Eww Cards

- There are **N** cards having an Integer number which is visible on both the sides of the card.
  -

  Game:

  -

  1.Two players are playing with cards

  2.In each turn the player have an option to choose the first card or last card.

  3.The player after choosing the card should remove card from the deck and add the number to his/her score

  4.After the removal of last card the one who is having more score will win the game.

  5.If both players have same score its a draw.

  -
- Cards are represented as an array of elements[C_1,C_2,C_3,C_4,C_5,...C_n]
- Player1 will start the game
- If player1 wins the game output "**PLAYER1**"
- If player2 wins the game output "**PLAYER2**"
- If it is a draw output "**DRAW**"
- **Note** :Every player plays Optimal

Input Format

- First line of input contains **T** no of testcases.
- First line of each testcase contains Integer **N** no of cards.
- Second line of each testcase contains **N** separate spaced integers

Constraints

- $\mathbf{0 < T < 10^3}$

- **$0 < N <= 20$**
- **$0 < A[i] <= 10^6$**

Output Format

- For each testcase
- If player1 wins the game output "**PLAYER1**"
- If player2 wins the game output "**PLAYER2**"
- If it is a draw output "**DRAW**"

Sample Input 0

```
1410 100 9 1
```

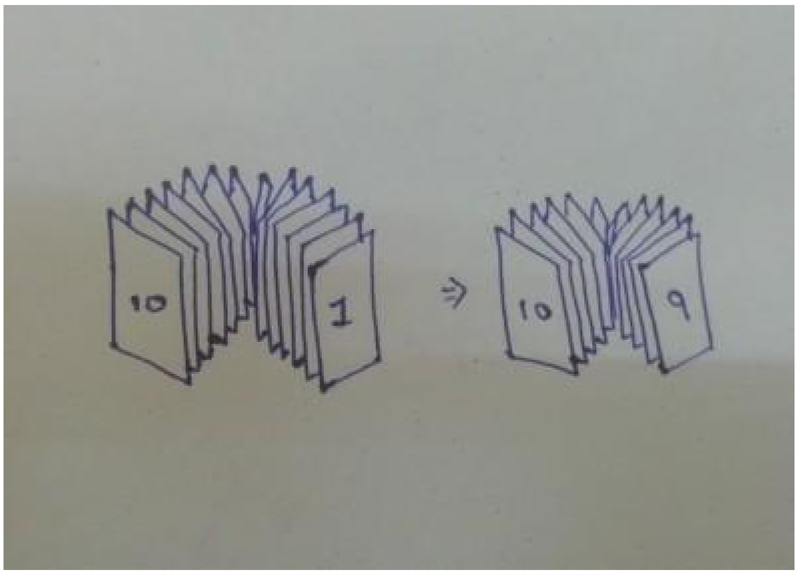
Sample Output 0

```
PLAYER1
```

Explanation 0

Explanation:

```
  Cards=[10,100,9,1]

  As player1 will start the game he may choose 10 or 1.

  if Player1 chooses the last element his score=1 Cards=[10,100,9]



  Player2 either he can choose 10 or 9 his score becomes 10 or 9 the player1 will
choose 100 win the game

  Output "PLAYER1" player1 wins the game
```



Sample Input 1

```
22 10 1031 10 3
```

Sample Output 1

```
DRAWPLAYER2
```

Solution:

The idea behind the recursive approach is simple. The two players Player 1 and Player 2 will be taking turns alternately. For the Player 1 to be the winner, we need score_{Player1} &geq; score_{Player2}. Or in other terms, score_{Player1} - score_{Player2} &geq; 0.

Thus, for the turn of Player 1, we can add its score obtained to the total score and for Player 2's turn, we can substract its score from the total score. At the end, we can check if the total score is greater than or equal to zero(equal score of both players), to predict that Player 1 will be the winner.

Thus, by making use of a recursive function *winner(nums,s,e,turn)* which predicts the winner for the numsnums*nums* array as the score array with the elements in the range of indices [s,e][s,e][*s*,*e*] currently being considered, given a particular player's turn, indicated by turn=1turn=1*turn*=1 being Player 1's turn and turn=−1turn=-1*turn*=−1 being the Player 2's turn, we can predict the winner of the given problem by making the function call `winner(nums,0,n-1,1)`. Here, n refers to the length of $nums$ array.

In every turn, we can either pick up the first($nums[s]$) or the last($nums[e]$) element of the current subarray. Since both the players are assumed to be playing smartly and making the best move at every step, both will tend to maximize their scores. Thus, we can make use of the same function `winner` to determine the maximum score possible for any of the players.

Now, at every step of the recursive process, we determine the maximum score possible for the current player. It will be the maximum one possible out of the scores obtained by picking the first or the last element of the current subarray.

To obtain the score possible from the remaining subarray, we can again make use of the same `winner` function and add the score corresponding to the point picked in the current function call. But, we need to take care of whether to add or subtract this score to the total score available. If it is Player 1's turn, we add the current number's score to the total score, otherwise, we need to subtract the same.

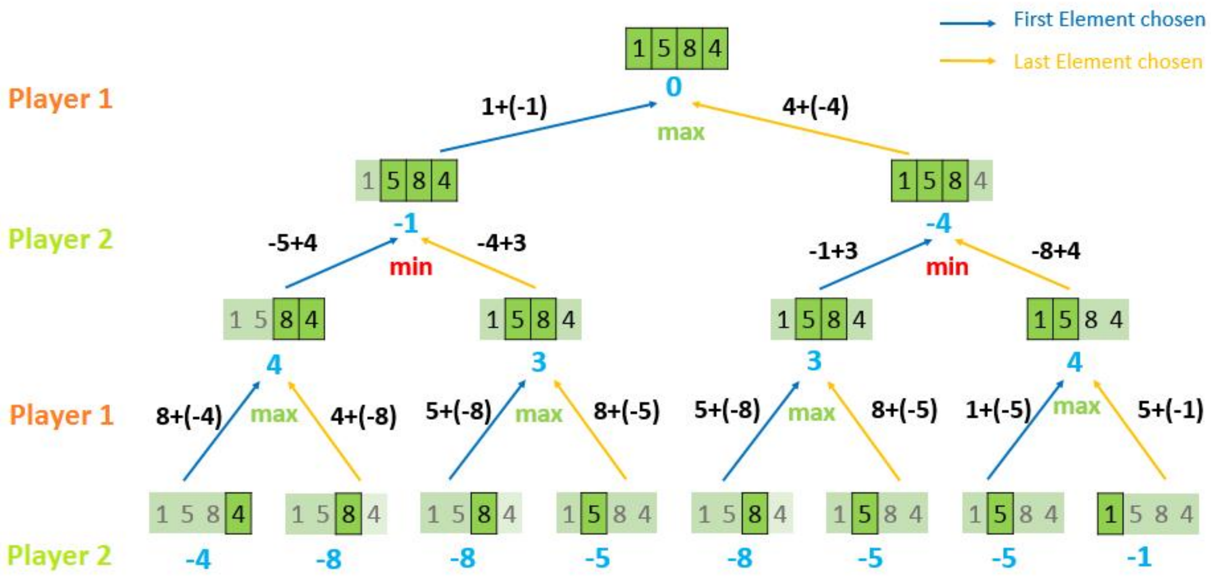
Thus, at every step, we need update the search space appropriately based on the element chosen and also invert the $turn$'s value to indicate the turn change among the players and either add or subtract the current player's score from the total score available to determine the end result.

Further, note that the value returned at every step is given by $turn *\text{max}(turn * a, turn * b)$. This is equivalent to the statement $max(a,b)$ for Player 1's turn and $min(a,b)$ for Player 2's turn.

This is done because, looking from Player 1's perspective, for any move made by Player 1, it tends to leave the remaining subarray in a situation which minimizes the best score possible for Player 2, even if it plays in the best possible manner. But, when the turn passes to Player 1 again, for Player 1 to win, the remaining subarray should be left in a state such that the score obtained from this subarrray is maximum(for Player 1).

This is a general criteria for any arbitrary two player game and is commonly known as the Min-Max algorithm.

The following image shows how the scores are passed to determine the end result for a simple example.



Solution in C:

```
#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int max(int a,int b){

   if(a>b){

      return a;

   }

   return b;

}

int min(int a,int b){

   if(a<b){

      return a;
```

```
  }

  return b;

}

int solve(int *nums,int i,int j,int turn){

    if(i>j){

      return 0;

    }

    if(turn==1){

      return  max(nums[i]+solve(nums,i+1,j,0),nums[j]+solve(nums,i,j-1,0));

    }

    return min(solve(nums,i+1,j,1)-nums[i],solve(nums,i,j-1,1)-nums[j]);



  }



int main() {



  int testcases;

  scanf("%d",&testcases);

  while(testcases--){

     int n;

    scanf("%d",&n);

    int nums[n];

    for(int i=0;i<n;i++){

      scanf("%d",&nums[i]);

    }
```

```
      int val=solve(nums,0,n-1,1);

     if(val>0){

        printf("PLAYER1\n");

     }else if(val<0){

        printf("PLAYER2\n");

     }else{

        printf("DRAW\n");

     }

  }

  return 0;

}
```

Solution in Java:

```
        import java.io.*;

        import java.util.*;



        public class Solution {



           public static void main(String[] args) {

              Scanner scan= new Scanner(System.in);

              int testcases=scan.nextInt();

              while(testcases-- >0){

                 int n=scan.nextInt();

                 int[] nums= new int[n];

                 for(int i=0;i<n;i++){
```

```java
                nums[i]=scan.nextInt();

            }

            int val=solve(nums,0,nums.length-1,1);

            if(val>0){

                System.out.println("PLAYER1");

            }else if(val<0){

                System.out.println("PLAYER2");

            }else{

                System.out.println("DRAW");

            }

        }

    }

    public static int  solve(int[] nums,int i,int j,int turn){

        if(i>j){

            return 0;

        }

        if(turn==1){

            return  Math.max(nums[i]+solve(nums,i+1,j,0),nums[j]+solve(nums,i,j-1,0));

        }

        return Math.min(solve(nums,i+1,j,1)-nums[i],solve(nums,i,j-1,1)-nums[j]);



    }

}
```