Problem Set 2, Problem 2 — **Jihong Tang**

COMPSCI 260 — NetID: jt290

Work Date: 09/23/2018 — Due Date: Fri 09/28/2018 5pm

Introduction to the problem

Problem 2 from problem set 2 is focused on the maximum subarray problem based on the background as alien invasion.

For question 2a, we need to solve this problem with the brute force algorithm. Before the implementing of the brute force algorithm, we need to consider ways to speed up the brute force process. Also, the analysis of the algorithm is necessary after the implement.

For question 2b and 2c, we need to use divide and conquer and given linear algorithm to solve the problem, implement it in python and do some algorithm analysis.

For question 2d, we need to use the given module to evaluate and estimate the running time of our programs written in different algorithms based on sets of random inputs of length of given size.

Problem2a

For using brute force algorithm to solve the maximum subarray problem, I first thought about calculating every single subarray that can be obtained from the given sequence to determine the optimal one, which can be worked out by three loops and the time complexity will be $\Theta(n^3)$. The algorithm pseudo-code can be found in Algorithm1.

Algorithm 1 $\Theta(n^3)$ Brute Force Algorithm for 2a

```
for i = 1 to n do
  for j= i to n do
    sum = 0
    for k = i to j do
      sum += sequence[k]
      if sum >max then
        max = sum
      end if
    end for
  end for
end for
return max
```

After the deep analyzing, I found the brute force can be improved if I start at all possible start sites to calculate and find the optimal max value. In this way, I need to check n start site, each check time will be $n, n-1, n-2 \cdots 2, 1$. Totally, for the worst case, the running time complexity will be like $\frac{n(n+1)}{2}$, which is definitely $\Theta(n^2)$.

To argue the $\Theta(n^2)$ algorithm is correct, I would make the statement that all possible subarries have been checked in this algorithm to find the optimal solution leading to the result that it is definitely correct though the running time may be a little bit long.

The code solution of the $\Theta(n^2)$ brute force algorithm can be found in **collective_similarity.py**, presented by the function **similarity_brute_force**.

Problem2b

Algorithm description

For question 2b, we need to work out a divide and conquer method to solve the same problem and implement it in python. Thanks to my roommate Shenghong Zhao, he offered me some ideas when I was struggling with the problem how to combine the right part and left part together in the divide and conquer algorithm. He is a transfer junior student at Duke, and he is not a student in the course COMPSCI 260.

As the algorithm, I will divide the given sequence into two parts called left part and right part respectively every time, and do three things:

1. Get the maximum subarray in the left and the right part respectively in a recursive way.
2. Work out the maximum subarray including the mid position.
3. Return the maximum subarray between the three arrays found above.

The base case of this divide and conquer algorithm is when there is only one element in the left part or right part. In this case, just return the only element value. The recursive step is in the process to find the maximum subarray in the left part and the right part. Also, return the maximum subarray to solve the subproblems.

In order to work out the maximum subarray including the mid position in each subproblems, I proposed to take the following steps:

1. Find the maximum subarray ended with mid position in the left part.
2. Find the maximum subarray began with mid position in the right part.
3. Add the value found in the above steps to get the maximum subarray value.

Algorithm analysis

I believe I have done the job to argue my algorithm is correct in the algorithm description part since I demonstrated many details about the algorithm. I have pointed out the base case and the recursive steps, which are two curcial part in a normal divide and conquer algorithm.

The worst case of this algorithm will be dividing the given array into $\Theta(\log_2 n)$ levels. At each level, the total work needed to be done can be added to $\Theta(n)$. Therefore,the time complexity would be $\Theta(n \log n)$ in the worst case for the divide and conquer algorithm.

The code solution of the $\Theta(n \log n)$ divide and conquer algorithm can be found in **collective_similarity.py**, presented by the function **similarity_divide_conquer**.

Problem2c

Question solution

For the question asking me to explain how to update the two given parameters to maintain their meanings throughout the scan and to offer some explainations about the correctness of such algorithm, I would make the following statements.

1. The parameter MAX_SO_FAR is easy to update, just check if the parameter MAX_INCLUDING_HERE is larger than the MAX_SO_FAR and update to the MAX_INCLUDING_HERE value when this is true at each position during the scannig process.

2. To update the parameter MAX_INCLUDING_HERE at each position, we need to tell if the last position MAX_INCLUDING_HERE value is positive. If it is positive, we could add the now position value to the last position MAX_INCLUDING_HERE to update the parameter. However, if it is negative, we could just update the parameter to the now position value since the negative value will definitely decreases the sum value.

3. This method is to solve subproblems in a iterative way and I think it is one kind of dynamic programming algorithm. In order to find the optimal solution in the given array, we created many subproblems to solve in the scanning process in this method. The parameter MAX_SO_FAR is the optimal solution for each subproblem, and the scanning process will not miss any situations or subproblems, therefore, the method will guarantee us find the optimal region.

Algorithm analysis

This algorithm is definitely a linear algorithm, the worst case will just be scanning the given array from the first position to the last position and do some comparing and updating work at each position. Therefore, the time complexity of this algorithm is $\Theta(n)$.

The code solution of the $\Theta(n)$ linear algorithm can be found in **collective_similarity.py**, presented by the function **similarity_linear**.

Problem2d

Brute force algorithm runtime

For brute force algorithm, the running times on sets of random inputs of different length can be shown as Table1.

Table 1: Brute force algorithm evaluating running time

Length of inputs	Running time(s)
10^2	0.000681
10^3	0.054552
10^4	5.725094
10^5	614.695245

Since the time complexity of brute force algorithm is $\Theta(n^2)$, I make the following estimates as shown in Table2:

Table 2: Brute force algorithm estimating running time

Length of inputs	Running time(s)
10^6	6×10^4
10^7	6×10^6
10^8	6×10^8
10^9	6×10^{10}

Divide and conquer algorithm runtime

For divide and conquer algorithm, the running times on sets of random inputs of different length can be shown as Table3

Table 3: Divide and conquer algorithm evaluating running time

Length of inputs	Running time(s)
10^2	0.000796
10^3	0.006518
10^4	0.058135
10^5	0.619473
10^6	5.850749
10^7	72.387909

Since the time complexity of divide and conquer algorithm is $\Theta(n \log n)$, I make the following estimates as shown in Table4:

Table 4: Divide and conquer algorithm estimating running time

Length of inputs	Running time(s)
10^8	800
10^9	9000

Linear algorithm runtime

For linear algorithm, the running times on sets of random inputs of different length can be shown as Table5

Table 5: Linear algorithm evaluating running time

Length of inputs	Running time(s)
10^2	0.000063
10^3	0.000325
10^4	0.003326
10^5	0.033505
10^6	0.358472
10^7	4.354069
10^8	64.660207

Since the time complexity of linear algorithm is $\Theta(n)$, I make the following estimates as shown in Table6:

Table 6: Linear algorithm estimating running time

Length of inputs	Running time(s)
10^9	600