Maximum Sum Subarray

Thomas Schwarz, SJ

Task

- Find the maximum of sums of a slice in an array
- Example:

```
[ 6, -10, -12,   1,   7,  16, -25,  -4,  11,   8,  14,  21,   4,
  8,  16,  19,   6,  11,  -8,  11,   6,  -9,  13,  -2,   4,   1,
 10,   7,   8,   5,  22,  20,  -4,   0,  30,  15,   8,   4,  16,
  5,   5,   2,   3,  25,   5,  -5,  -1,   0, -18,   0,  20,  13,
 -1,  16,  -5,   2,  31, -24,  -1,   1, -28,  19,  11,  -5,  -5,
 12,   2,   7,  18,  20, -12,   7,  -9,  19,  10,   1,   9,  39,
-12,  16,   5,  11,  24,   5, -10,  22, -10,  23, -25,   5,   7,
-25,   1,   8,   4,   7, -10,  11, -23,  -5]
```

- with maximum sub-array 537

Simple Algorithm

- Cubic Algorithm
  - Try out all beginning indices and all ending indices

```
def simple_max(lista):
    best_sum, best_i, best_j = -10000, -1,-1
    for i in range(0, len(lista)):
        for j in range(i+1, len(lista)):
            if np.sum(lista[i:j]) > best_sum:
                best_i, best_j = i, j
                best_sum = np.sum(lista[i:j])
    return best_i, best_j, best_sum
```

  - This gives $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (j-i) = \frac{1}{6}(n^3 - n)$ times we access an array element

Divide and Conquer

- Divide and conquer algorithm:
  - Divide the array into halves
  - Out of two halves:
    - Calculate four different values:
      - Total maximum sum sub-array
      - Total maximum sum sub-array starting on left
      - Total maximum sum sub-array ending at right
      - Total sum

Divide and Conquer

- For simplicity: just calculate the maxima and not the indices
-

```
def max_sub_array(lista):
    #divide
    left = lista[:len(lista)//2]
    right = lista[len(lista)//2:]
    #calculate and then return four values

    return total, ttl_from_left, ttl_from_right, suma
```

Divide and Conquer

- For the calculation, we get the four values for the left and right half

```
def max_sub_array(lista):
    #divide
    left = lista[:len(lista)//2]
    right = lista[len(lista)//2:]
    #recursive step
    ltotal, lttl_from_left, lttl_from_right, lsuma =
max_sub_array(left)
    rtotal, rttl_from_left, rttl_from_right, rsuma =
max_sub_array(right)

    suma = lsuma+rsuma

    #Calculate the other three return values as well

return total, ttl_from_left, ttl_from_right, suma
```

Divide and Conquer

- Getting the sum is easy:
  - Just add up the sums of the left and right

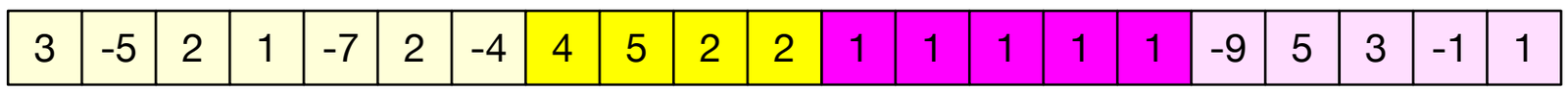
Divide and Conquer

- How do we calculate the maximum sum sub-array from the information in the left and right halves:
  - Case 1:
    - The total maximum sub-array is the maximum of the total maximum sub-arrays of both sides

1	-5	2	-3	4	5	18	2	-2	1	-1	-1	1	-8	6	-2	1	3	5	2	1

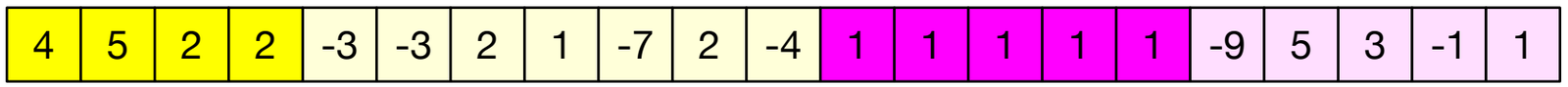
Divide and Conquer

- Case 2:
  - The best choice is composed of the maximal one on the left ending at the end and the one on the right starting at the beginning

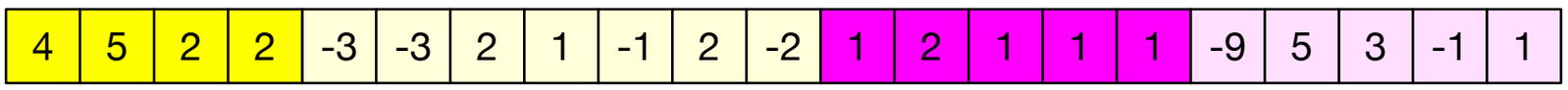


Divide and Conquer

- How about starting on the left?
  - Case 1: Best case is the one starting on the left



  - Case 2: Best case is all of left plus the one subarray starting on the right



- All of left gives you 9, violet part gives you 6, total is 15
  - This is why we also calculate the sum of each part

Divide and Conquer

- Similarly, maximum sum sub-array ending at the end could be:
    - Best sub-array ending at the end of the left sub-array plus all of the right half
    - Just the best sub-array ending at the end of the right half

Divide and Conquer

- Time analysis:
  - Recurrence is $T(n) = 2T(n/2) + \Theta(1)$
  - MT: Compare $\Theta(1)$ with $n^{\log_2(2)} = 1$
  - $T(n) = \Theta(n)$

Implementation

- In Python, you can use tuples and tuple extraction in order to pass several values

```
def maxsub(lista):
    if len(lista)==1:
        return max(0,lista[0]), max(0,lista[0]),  max(0,lista[0]),
lista[0]
    else:
        left = lista[:len(lista)//2]
        right = lista[len(lista)//2:]

        ltot, lbeg, lend, lsum = maxsub(left)
        rtot, rbeg, rend, rsum = maxsub(right)


        return mytot, mybeg, myend, mysum
```