CS341 – Algorithm

May 9, 2019

Bentley's Problem
What is Bentley's Problem?
Given $A[1 \cdots n]$, find $max_{1 \le i \le j \le n} \sum_{k=i}^{j} A[k]$ or return 0 if all elements of the array are negative.
There are four ways to solve this problem.

- The simplies way is compute all the possible arrays and selete the one with largest sum.
  This takes $O(n^3)$.

- The second way is able to increase the efficiency into $O(n^2)$.

- The third way is faster by using Divide-and-Conquer.
  This approach is using the idea of merge sort $O(nlogn)$.
  We divide the entire array into two equally-sized parts.
  The solution must either be entirely in the left part or entirely in the right part, or it must be crossing the partition line.
  Therefore we recusive call the function to find the maximum subarray for left part, say MAXL, and right part, say MAXR.
  To find the maximum subarray "going over the middle partition line", say MAXM. We can achieve this in linear time $\Theta(n)$.

  Explain in low level, each time we start from the partition index, i, and decrement it to get the maximum subarray from $A[0]$ to $A[i]$ and in the meantime, we can make another thread to increment the i to n, so that we find the maximum subarray from $A[i+1]$ to $A[n]$. Lastly, we add them together to get the MAXM.
  Since it goes through the entire array, it takes $\Theta(n)$.
  Then we are able to compare MAXM, MAXL and MAXR to get the maximum subarray.

Why Divide-and-Conquer algorithm has $O(nlogn)$ complexity?
Simply because each time we divide the array into two parts and do $\Theta(n)$ works, then we get the following formula:

$$T(n) = 2T(\frac{n}{2}) + cn$$

It is a simple recrusion. After simply, we get the running complexity is $O(nlogn)$.

- The fourth way is the fastest way that only takes $O(n)$!
  Consier let $maxsol(i)$ be the maximum solution for array $A[1 \cdots i]$ and $tail(i)$ is the maximum solution ending at position i.

  Notice that: $maxsol(i)$ does not necessary contains $A[i]$, however, $tail(i)$ must contains $A[i]$ as the definition states above.

  Consider the relationship between $maxsol(i)$ and $maxsol(i-1)$.

  $$maxsol(i) = max\{maxsol(i-1), tail(i)\}$$

  The algorithm is following:

Algorithm 1 The Fastest Algorithm

```
1: maxsol := 0;
2: tail := 0;
3: for i = 1 to n do
4:     tail := max(tail + A[i], 0);
5:     maxsol := max(maxsol, tail);
6: end for
```

It only goes through the entire array once, therefore the complexity of this algorithm is $O(n)$.