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

}

}

}

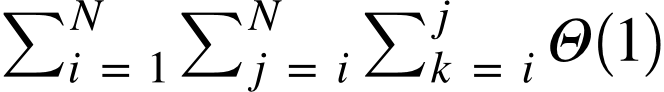
}

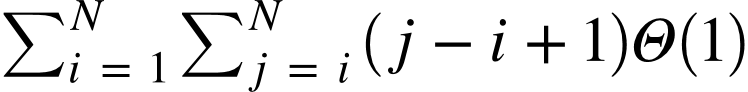
return max\_sum, A[start\_index, ..., end\_index]

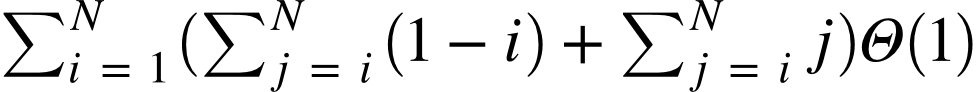
}

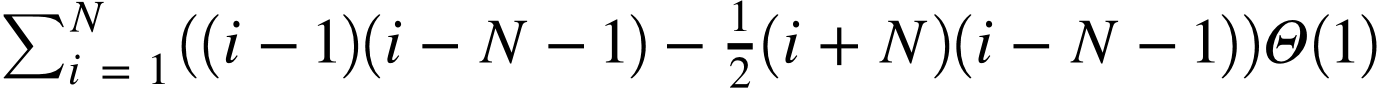
Theoretical Run-time Analysis

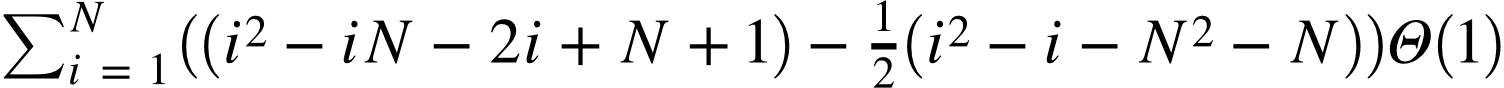
The outer i loop runs from 1 to N, the first inner j loop runs from i to N, and the second inner loop runs from i to j. We can compute the number of iterations as :

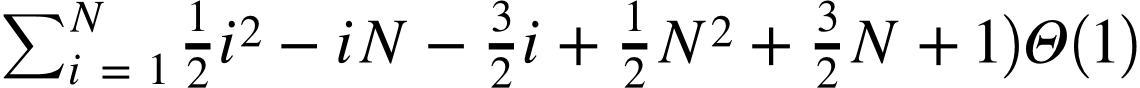


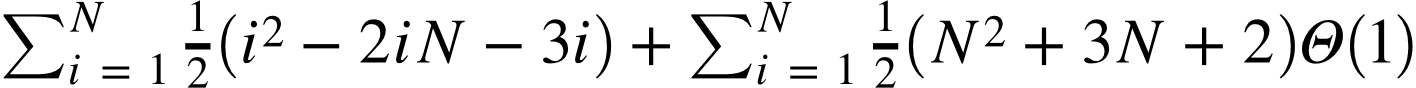




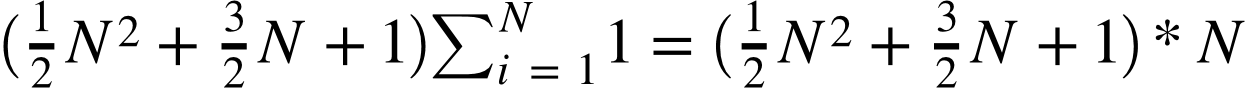
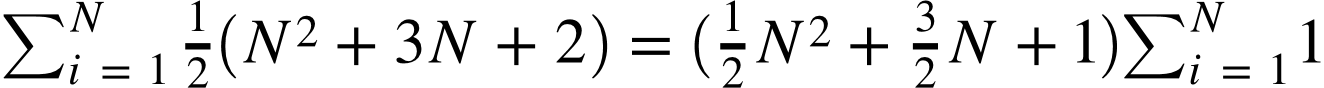


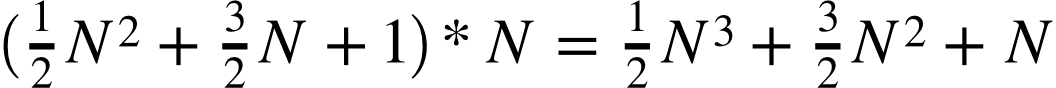


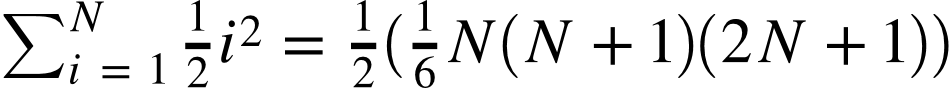


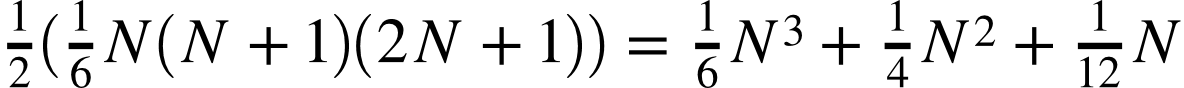


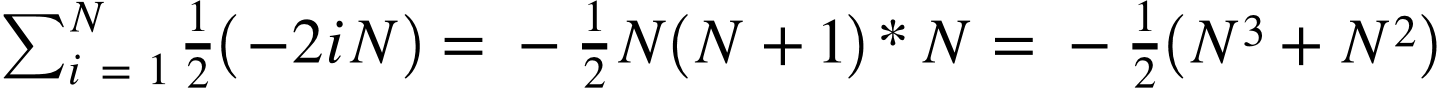
So we can find the sums term by term for the terms with i while the sum of terms without i will remain constant therefore :

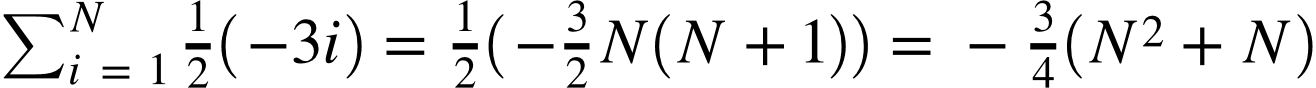


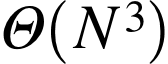










Thus the runtime of the whole algorithm is equivalent to .

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

}

}