https://leetcode.com/problems/house-robber/description/

Brute Force Solutions

In [ ]:
```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) <= 2:
            return max(nums)

        include = nums[0] + self.rob(nums[2:])
        exclude = self.rob(nums[1:])

        return max( [include, exclude] )
```

In [ ]:
```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        return self.rob_util(nums, 0)

    def rob_util(self, nums, i):
        if i == len(nums)-1:
            return nums[i]
        elif i == len(nums)-2:
            return max( [nums[i], nums[i+1]])

        include = nums[i] + self.rob_util(nums, i+2)
        exclude = self.rob_util(nums, i+1)

        return max( [include, exclude] )
```

In [ ]:
```python
```

Memoization

In [ ]:
```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        cache = {}
        return self.rob_util(nums, 0, cache)

    def rob_util(self, nums, i, cache):
        if i == len(nums)-1:
            return nums[i]
        elif i == len(nums)-2:
            return max( [nums[i], nums[i+1]])

        if i in cache:
            return cache[i]

        include = nums[i] + self.rob_util(nums, i+2, cache)
        exclude = self.rob_util(nums, i+1, cache)

        cache[i] = max( [include, exclude] )
        return cache[i]
```

In [ ]:

In [ ]:

Knapsack problem

Given max weight W
array of weights and values
optimize for max value, staying within the limit of max weight
Return max value that can be put in knapsack

values [10 20 60]
weights [20 40 30]

W = 60
maxValue ?

```
                20
          / 60        \ 40
          40           40
       / 60  \ 20    / 40     \ 0
```

In [8]:
```python
def knapsack(w, v, W):
    if W == 0:
        return 0
    if len(w) == 0:
        return 0

    include = 0
    if w[0] <= W: # there is capacity to use current weight
        include = v[0] + knapsack(w[1:], v[1:], W-w[0])
    exclude = knapsack(w[1:], v[1:], W)

    return max([include, exclude])


print(knapsack([20,40,30],[10,20,60], 60))
print(knapsack([20,40,30],[10,20,60], 70))
```

```
70
80
```

In [11]:
```python
def knapsack(w, v, W):
    return knapsack_util(w, v, W, 0)

def knapsack_util(w, v, W, i):
    if W == 0:
        return 0
    if i == len(w):
        return 0

    include = 0
    if w[i] <= W: # there is capacity to use current weight
        include = v[i] + knapsack_util(w, v, W-w[i], i+1)
    exclude = knapsack_util(w, v, W, i+1)

    return max([include, exclude])

print(knapsack([20,40,30],[10,20,60], 60))
print(knapsack([20,40,30],[10,20,60], 70))
```

```
70
80
```

In [ ]:
```python

```