Knapsack

```
def knapSack(W, wt, val):
    n=len(val)
    table = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for j in range(W + 1):
            if i == 0 or j == 0:
                table[i][j] = 0
            elif wt[i-1] <= j:
                table[i][j] = max(val[i-1] + table[i-1][j-wt[i-1]],  table[i-1][j])
            else:
                table[i][j] = table[i-1][j]

    return table[n][W]

val = [50,100,150,200]
wt = [8,16,32,40]
W = 64

print(knapSack(W, wt, val))
```

BCT 3RD

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Bank{
    address public owner;
    mapping(address => uint256) private userbalance;



    constructor() public {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, 'you are not the owner of this contract');
       _;
    }

    function deposite() public payable returns(bool) {
        require(msg.value > 10 wei , 'please deposite atleast 10 wei');
        userbalance[msg.sender] += msg.value;
        return true;
    }

    function withdraw(uint256 _amount) public payable returns(bool) {
        require( _amount <=userbalance[msg.sender], 'you dont have sufficient funds');
        userbalance[msg.sender] -= _amount;
        payable(msg.sender).transfer(_amount);
        return true;
    }
```

```
    function getbalance() public view returns(uint256) {
        return userbalance[msg.sender];
    }

    function getcontractbalance() public view onlyOwner returns(uint256) {
        return address(this).balance;
    }

    function withdrawfunds(uint256 _amount) public payable onlyOwner returns(bool) {
        payable(owner).transfer(_amount);
        return true;
    }

}
```

4th BCT

```
// Solidity program to implement
// the above approach
pragma solidity >= 0.7.0<0.8.0;

// Build the Contract
contract MarksManagmtSys
{
    // Create a structure for
    // student details
    struct Student
    {
        int ID;
```

```
        string fName;
        string lName;
        int marks;
    }
    address owner;
    int public stdCount = 0;
    mapping(int => Student) public stdRecords;

    modifier onlyOwner
    {
        require(owner == msg.sender);
        _;
    }

    constructor()
    {
        owner=msg.sender;
    }

    // Create a function to add
    // the new records
    function addNewRecords(int _ID, string memory _fName, string memory _lName, int _marks)
public onlyOwner
    {
        // Increase the count by 1
        stdCount = stdCount + 1;
        // Fetch the student details
        // with the help of stdCount
        stdRecords[stdCount] = Student(_ID, _fName, _lName, _marks);
    }
```

```
    // Create a function to add bonus marks
    function bonusMarks(int _bonus) public onlyOwner
    {
        stdRecords[stdCount].marks = stdRecords[stdCount].marks + _bonus;


    }
}
```