Enumeration

Pseudo-code

There are a lot of websites on the internet that talk about the enumeration in the form of codes and the logic in the enumeration is almost the same for all. Basically, the pseudo code that I do here is based on python, but it should work with any other language since the idea is the same.

Here is the Enumeration pseudo-code :

```
Enum(A[1,...,N])
   if N == 0
      return 0, A
    else
      max_sum = -Infinity
for i from 1 to N
      for j from i to N
         curr_sum = 0
         for k from i to j
            curr_sum = curr_sum + A[k]
            if curr_sum > max_sum
               max_sum = curr_sum
               start_index = i
               end_index = j
   return max_sum, A[begIndex, ..., endIndex]
```

Theoretical Run-time Analysis

Here we would like to know from our pseudo-code and estimate the theoretical r un-time for the enumeration. From the iterative pseudo-code that we have, we can try to count the theoretical run-time.

First, we can see from the outer i loop. So the first outer loop for i runs from 1 to N, and th en we can see there are two inner loop inside that loop, which is j that runs from i to N and then k that run from i to j. Here from the iterative work, we can know that all the for loop are connected each other and works to result the curr_sum, which means that based on th at, our f(n) is $\Theta(N^3)$. Then we can compute the number of iteration as :

$\sum_{i=1}^{N}\sum_{j=i}^{N}\sum_{k=i}^{j}\Theta(1)$. Inputting that calculation of iteration to calculator, and we will get : $\sum_{i=1}^{N}\frac{1}{2}(i^2-2iN-3i)+\sum_{i=1}^{N}\frac{1}{2}(N^2+3N+2)\Theta(1)$ with i will remain as a constant . By keep doing the calculation and i remain as a constant, We will get an equation in the e nd with N^3, N^2, and N, which is : $\frac{1}{6}N^3+\frac{1}{2}N^2+\frac{1}{3}N\cdot\Theta(1)=\Theta(N^3)$.

Finally, based on our calculation, theoretically, the run-time for this algorithm is Cubical.

Divide and Conquer

Pseudo-code

The "Divide and Conquer" maximum sub-array algorithm is described by the following pseudo-code:

```
Div_and_Conq(A[1,...,N]){
  if N == 0 {
    return 0, A
  } else if N == 1 {
    return A[0], A
  }

  tmp_max = 0
  mid_max = 0
  mid_start = 0
  mid_end = 0
  left_max = 0
  right_max = 0

  midpoint = N / 2

  mid_start = midpoint
  mid_end = midpoint

  for i from A[N,...,midpoint] {
    tmp_max = tmp_max + A[i]
    if tmp_max > left_max {
      left_max = tmp_max
      mid_start = i
```

```
        }
    }
    tmp_max = 0

    for i from A[midpoint,...,N] {
        tmp_max = tmp_max + A[i]
        if tmp_max > right_max {
            right_max = tmp_max
            mid_end = i + 1
        }
    }
    mid_max = left_max + right_max

    left_max, left_subarray = DIVIDE_AND_CONQUER(A[0,...,midpoint])
    right_max, right_subarray = DIVIDE_AND_CONQUER(A[midpoint,...,N])

    if mid_max >= left_max and mid_max >= right_max {
        return mid_max, A[mid_start,...,mid_end]
    } else if left_max >= right_max and left_max > mid_max {
        return left_max, left_subarray
    } else if right_max > left_max and right_max > mid_max {
        return right_max, right_subarray
    }
}
```