Knapsack - 01

minor changes in code

1) Subset sum
2) Equal sum partition
3) Count of Subset sum
4) Minimum Subset sum difference
5) Target sum
6) count of subset sum with given diff.

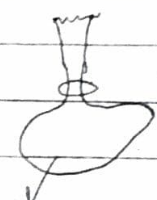
Knapsack Problem

- Fractional Knapsack
- 01 knapsack
- Unbounded knapsack (multiple occurence allow)

IP: wt[] = 1 3 4 5

val[] = 1 4 5 7

W = 7 (capacity)



Bag

P1	P2	P3	P4
W1	W2	W3	W4

W

knapsack (max profit)

* 0-1 Knapsack Recursive -

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

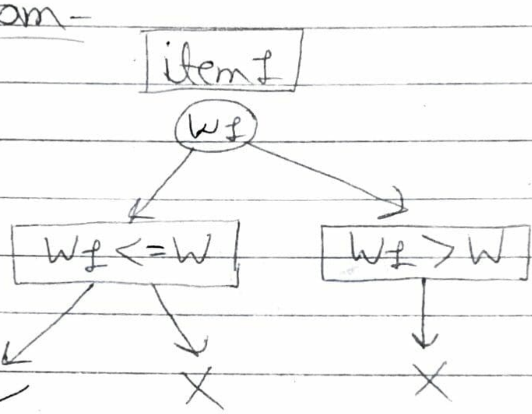
IP: wt[] = | 2 | 3 | 4 | 5 |

val[] = | 1 | 4 | 5 | 7 |

W = 7

item → include
item → not include

Choice diagram -



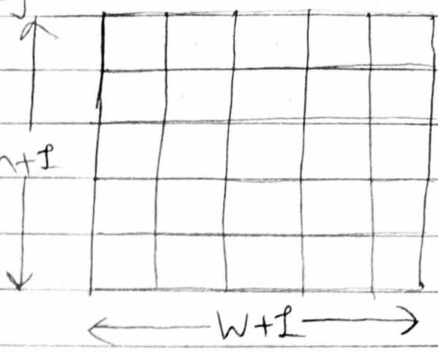
Base condition - Think of the smallest valid input

```
if (n==0 || W==0)
    return 0;
```

```
int knapsack (int wt[], int val[], int W,
                                  int n)
{
    if (n==0 || W==0)
        return 0;

    if (wt[n-1]<=W )
    return max(val [n-1]+ knapsack (wt,
              val , W-wt[n-1] , n-1 ),
                  Knapsack (wt, val, W, n-1))

    elif (wt[n-1]>W )
     return knapsack (wt, val, W, n-1);

}
```

Memoization-

```
int  t[n+1][W+1]
```



initialize all values of matrix to -1

```
int knapsack (int wt[], int val[], int W, int n)
{
    if ( t[n][W] != -1 )
        return t[n][W];

    if (wt[n-1] <= W)
    t[n][W] = max(val[n-1] + knapsack (wt, val,
        W-wt[n-1], n-1), knapsack(wt, val, W, n-1));

    elif (wt[n-1] > W)
    t[n][W] = knapsack (wt, val, W, n-1);

}
```

* 0/1 Knapsack Top-down approach -

n = 5
W = 4

t[6][5]

t[n+1][W+1]

	0	1	2	3	4
0	0	0	0	0	0
1	0				
2	0				
3	0				
4	0				
5	0				

n+1 (rows), W+1 (columns), t[n][W]

I/P
wt[] = [1, 3, 4, 5, 6]
val[] = [1, 4, 5, 7, 8]
W = 4

⇒ Problem for t[6][5]

Base Condition ⇌ Initialization.

```
if (n==0 || W==0)
    return 0;
```

→

```
for(int i=0; i<n+1; i++)
for(int j=0; j<W+1; j++)
    if (i==0 || j==0)
        t[i][j] = 0;
```

```
if(wt[n-1] <= W)
return max(val[n-1] +
     knapsack(wt, val, W-
     wt[n-1], n-1),
    knapsack(wt, val, W, n-1))
```

→

```
for (int i=0; i<n+1; i++)
for (int j=0; j<W+1; j++ ){
  if (wt[i-1]<=j)
  t[i][j] = max(          )
  else
  t[i][j] = t[i-1][j]
}
```

Program

```
int main()
{
  int wt[] = {1,3,4,5};
  int val[] = {1,4,5,7};
  int W=7, n=4;
  int t[n+1][W+1];
  for (int i=0; i<n+1; i++)
    for (int j=0; j<W+1; j++)
       if (i==0 || j==0)
          t[i][j] = 0;
  for (int i=1; i<n+1; i++)
    for (int j=1; j<W+1; j++) {
      if (wt[i-1]<=j)
         t[i][j] = max (val[i-1]+ t[i-1]
              [j-wt[i-1]], t[i-1][j]);
      else
         t[i][j] = t[i-1][j]; }
  printf("Result : %d", t[n][W]);
  return 0;
}
```

* Subset sum Problem –

arr[] : | 2 | 3 | 7 | 8 | 10 |

Sum = 11

Is there any subset which sum is equal to 11? True/False.

STEPS

1- Problem stmt
2- Similiarity
3- Code variation → initaliz. , Code

t[n+1] [Sum+1]
t [6] [12]

→ j (sum)

i ↓ (n)

	0	1	2	3	4							
0	T	F	F	F	F	F	F	F	F	F	F	F
1	T											
2	T											
3	T											
4	T											
5	T											

o/p → True/False

t[0][0] → arr[] : { }
Sum : 0
True

t[1][0] → arr[] : 2
Sum : 0
True

t [0][1] → arr[] : { }
Sum : 1
False

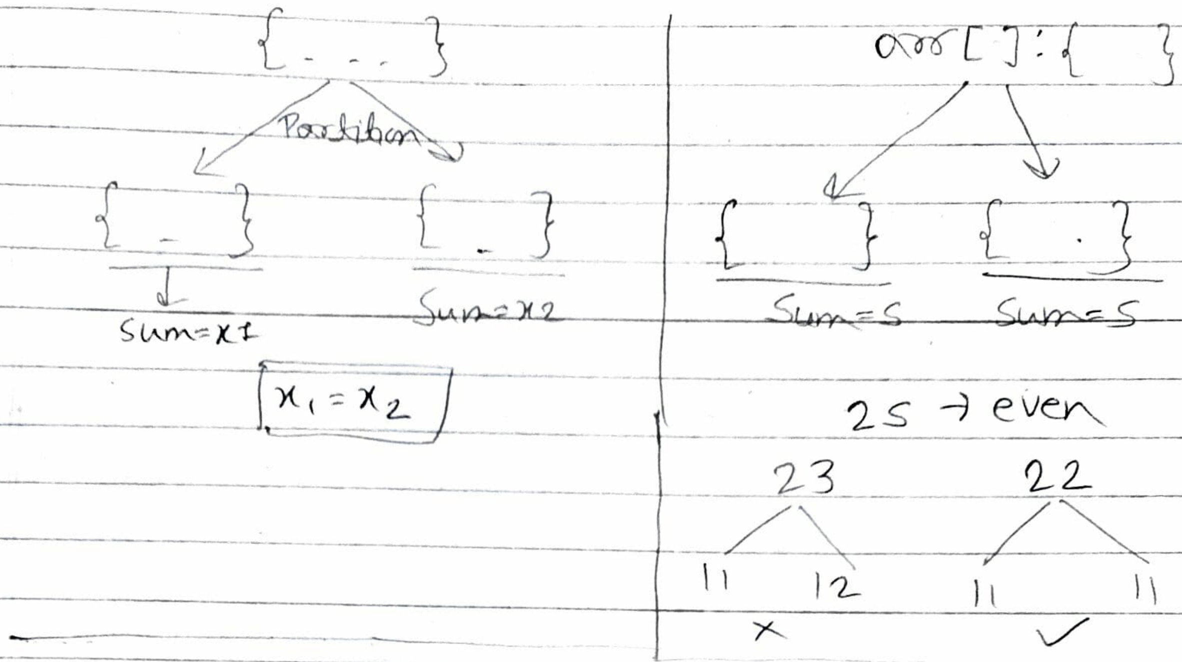
t [0] [2] → arr[] : { }
Sum : 2
False

Program

```
int main()
{
   int arr[] = {2,3,7,8,10}
   int sum = 11;
   int n=5;
   int t[n+1][sum+1];
   for(int i=0; i<n+1; i++)
    { for (int j=0; j<sum+1; j++)
       { if(i==0)
         t[i][j] = false;
         if(j==0)
         t[i][j] = true;
       }
    }
   for(int i=1; i<n+1; i++)
    { for(int j=1; j<sum+1; j++)
       { if (arr[i-1] <= j)
          t[i][j] = t[i-1][j-arr[i-1]] ||
                                t[i-1][j];
         else
          t[i][j] = t[i-1][j];
       }
    }
  printf("Result : %d", t[n][sum]);
  return 0;
}
```

* Equal Sum Partition -

arr[]: {1, 5, 11, 5}
O/p: T/F




```
if (sum(arr) % 2 != 0)
    return false;
else
    return subsetsum(arr, sum(arr)/2);
```

Base condition is same as Subset Sum Problem.
If there is the subset in the array which sum is equal to sum(arr)/2 Then we can divide the array into two equal sum partition.

* Count of Subset ~~Sum~~ To A Given Sum -

IP: arr[] = 2, 3, 5, 6, 8, 10   n=6
sum: 10

Count no of subset that can be form whose sum is equal to the Given Sum?

So, for sum: 10
there are 3 subsets ({2, 8}, {5, 2, 3}, {10}) can be form.
If there are no subset can be form whose sum equal to the given sum then output '0'.

t [n+1] [sum+1]
t [7] [11]

sum → j

	0	1	2	3	4 - - -
0	1	0	0	0	0 - - - - - - -
1	1				
2	1				
:	1				
	:				

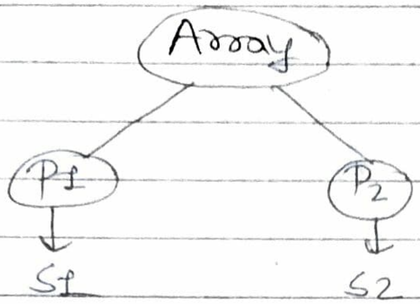
size of array ↓ i

```
int main()
{ int arr[10] = {2, 3, 5, 6, 8, 10 }, n=6;
  int sum=10, t[n+1][sum+1];
  for (int i=0; i<n+1; i++)
  { for (int j=0; j<sum+1; j++)
     { if(i==0)
          t[i][j]=0;
       if(j==0)
          t[i][j]=1;
  } }
  for (int i=1; i<n+1; i++)
   { for (int j=1; j<sum+1; j++)
      { if(arr[i-1]<=j)
     t[i][j] = t[i-1][j-arr[i-1]] +
                         t[i-1][j];
         else
     t[i][j] = t[i-1][j];
        }
    }
  printf("Result : %d", t[n][sum]);
  }
```

* Minimum Subset Sum Difference -

arr[ ] : | 1 | 6 | 11 | 5 |

o/p : 1



P1 : Partition 1

P2 : Partition 2

S1 : sum 1

S2 : sum 2

$\boxed{|S_1 - S_2| = \min}$

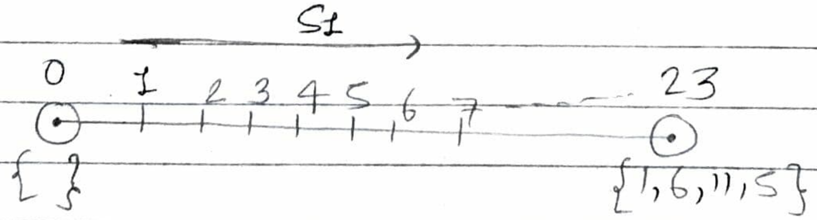
$\underline{\{1, 6, 5\}}$ 12 , $\underline{\{11\}}$ 11

12 - 11 = 1 (min)

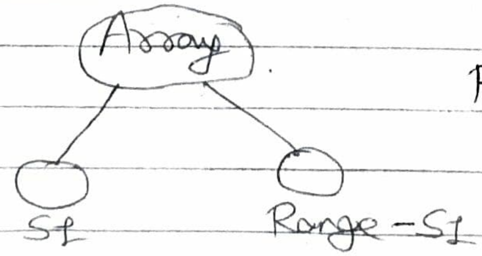
arr : | 1 | 6 | 11 | 5 |

→ { } → P1 → S1 = 0

→ {1, 6, 11, 5} → P2 → S2 = 23



Range = 0 — 23 {o/p will lie b/w this



Range = $\sum arr[i]$

$|S_1 - S_2| = \text{minimum}$

or

$\Rightarrow \quad S_2 - S_1 = \text{minimum}$

$\Rightarrow \quad (\text{Range} - S_1) - S_1 = \text{min}$

$\Rightarrow \quad (\text{Range} - 2S_1) = \text{minimum}$

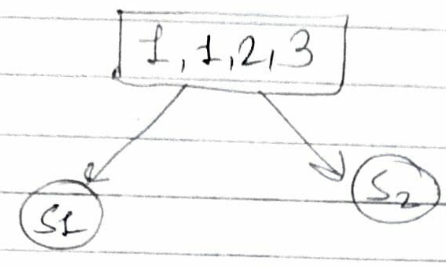
Program-

```
int findMin(int arr[], int n)
{ int sum=0;
  for (int i=0; i<n; i++)
    sum+=arr[i];
  bool t[n+1][sum+1];
  for(int i=0; i<=n; i++)
    t[i][0]=true;
  for (int i=1; i<=sum; i++)
    t[0][i]=false;
  for (int i=1; i<=n; i++)
  { for(int j=1; j<=sum; j++)
    { t[i][j] = t[i-1][j];
      if (arr[i-1]<=j)
      t[i][j] |= t[i-1][j-arr[i-1]];
    } }
  int diff;
  for (int j=sum/2; j>=0; j--)
  { if (t[n][j]==true)
    { diff = sum-2*j;
      break;
  } }
  return diff; }
print(findMin (arr, n));
```

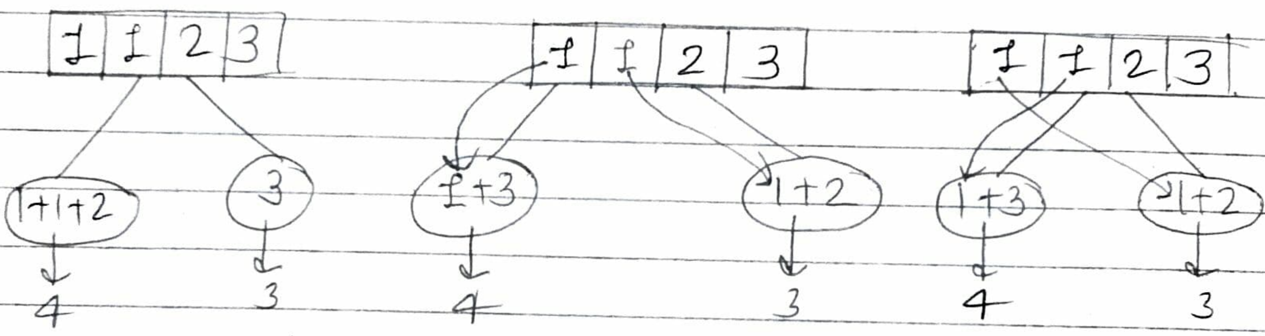
* Count the number of subset sum with a given difference -

arr[] :	1 | 1 | 2 | 3
Diff :	1

O/p :- 3



$$\boxed{S_1 - S_2 = Diff}$$



4-3 = 1   4-3 = 1   4-3 = 1

$\rightarrow$ sum (S1) − sum (S2) = diff — ①

$\rightarrow$ sum (S1) + sum (S2) = sum (arr) — ②

① + ② $\rightarrow$

$\Rightarrow$ 2 sum (S1) = diff + sum (arr)

$$\Rightarrow \boxed{\text{sum}(S1) = \frac{\text{diff} + \text{sum}(arr)}{2}}$$

$$\text{sum}(S1) = \frac{1+7}{2} = 4$$

Count = ?

This problem is reduced in 'Count of subset sum with a Given Sum'.

* Target Sum -

arr: | 1 | 1 | 2 | 3 | → +/-

Sum : 1

O/p :- 3

we have to assign + or - sign before every array elements so that sum of array is equal to given sum.

Let say -

$$+1-1-2+3$$
$$=1=\text{sum}$$

And then we have to count all possible ways to do so, in the output.

$(+1-1-2+3) = 1$ | $(-1+1-2+3) = 1$ | $(+1+1+2-3) = 1$

$+1+3$ → $+1+3$ → S_1

$-1-2$ → $-(1+2)$ → S_2

$$\boxed{S_1 - S_2 = \text{sum}}$$

* This problem is reduced in the previous problem - 'Count the no' of subset with a given diff'.