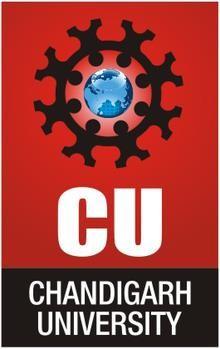
**CHANDIGARH UNIVERSITY**

UNIVERSITY INSTITUTE OF ENGINEERING

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



|  |  |
| --- | --- |
| **Submitted By:                                                                          Submitted To:**  Yash Gupta ER. Monika(E12802) | |
| **Subject Name** | Design Analysis and Algorithm |
| **Subject Code** | 20CSP\_312 |
| **Branch** | CSE |
| **Semester** | 5th |

**LAB -INDEX**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sr.No** | **Program** | **Date** | **Evaluation** | | | | **Sign** |
| **LW(12)** | **VV(8)** | **FW(10)** | **Total (30)** |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Experiment 7**

**1. Aim/Overview of the practical:**

Code to implement 0-1 Knapsack using Dynamic Programming

**2. Algorithm:**

* 1. Calculate the profit-weight ratio for each item or product.
  2. Arrange the items on the basis of ratio in descending order.
  3. Take the product having the highest ratio and put it in the sack.
  4. Reduce the sack capacity by the weight of that product.
  5. Add the profit value of that product to the total profit.
  6. Repeat the above three steps till the capacity of sack becomes 0 i.e. until the sack is full.

for w = 0 to W do c[0, w] = 0

for i = 1 to n do c[i, 0] = 0

for w = 1 to W do if wi ≤ w then

if vi + c[i-1, w-wi] then

c[i, w] = vi + c[i-1, w-wi]

else c[i, w] = c[i-1, w]

else c[i, w] = c[i-1, w]

**3. Steps for experiment/practical/Code:**

#include <stdio.h>

int max(int a, int b)

{

return (a > b) ? a : b;

}

int knapSack(int W, int wt[], int val[], int n)

{

int i, w;

int K[n + 1][W + 1];

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

{

for (w = 0; w <= W; w++)

{

if (i == 0 || w == 0)

K[i][w] = 0;

else if (wt[i - 1] <= w)

K[i][w] = max(val[i - 1]

+ K[i - 1][w - wt[i - 1]],

K[i - 1][w]);

else

K[i][w] = K[i - 1][w];

}

}

return K[n][W];

}

int main()

{

int val[] = { 60, 100, 120 };

int wt[] = { 10, 20, 30 };

int W = 50;

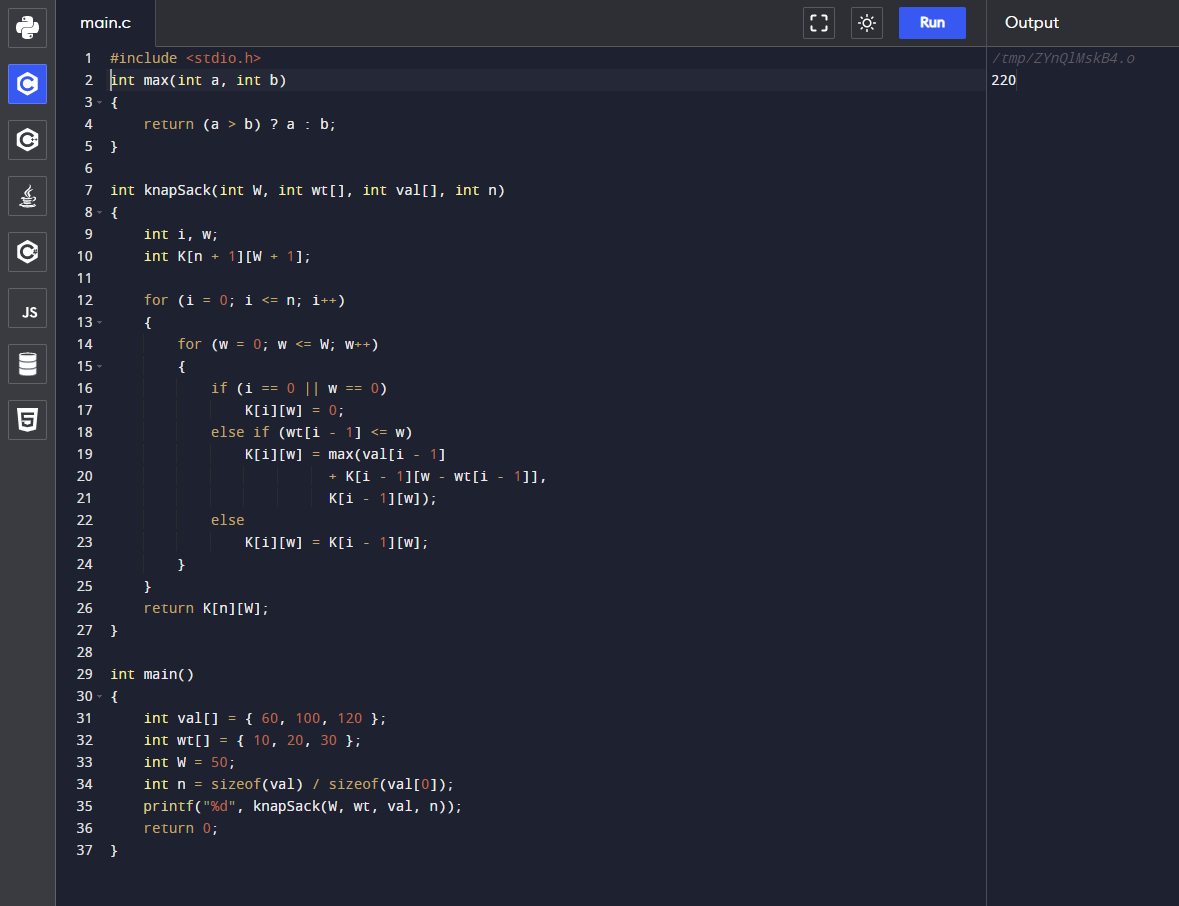
int n = sizeof(val) / sizeof(val[0]);

printf("%d", knapSack(W, wt, val, n));

return 0;

}

**4. Result/Output/Writing Summary:**



**5. Observations/Discussions/ Complexity Analysis:**

Time complexity – O (N\*W).

‘N’ – is the number of weight, ‘W’ – is capacity

6. **Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |