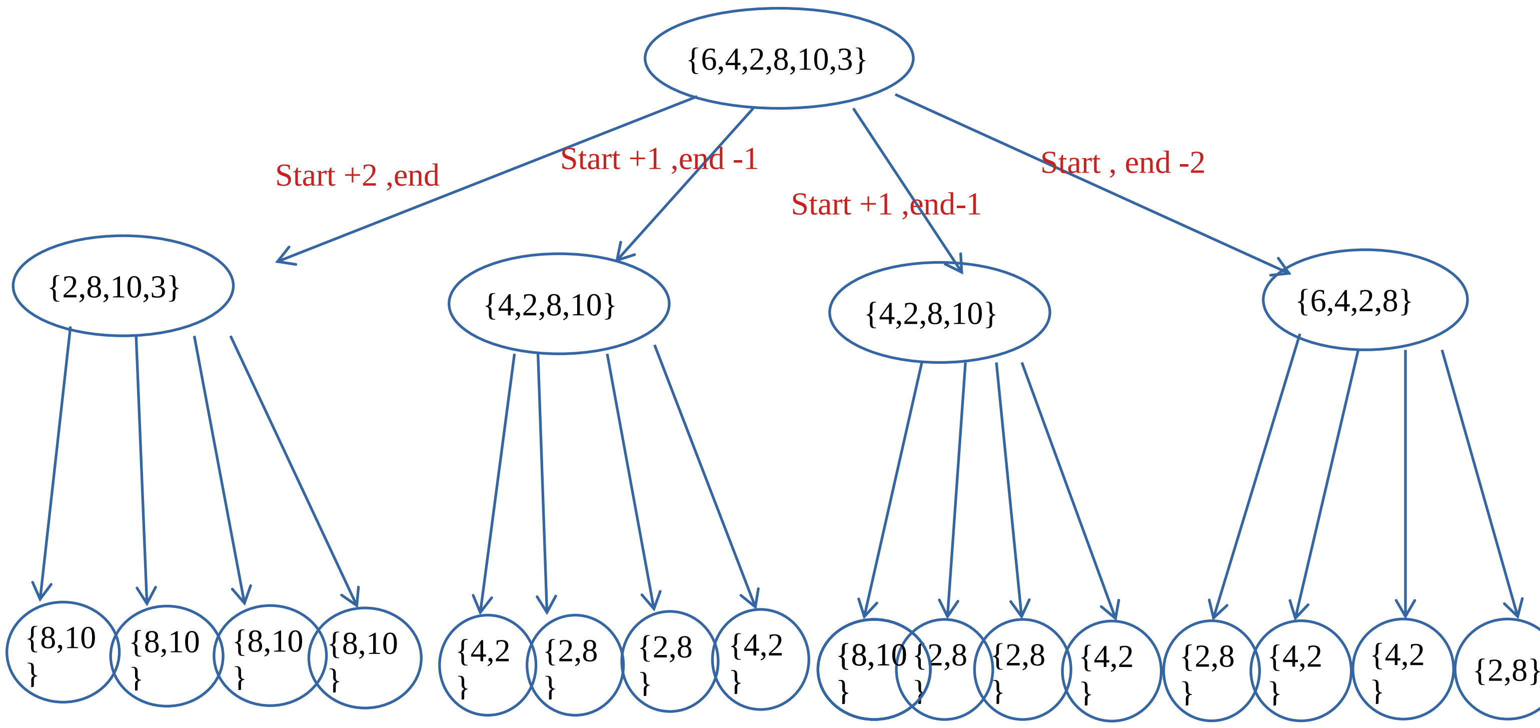
Final project – Dynamic Programming Cards War Game

part1 : Divide & Conquer

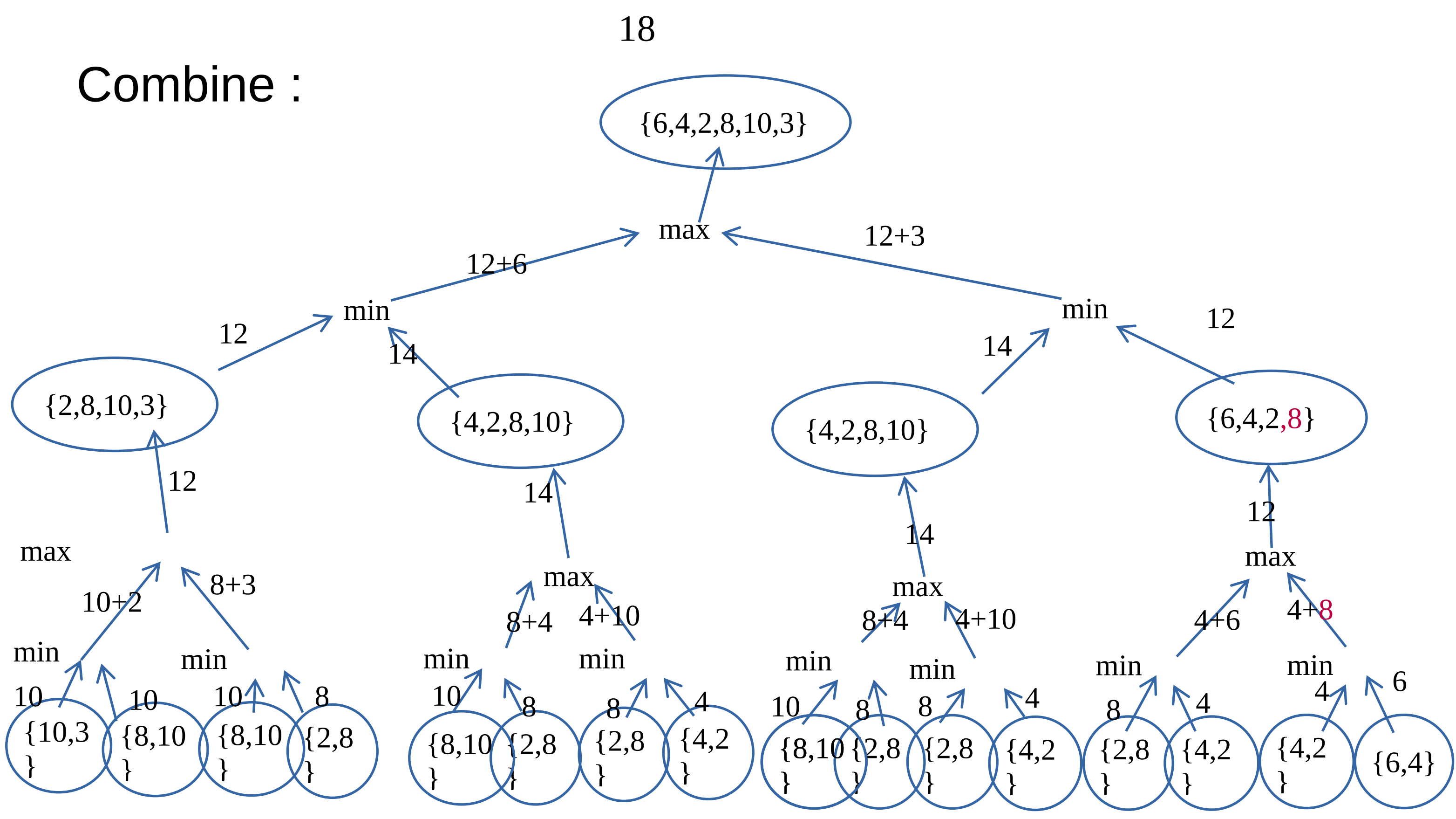
1- int f(int *A , int start , int end);
2- start : start of A ;
    end : end of A ;

Divide :



Anchor (start +1 =end)

Combine :



//Anchor :
if(start+1==end)
    return max(A[start] ,A[end])

Recursive code :

```
int f(int *A, int start , int end ) {

    if (end == start + 1)

 Return  max(A[start], A[end]);

return max(A[start]+min(f(A, start + 2, end), f(A, start + 1, end
- 1)), A[end] + min(f(A, start + 1, end - 1),f(A, start, end - 2)));}
```

part 2 :

Dynamic Programming :

Index in A	0	1	2	3	4	5
value	6	4	2	8	10	3

table_DP

End / start	0	1	2	3	4	5
0	[6,0]	[6,4]	[8,4]	[12,8]	[18,12]	[18,15]
1	--	[4,0]	[4,2]	[10,4]	[14,10]	[15,12]
2	--	--	[2,0]	[8,2]	[12,8]	[12,11]
3	--	--	--	[8,0]	[10,8]	[11,10]
4	--	--	--	--	[10,0]	[10,3]
5	--	--	--	--	--	[3,0]

each cell represasnt by struct of value for player1 & value for player2 (me ,player2)

to find the value for me in each cell aply this equathon

table_DP[start][end].me=max(A[start]+table_DP[start+1][end].player2 , A[end]+table_DP[start][end-1].player2);

if(A[start]+table_DP[start+1][end].player2) is max
table_DP[start][end].player2= table_DP[start][end].me;
if(A[end]+table_DP[start][end-1].player2) is max
A[start]+table_DP[start][end-1].player2
code to fill table :

```
int card_war_dp(int* input_data, int size_data)
{
    struct cell table_DP[size_data][size_data];
    for(int i=0;i<size_data;i++){
        table_DP[i][i].me=input_data[i];
        table_DP[i][i].player2=0;
    }
    int *arr1=new int[size_data/2];
    int *arr2=new int[size_data/2];

    int pos1;
    int pos2;

    int offset=1;
    while (offset <= size_data) {
        for (int i = 0; i < size_data; i++) {
            for (int j = i; j < size_data; j++) {
                if (j == i + offset)
                {

                    int F=input_data[i];
                    int B=input_data[j];
                    table_DP[i][j].me=max(B+table_DP[i][j-1].player2 ,
            F+table_DP[i+1][j].player2);
                    if(B+table_DP[i][j-1].player2>=F+table_DP[i+1][j].player2)
                    {
                        table_DP[i][j].player2=table_DP[i][j-1].me;
                    }
                    else
                    {
                        table_DP[i][j].player2=table_DP[i+1][j].me;
                    }
                }

            }
}
        offset++;
```

```
   }

return table_DP[0][size_data - 1].me;

}
```

code print the sequence of moves :

```
 int  c=0;
   int p = size_data-1;
   bool toggele= true;
   for (int n = size_data - 1; n >= 0; n--)
   {
      if(table_DP[c][p].player2 == table_DP[c+1][p].me){
         if(toggele){
            cout << "F" ;//<< table_DP[c][p].me-table_DP[c+1][p].player2 <<",";
         }
         else{
            cout << "f" ;//<< table_DP[c][p].me-table_DP[c+1][p].player2<<",";
         }
         c++;
      }
      else{
         if(toggele){
            cout << "B"; //<< table_DP[c][p].me-table_DP[c][p-1].player2<<",";
         }
         else{
            cout << "b" ;//<< //table_DP[c][p].me-table_DP[c][p-1].player2<<",";
         }
         p--;
      }

      toggele=!toggele;
   }
```