Given n integers a_i; $i = 1, n$; and an integer B. We aim to find the subset of integers a_i whose sum is maximized and does not exceed B.

Example: Input: n = 4; a[] = [5,1,3,4]; B = 11;

Output (solution) : {5,1,4} with max sum (objective) = 10.

1) **What is the type of this optimization problem?**
2) **How is called each of the solutions {5,1,4} ; {5,4} ; and {5,3,4} for this problem.**
3) **We define the decision variable $x = (x_i)$; $x_i \in \{0, 1\}$ such that:**

$$x_i = \begin{cases} 1 \text{ if } a_i \text{ belongs to the solution} \\ 0 \text{ otherwise} \end{cases}$$

   a) **Give the encoding of each of the solutions {5,1,4} and {5,4}.**
   b) **Write the mathematical formulation of this problem.**
   c) **Deduce the search space for this problem and its size.**
   d) **Show that this problem is a special case of the knapsack problem.**
   e) **Deduce the complexity class of this problem.**
4) **Apply dynamic programming method to solve the example above then write the corresponding algorithm.**

Answers:

1) Combinatorial optimisation ……0.25
2) {5,1,4} is an optimal solution; {5,4} is a candidate (approximate) solution; {5,3,4} candidate but unfeasible solution. ……0.75
3) a) {5,1,4} = 1101 ; {5,4} = 1001 ……1

   b) $\begin{cases} \max \sum_{i=1}^{n} x_i a_i \\ \text{s.t.} : \sum_{i=1}^{n} x_i a_i \leq B \\ x_i \in \{0,1\} \end{cases}$ ……1

   c) search space = $\{0,1\}^n$; $|\text{search space}| = 2^n$ ……0.50

   d) this is a special case of KSP where item = value = weight = a_i and capacity = B. ……0.50

   e) since KSP is NP-complete, therefore this problem is NP-complete. ……0.50

4) ……3

item	a_i	0	1	2	3	4	5	6	7	8	9	10	11	sol
		0	0	0	0	0	0	0	0	0	0	0	0	
1	5	0	0	0	0	0	5	5	5	5	5	5	5	1
2	1	0	1	1	1	1	5	6	6	6	6	6	6	1
3	3	0	1	1	3	4	5	6	6	6	9	9	9	0
4	4	0	1	1	3	4	5	6	7	8	9	10	10	1

Algorithm

```
// Input n, B , a[]
for (j=0; j<=B; j++) dp[0][j] = 0;
for (i=0; i<=n; i++) dp[i][0] = 0;
for (i=1; j<=n; i++)
      for (j=1; j<=B; J++)
            if (a[i] >j)
                dp[i][j] = dp[i-1][j]
            else
                dp[i][j] = max (dp[i-1][j]; dp[i-1][j-a[i]]+a[i])
j = B ; i = n ;
while ( i>0)
    {if (dp[i][j] != dp[i-1][j])  {x[i] = 1; j- = a[i];}
     i--; }
// Output dp[n][B] ; x[]
```

……2.5