Basics of Bits

__builtin_popcount(n) -> counts no. of set bits (1's) of n.

n -> check if the xth bit is on or not?

n=5 x=2
101
100
00000000001
00000000100
1<<2

1<<0 =1
if(n&(1<<x))

2) Set the xth bit of a given number.
n=7 x=1
111 -> 111
010
n|(1<<x)

3) Toggle (flip) the xth bit of a given number.

n=7 x=2
111 -> 011
100
n^(1LL<<x)
=7^(4)=3

4) LSB -> Least significant bit
n=10 -> 1010 -> 1th bit 2^1
n=7 ->111 -> 2^0= 1
n=6 -> 110 -> 2^1=2

O(logN)

O(1) -> LSB(n)=n&(-n)

n=2 -> 0000000000010

```
~n     1111111111101 (1's complement)
       <----32bits->
       +           1
       _____________
-n=    1111111111110  (2's complement)
```

-n = ~n+1

5) Check if a given number is a power of 2 or not.
4=2^2
3
100
011
000

n&(n-1)-> 0 if n is a power of 2
OR
if(__builtin_popcount(n)==1) // O(1)

6) Unset xth bit of a number .
n&(~(1<<x))

7) 5 -> 101 -> 2^0+2^2
100
O(logN) -> O(count_of_set_bits)

```
while(n>0){
  cout<<LSB(n)<<" ";
  n-=LSB(n);
}
```

```
//atcoder o-matching

int dp[21][(1LL<<21)];
int a[21][21];
int n;

int add(int x,int y){
  return (x%mod+y%mod)%mod;
}

int f(int i,int mask){

  if(i==n)
    return 1LL;

  //i->no of man u r on
  //i->no of unset bits

  //memo
  if(dp[i][mask]!=-1LL)
    return dp[i][mask];

  int ans=0;

  for(int j=0;j<n;++j){
    //check if compatible
    if(!a[i][j])
      continue;

    int submask=(1LL<<j);

    if(mask&submask){

      ans=add(ans,f(i+1,mask^submask));

    }
  }

  return dp[i][mask]=ans;
}
```

```
inline void solve(){

  cin>>n;

  fr(i,n){
    fr(j,n){
      cin>>a[i][j];
    }
  }

}


//space optimization
int dp[(1LL<<21)];
int a[21][21];
int n;

int add(int x,int y){
  return (x%mod+y%mod)%mod;
}

int f(int mask){

  int man=n-setbits(mask);

  if(man==n)
    return 1LL;

  //i->no of man u r on
  //i->no of unset bits

  //memo
  if(dp[mask]!=-1LL)
    return dp[mask];
  int ans=0;

  for(int j=0;j<n;++j){
    //check if compatible
    if(!a[man][j])
      continue;

    int submask=(1LL<<j);

    if(mask&submask){

      ans=add(ans,f(mask^submask));

    }
  }

  return dp[mask]=ans;
}



inline void solve2(){


  for(int submask=mask;submask;submask=(submask-1)&mask){
        //to iterate over all the submasks
  }
}
```

EQUAL SUBSET PROBLEM

Qs) Divide the array into two non-empty parts such that the sum of elements in one part is equal to sum of elements in the second part.

Normal approach->

```
#include<bits/stdc++.h>
using namespace std;

int n;
vector<int> v;

int dp[10005][20];

bool check(int total,int pos){
    if(pos==n){
        if(total==0){
            return true;
        }
        return false;
    }
    if(dp[total][pos]!=-1){
        return dp[total][pos];
    }

    dp[total][pos] = check(total-v[pos],pos+1) || check(total,pos+1);
    return dp[total][pos];
}

//true, false
```

```
// 0001000110010000
// 0101000011010000
//.......

//total cases- 2^n cases

int main() {

  memset(dp,-1,sizeof(dp));
  cin>>n;
  v.resize(n);

  int total=0;

  for(int i=0;i<n;i++){
      cin>>v[i];
      total+=v[i];
  }
  if(total%2!=0){
      cout<<"No";
  }else{
      if(check(total/2,0)){
          cout<<"Yes";
      }else{
          cout<<"No";
      }
  }

}

//complexity-> sum * n
```

Bitmask approach->

```
#include<bits/stdc++.h>
using namespace std;
```

```
int n;
vector<int> v;

//total cases- 2^n cases

//0000101010011

bool check(int total){
    for(int i=0;i<pow(2,n);i++){
        int tempsum=0;
        for(int j=0;j<n;j++){
            if(((i>>j)&1)==1){
                tempsum+=v[j];
            }
        }
        if(tempsum==total){
            return true;
        }
    }
    return false;
}

int main() {

  cin>>n;
  v.resize(n);

  int total=0;

  for(int i=0;i<n;i++){
      cin>>v[i];
      total+=v[i];
  }
  if(total%2!=0){
      cout<<"No";
  }else{
   if(check(total/2)){
     cout<<"Yes";
   }else{
     cout<<"No";
   }
 }

}

//complexity->  n * 2^n
```