1140. Stone Game II

Alice and Bob continue their games with piles of stones. There are a number of piles **arranged in a row**, and each pile has a positive integer number of stones `piles[i]`. The objective of the game is to end with the most stones.

Alice and Bob take turns, with Alice starting first. Initially, `M = 1`.

On each player's turn, that player can take **all the stones** in the **first** `X` remaining piles, where `1 <= X <= 2M`. Then, we set `M = max(M, X)`.

The game continues until all the stones have been taken.

Assuming Alice and Bob play optimally, return the maximum number of stones Alice can get.

Example 1:

```
Input: piles = [2,7,9,4,4]
Output: 10
Explanation:  If Alice takes one pile at the beginning, Bob takes two piles, then Alice takes 2 piles again. Alice can get 2 + 4 + 4 = 10 piles in total. If Alice takes two piles at the beginning, then Bob can take all three piles left. In this case, Alice get 2 + 7 = 9 piles in total. So we return 10 since it's larger.
```

Example 2:

```
Input: piles = [1,2,3,4,5,100]
Output: 104
```

Constraints:

- `1 <= piles.length <= 100`
- `1 <= piles[i] <= 10`4

1140. Stone Game II

```
/*
   DP: 3D Array Memoization
   Time complexity: O(n*M^2)
   Space complexity: O(n^3+n*M^2)

   Without memoization
   Time complexity: O(n^M)
   Space complexity: O(n^M)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<vvi> vvvi;
class Solution {
   public:
      int stoneGameII(std::vector<int>& piles) {
         int n=piles.size();
         vvvi memo(2,vvi(n+1,vi(n+1,-1)));
         auto solve=[&](bool is_alice,int i,int M,auto& self)->int{
            if(i>=n) return 0;

            if(memo[is_alice][i][M]!=-1) return memo[is_alice][i][M];

            int ans=(is_alice)?0:INT_MAX;

            int prefix_sum=0;
            for(int X=1;X<=std::min(2*M,n-i);++X){
               //if(i+X>n) break;
               prefix_sum+=piles[i+X-1];
               ans=is_alice
                  ?std::max(ans,prefix_sum+self(!is_alice,i+X,std::max(M,X),self))
                  :std::min(ans,self(!is_alice,i+X,std::max(M,X),self));
            }
            return memo[is_alice][i][M]=ans;
         };

         return solve(true,0,1,solve);
      }
};
```

1140. Stone Game II

```
/*
   DP: 2D array Memoization
   Time complexity: O(n*M^2)
   Space complexity: O(n^2+n.M^2)

   Without memoization
   Time complexity: O(n^M)
   Space complexity: O(n^M)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<vvi> vvvi;
class Solution {
   public:
      int stoneGameII(std::vector<int>& piles) {
         int n=piles.size();
         vvi memo(n+1,vi(n+1,-1));

         auto solve=[&](int i,int M,auto& self)->int{
            if(i>=n) return 0;

            if(memo[i][M]!=-1) return memo[i][M];

            int ans=INT_MIN;

            int prefix_sum=0;
            for(int X=1;X<=std::min(2*M,n-i);++X){
               //if(i+X>n) break;
               prefix_sum+=piles[i+X-1];
               ans=std::max(ans,prefix_sum-self(i+X,std::max(X,M),self));
            }
            return memo[i][M]=ans;
         };


         int sum=accumulate(piles.begin(), piles.end(), 0);
         return (sum+solve(0,1,solve))/2;
      }
};
```

1140. Stone Game II

```
/*
   DP: Tabulation
   Time complexity: O(n*M^2)
   Space complexity: O(n^2)

*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<vvi> vvvi;
class Solution {
   public:
      int stoneGameII(std::vector<int>& piles) {
      int n = piles.size();

      vvi dp(n+1,vi(n+1,0));

      vi suffix_sum(n+1,0);

      for (int i=n-1;i>=0;--i) {
         suffix_sum[i]=suffix_sum[i+1]+piles[i];
      }

      for (int i=n-1;i>=0;--i) {
         for (int M=1;M<=n;++M) {
            for (int X=1;X<=std::min(2*M,n-i);++X) {
               dp[i][M] =std::max(dp[i][M],suffix_sum[i]-dp[i+X][std::max(M,X)]);
            }
         }
      }

      return dp[0][1];
      }
};
```