Analysis of Divided-and-Conquer Algorithm

Kexin Ding

1. O(N logN) divided-and-conquer algorithm

1) Explanation for Algorithm

There will be three cases happens in the process of finding maximum subarray:

(1) the start and the end index of the maximum subarray are completely in the left subproblem

(2) the start and the end index of the maximum subarray are completely in the right subproblem

(3) the maximum subarray is across the split point. The start and the end index of the maximum subarray are in two sides of the split point.

The first two cases can be found recursively. The last case can be found in linear time.

In this algorithm:

Divide: the subarray into two subarrays of equal size as possible by finding the midpoint mid of the subarrays.

Conquer: by finding a maximum subarray of A[low .. mid] and A[mid+1 .. high].

Combine: by also finding a maximum subarray that crosses the midpoint, and using the best solution of the three (the subarray crossing the midpoint and the best of the solutions in the Conquer step).

2) Source code and screenshot execution results

Please check the source code in mss_nlgn.py

Input:

```
Enter the list of numbers: 1 -4 3 -4
```

Test case1:

Index	0	1	2	3
Arr[i]	1	-4	3	-4

Result:

```
Enter the list of numbers: 1 -4 3 -4
The maximum subarray starts at index 2, ends at index 2 and has sum 3.0.
```

Test case2:

Index	0	1	2	3	4	5
Arr[i]	1	-4	3	4	-2	6

Result:

```
Enter the list of numbers: 1 -4 3 4 -2 6
The maximum subarray starts at index 2, ends at index 5 and has sum 11.0.
```

3) Time complexity analysis

$T(N) = 2T(\frac{N}{2}) + \theta(N)$

Use master method:

$a = 2$ $b = 2$ $f(N) = \theta(N)$

$N^{\log_b a} = N^{\log_2 2} = N^1 = N$

$\lim_{n\to\infty} \frac{f(N)}{g(N)} = \lim_{n\to\infty} \frac{\theta(N)}{N} = 1$

Hence according to the case2 of master method, $T(N) = \theta(N\ logN)$

2. O(N) divided-and-conquer algorithm

1) Explanation for Algorithm

We use a for loop in this O(N) algorithm and we will traverse all element which can be used as the start index and find the maximum subarray. For each iteration, we will memorize its start index, end index and the sum. When the value of sum is larger than the previous sum value, the start index, end index and the sum will be update. And the final result will be the maximum subarray and its sum.

2) Source code and screenshot execution results

Please check the source code in mss_linear.py

Input:

```
Enter the list of numbers: 1 -4 3 -4
```

Test case1:

Index	0	1	2	3
Arr[i]	1	-4	3	-4

Result:

```
Enter the list of numbers: 1 -4 3 -4
The maximum subarray starts at index 2, ends at index 2 and has sum 3.0.
```

Test case2:

Index	0	1	2	3	4	5
Arr[i]	1	-4	3	4	-2	6

Result:

```
Enter the list of numbers: 1 -4 3 4 -2 6
The maximum subarray starts at index 2, ends at index 5 and has sum 11.0.
```

3) Correctness Analysis

Invariant: At the start of each iteration of the for loop, the start index and the end index can always represent the maximum subarray we got so far. And the value of maximum sum can always be the maximum we got so far.

Initialization: Before the first iteration, the start and end index are 0(the first index of the array) and the maximum sum is the value of the first element in the array. This is the maximum subarray and its sum we can got so far which is satisfied the Invariant.

Maintenance: During and iteration, the start and end index of the maximum subarray and the maximum value will be update if the algorithm finds that there is a sum value is larger than previous sum value. Hence before an iteration of the loop, the start/end index and sum are always remains the invariant.

Termination: When the loop terminates, we will return the start and the end index of the maximum subarray and its sum. Because of the algorithm, those values will be updated or kept same when there is no larger sum value in this iteration. Hence the invariant will still remain.