Recursion 01

The fundamentals

Designing recursive algorithms/functions

1. Base case
2. Making progress
3. Design rule
4. No compound interest

```python
def printOut(n):
    if n < 10:
        printDigit(n)
        return
    printOut(int(n/10))
    printDigit(n%10)
```

```
def printOut(n):
    if n < 10:
        printDigit(n)
        return
    printDigit(n%10)
    printOut(int(n/10))
```

Print digits of n in reverse order

Multiply a value by an integer

```
def mul(val, n):
    if n==1:
        return val
    t = mul(val, n-1)
    return val + t
```

Integral power of a number

```
def power (num, p):
    if p==0:
        return 1
    t = power(num, p-1)
    return num * t
```

Return product of all values in an array

```
def prod(a, size)
    if size==0:
        return 0.0
    t = prod(a, size-1)
    return t * a[size-1]
```

1.0

Maximum/minimum value in an array

```
def max(nums, count):
    if count==1:
        return nums[0]
    t = max(nums, count-1)
    lv = a[count-1]
    return lv if lv > t else t
```

Summing up

- There must be some n/size/count as argument of the recursive function.
- That should make progress in recursive calls towards the base case.
- Base case(s) are for its very small values; typically, 0 or 1.
- In Base case(s), answer should be direct (without any computation).

Integral power of a number (version 2)

```
def power (num, p):
    if p==0:
        return 1
    t1 = power(num, int(p/2))
    t2 = power(num, p-int(p/2))
    return t1 * t2
```

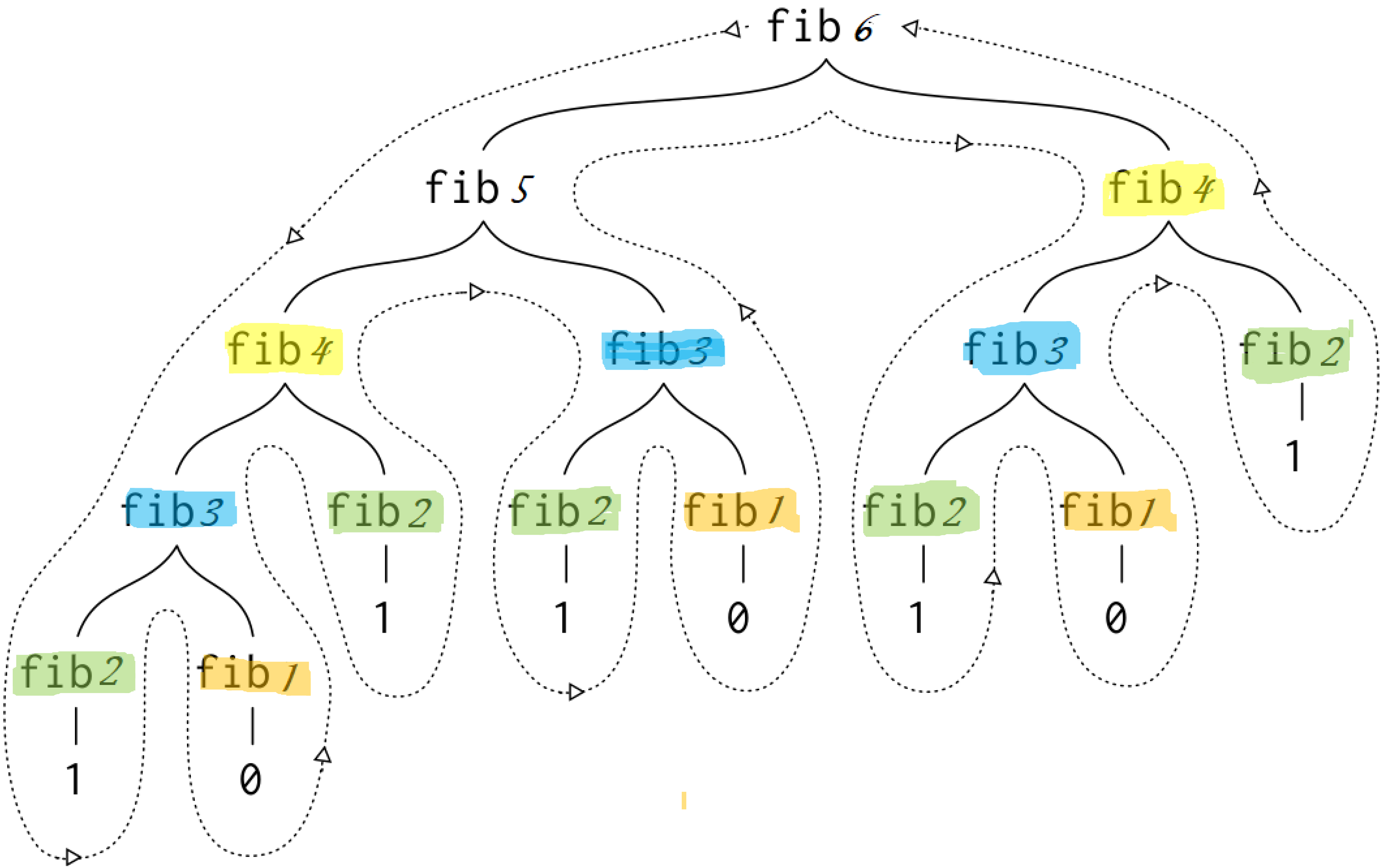
issue in making progress

Integral power of a number (version 3)

```
def power (num, p):
    if p==0:
        return 1
    v = power(num, int(p/2))
    v = v * v
    return v if p%2==0 else v * num
```

n'th Fibo number

```
def fib(n):
    if (n==1 || n==2):
        return n-1
    return fib(n-1) + fib(n-2)
```



fib 6
fib 5
fib 4
fib 4
fib 3
fib 3
fib 2
fib 3
fib 2
fib 2
fib 1
fib 2
fib 1
1
fib 2
fib 1
1
1
0
1
0
1
0

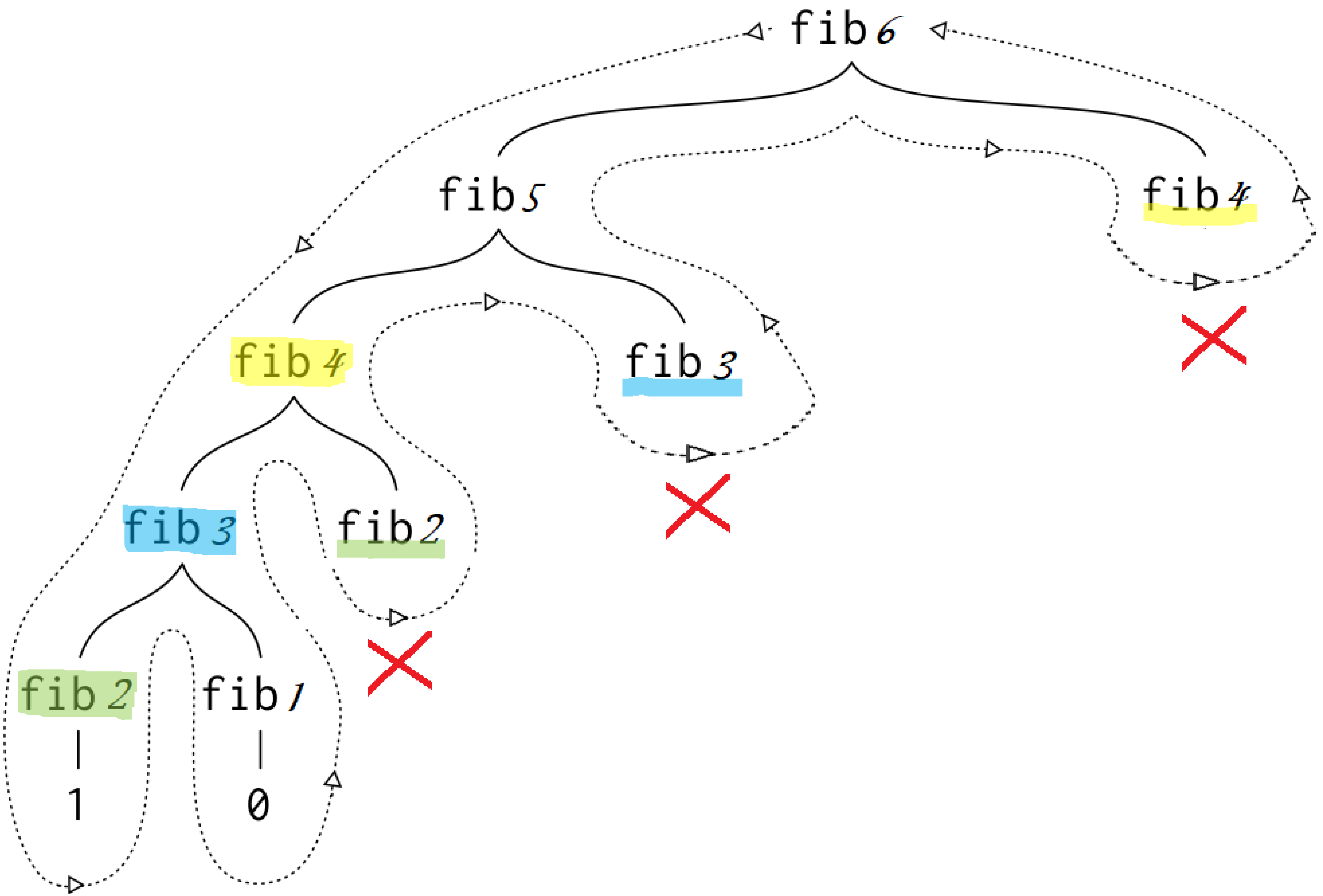
n'th Fibo number (efficient logic)

```
def fibR(n, t):
    if t[n-1]==-1:
        if n==1 OR n==2:
            t[n-1] = n-1
        else:
            t[n-1] = fibR(n-1, t) + fibR(n-2, t)
    return t[n-1]
```

<u>how to call</u>

```
print(fibo(15))

def fibo(n):
    tmp = [-1]*n
    return fibR(n, tmp)
```


fib 6
fib 5
fib 4
fib 4
fib 3
fib 3
fib 2
fib 2
fib 1
1
0

using better formulation, and logic tricks and techniques, algo's can be make efficient

formal theory will be discussed later in the course as divide and conquer, greedy approach, memoization and dynamic programming

Wrapper functions

vs

Helper or auxiliary functions

Linear search

```
def search (nums, size, val):
	if size==0:
		return false
	return search(nums, size-1, val) or (nums[size-1] == val)
	# return (nums[size-1] == val) or search(nums, size-1, val)
```

Linear search

```
def position (nums, size, val):
        if size==0:
                return -1; # better to raise exception
        if nums[size-1] == val:
                return size-1
        else:
                return position(nums, size-1, val)
```

Binary search

```
def bsearchR(nums, li, hi, val):
    if (li > hi):
        return -1 # better to raise exception
    mi = int((li + hi ) / 2)
    if (nums[mi] == val):
        return mi
    else if (val < nums[mi]):
        return bsearchR(nums, li, mi, val)
    else if (val < nums[mi]):
        return bsearchR(nums, mi, hi, val)
```

```
# call an overloaded function
p = bsearch(a, n, x)
# which should call the function at left
```

Practice Questions

Write recursive function and corresponding iterative functions for the following tasks:

1. To return the SUM of 1st N natural numbers
2. To return the PRODUCT of 1st N negative integers
3. To return the VALUE of 1 + 1/2 + 1/3 + 1/4 + ... + 1/N
4. To return the COUNT of even numbers in an array of size N
5. To return the BIONOMIAL COEFFICIENT (N, K) or nCk

Also, write main functions to test the results of above mentioned functions.

Integer Multiplication

Karatsuba's Multiplication

Let A and B are two n-digit numbers, and C = A X B (product of A and b)

Now let $A = A_l \times 10^{n/2} + A_r$ and $B = B_l \times 10^{n/2} + B_r$

So that $\mathbf{A \times B = (A_l \times 10^{n/2} + A_r) \times (B_l \times 10^{n/2} + B_r)}$

$$\mathbf{= A_lB_l \times 10^n + (A_lB_r + A_rB_l) \times 10^{n/2} + A_rB_r}$$

4 unique n/2 digit multiplications

Again let

$4375 = 43 \times 10^2 + 75$

$$P_1 = A_lB_l$$
$$P_2 = (A_l + A_r) \times (B_l + B_r)$$
$$P_3 = A_rB_r$$
$$S = P_2 - P_1 - P_3 = (A_lB_r + A_rB_l)$$

And so

$$\mathbf{A \times B = P_1 \times 10^n + S \times 10^{n/2} + P_3}$$
$$\mathbf{= P_1 \times 10^n + (P_2 - P_1 - P_3) \times 10^{n/2} + P_3}$$

3 unique n/2 digit multiplications

```
def MUL(x,y,n):
    if n == 1:
        return x*y

    m = int(n/2)
    zeros = 10**m

    xl = int(x / zeros)
    xr = x - xl * zeros
    yl = int(y / zeros)
    yr = y - yl*zeros

    xlyl = MUL(xl, yl, m)
    xlyr = MUL(xl, yr, m)
    xryl = MUL(xr, yl, m)
    xryr = MUL(xr, yr, m)

    return xlyl*(zeros**2) + (xlyr+xryl)*zeros + xryr

def main():
    num1 = 73624239617454239617455926096592
    num2 = 23737362423961745592624239961745
    res = MUL(num1, num2, 32)
    print(num1*num2)
    print(res)

main()
```

Sorting Function

```
def SortRec(a, n):
    if n <= 1:
        return
    SortRec(a, n-1)
    bubble(a)
```

```
def bubble(a):
    i = len(a) - 1
    while i > 0:
        if a[i] < a[i-1]:
            t = a[i]
            a[i] = a[i-1]
            a[i-1] = t
        i = i-1
```

0/1 Knapsack Problem

The 0-1 Knapsack Problem

Problem statement

A thief robbing a store finds n items; the ith item is worth v_i dollars and weights w_i pounds. He wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack. What items should be taken?

Formally, the problem can be stated as follws:

- Input: n items of values $v_1, v_2, \ldots, v_n$ and of the weight $w_1, w_2, \ldots, w_n$, and a total weight W, where v_i, w_i and W are positive integers.
- Output:a subsect $\mathcal{S} \subseteq \{1, 2, \ldots, n\}$ of the items such that

$$\sum_{i\in\mathcal{S}} w_i \leq W \quad \text{and} \quad \sum_{i\in\mathcal{S}} v_i \quad \text{is maximized.}$$

This is called the **0-1 knapsack problem** because each item must either be taken or left behind; the thief cannot take a fractional amount of an item or take an item more than once.

Example

$n = 9,$

$v = \langle 2, 3, 3, 4, 4, 5, 7, 8, 8\rangle,$

$w = \langle 3, 5, 7, 4, 3, 9, 2, 11, 5\rangle,$

$W = 15.$

Max benifit from problem at left 23

– why?

set of items to take
$\{9, 7, 5, 4\}$

```
def KS(n, C, v, w):
        if n==0 or C <= 0:
                return 0, []
        if w[n-1] > C:
                return KS(n-1, C, v, w)
        exKSv, exKSs = KS(n-1, C, v, w)
        inKSv, inKSs = KS(n-1, C-w[n-1], v, w)
        if exKSv >= inKSv + v[n-1]:
                return exKSv, exKSs
        else:
                inKSv = inKSv + v[n-1]
                inKSs.append(n)
                return inKSv, inKSs
```