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.

The "Enumeration" maximum sub-array algorithm is described by the following pseudo-code:

```
ENUMERATION-MAX-SUBARRAY(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 {
      current_sum = 0
      for k from i to j {
        current_sum = current_sum + A[k]
        if current_sum > max_sum {
          max_sum = current_sum
          start_index = i
          end_index = j
        }
      }
    }
  }
  return max_sum, A[start_index, ..., end_index]
}
```

Theoretical Run-time Analysis

The outer i loop runs from 1 to N, the first inner j loop runs from i to N, and the second inner loop runs from i to j. We can compute the number of iterations as :

$$\sum_{i=1}^{N}\sum_{j=i}^{N}\sum_{k=i}^{j}\Theta(1)$$

$$\sum_{i=1}^{N}\sum_{j=i}^{N}(j-i+1)\Theta(1)$$

$$\sum_{i=1}^{N}\left(\sum_{j=i}^{N}(1-i)+\sum_{j=i}^{N}j\right)\Theta(1)$$

$\sum_{i=1}^{N}((i-1)(i-N-1)-\frac{1}{2}(i+N)(i-N-1))\Theta(1)$

$\sum_{i=1}^{N}((i^2-iN-2i+N+1)-\frac{1}{2}(i^2-i-N^2-N))\Theta(1)$

$\sum_{i=1}^{N}\frac{1}{2}i^2-iN-\frac{3}{2}i+\frac{1}{2}N^2+\frac{3}{2}N+1)\Theta(1)$

$\sum_{i=1}^{N}\frac{1}{2}(i^2-2iN-3i)+\sum_{i=1}^{N}\frac{1}{2}(N^2+3N+2)\Theta(1)$

So we can find the sums term by term for the terms with i while the sum of terms without i will remain constant therefore :

$\sum_{i=1}^{N}\frac{1}{2}(N^2+3N+2)=(\frac{1}{2}N^2+\frac{3}{2}N+1)\sum_{i=1}^{N}1$
$(\frac{1}{2}N^2+\frac{3}{2}N+1)\sum_{i=1}^{N}1=(\frac{1}{2}N^2+\frac{3}{2}N+1)*N$

$(\frac{1}{2}N^2+\frac{3}{2}N+1)*N=\frac{1}{2}N^3+\frac{3}{2}N^2+N$

$\sum_{i=1}^{N}\frac{1}{2}i^2=\frac{1}{2}(\frac{1}{6}N(N+1)(2N+1))$

$\frac{1}{2}(\frac{1}{6}N(N+1)(2N+1))=\frac{1}{6}N^3+\frac{1}{4}N^2+\frac{1}{12}N$

$\sum_{i=1}^{N}\frac{1}{2}(-2iN)=-\frac{1}{2}N(N+1)*N=-\frac{1}{2}(N^3+N^2)$

$\sum_{i=1}^{N}\frac{1}{2}(-3i)=\frac{1}{2}(-\frac{3}{2}N(N+1))=-\frac{3}{4}(N^2+N)$

Thus the runtime of the whole algorithm is equivalent to $\Theta(N^3)$.

Divide and Conquer

Pseudo-code

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

```
DIVIDE_AND_CONQUER(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
    }
}
```