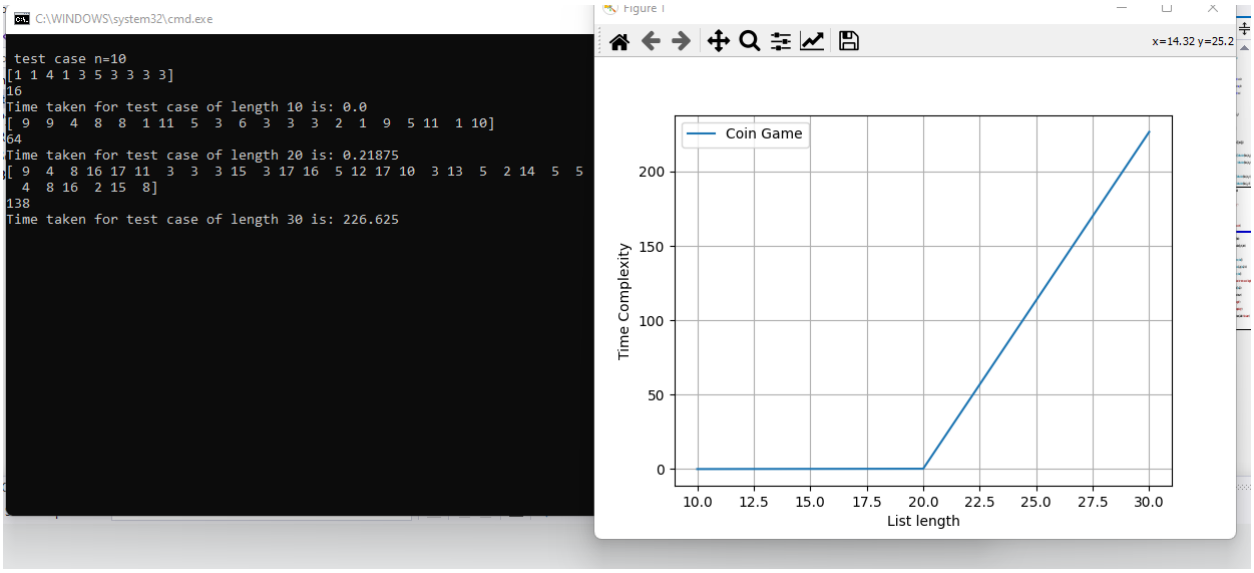

Fig 11: Brute Force: Test Case Plot (n=10)

Greedy Algorithm:

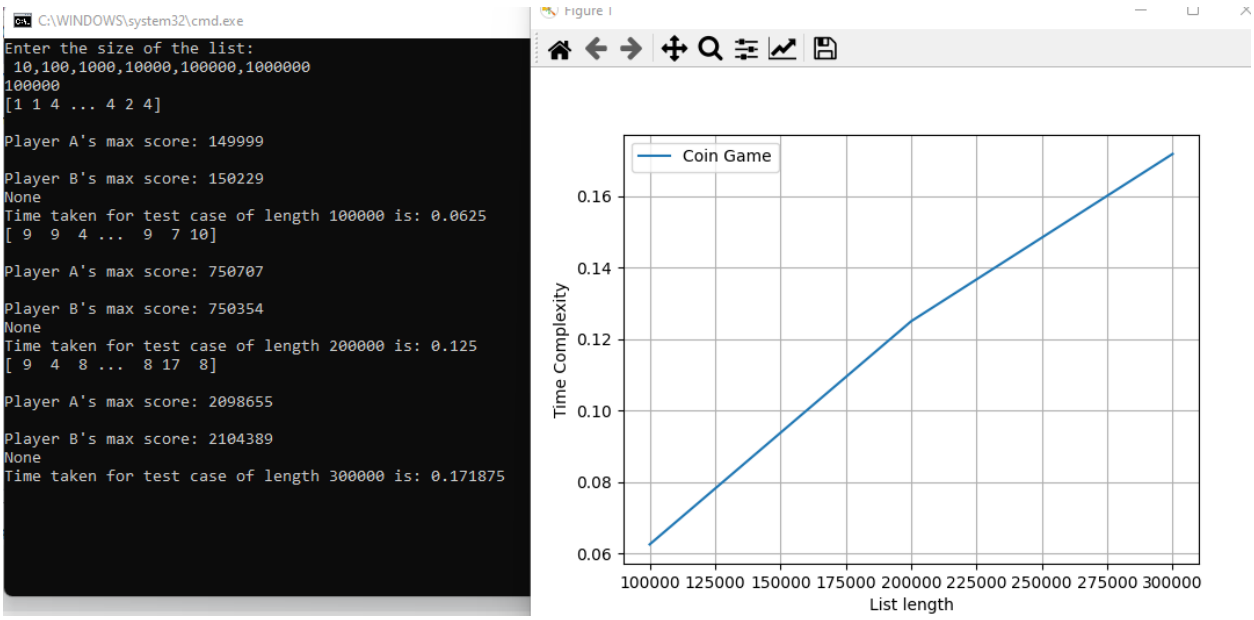

Fig 12: Greedy: Test Case Plot (n=100000)

Dynamic Program:

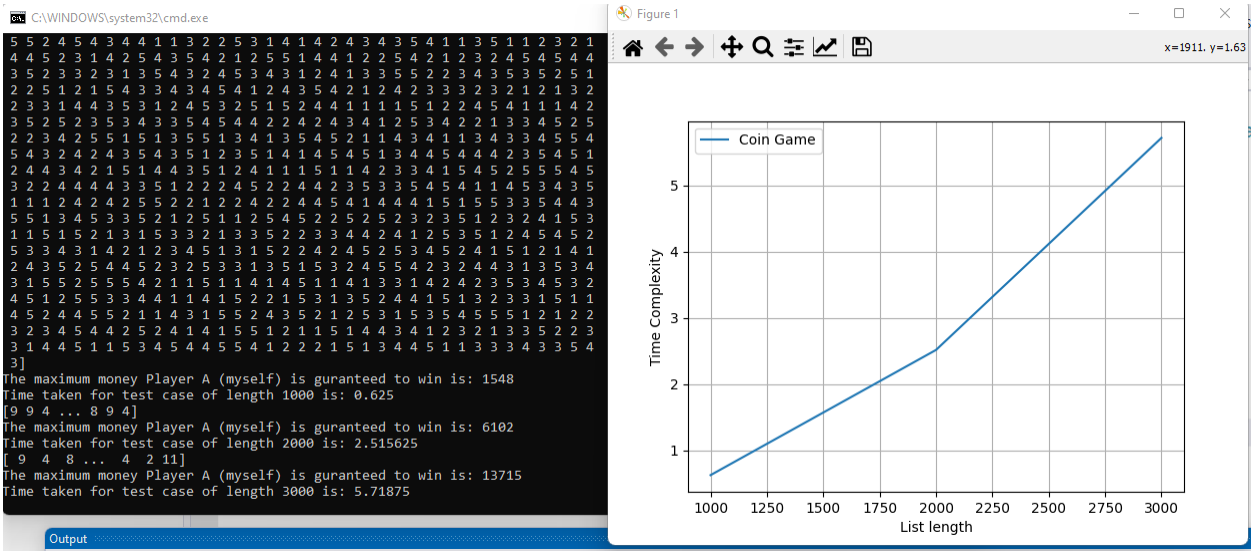

Fig 13: Dynamic: Test Case Plot (n=1000)

3. Which algorithm you would recommend and why.

I would say, Dynamic Programming as it is more efficient and produces optimal solution. Although greedy approach does not take much time to run for all the test cases, but it does not produce optimal solutions all the time as we have seen, and it does not consider that the opponent is also paying optimally which sometimes led to losing of the game for the user. Dynamic Programming solve all the sub-problems and then select one that leads to an optimal solution. These subproblems in dynamic program are memoized rather than computed again and again and that reduces the time complexity.