Assignment1

卫卓虹

2018/10/6

1 Problem One

1.1 Solution

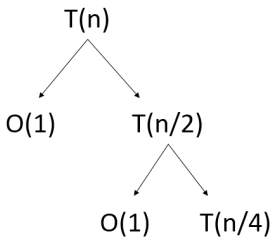
Given that the median of medians of two databases is the median of joint databases, we can use divide and conquer to design an algorithm whose inputs are two databases D_1 and D_2 and two pointers p_1 and p_2 which are both initialed as n/2 in the subset,which iteratively finds the medians of the subsets generated by pointers until one step where the set input is not dividable anymore.

Input: $database1, database2$

Output: $median$

1: **function** MEDIAN OF DATABASES(D_1, D_2)
2: $q_1 \leftarrow n/2$
3: $q_2 \leftarrow n/2$
4: **for** $i = 2$ to $logn$ **do**
5: $m_1 \leftarrow query(D_1, q_1)$
6: $m_2 \leftarrow query(D_2, q_2)$
7: **if** $m_1 > m_2$ **then**
8: $q_1 \leftarrow q_1 - n/2^i$
9: $q_2 \leftarrow q_2 + n/2^i$
10: **else**
11: $q_1 \leftarrow q_1 + n/2^i$
12: $q_2 \leftarrow q_2 - n/2^i$
13: **end if**
14: **end for**
15: **return** $min(m_1, m_2)$
16: **end function**

1.2 Subproblem Reduction Gragh



1.3 Correctness Validation

With the technique of loop-variant,we can prove it in following steps:

Initialization: At first step,$p_1 = p_2 = n/2$,they are both the median of their own subsets.

Maintenance: Suppose $m_1 > m_2$,since m_1 is the median of D_1,the global median should in the smaller binary part

of D_1 or bigger part of D_2.Then searching field is updated to D_1 and D_2,while p_1 and p_2 is also changed to their boundary.The searching field is updated to D_1^2 and D_2^2,while p_1 and p_2 is also changed to their boundary.During the for loop,loop invariant should hold and the algorithm is still searching in potential field.

Termination: At termination,i=logn , m_1 and m_2 is the nth and (n+1)th smallest point.According to the definition of median,we choose the smaller one of m_1 and m_2 as the final median.

1.4 Complexity Analysis

The time complexity is T(n) = 2T(n/2)+O(1)=O(logn)

2 Problem Five

2.1 Solution

The solution of this problem is equivalent to normal counting-inversion problem except for an extra judgement($a_i > 3a_j$). Thus similarly we use divide and conquer technique as follows:

Divide:Divide A into two arrays A[0...n/2-1]and A[n/2...n-1]; thus counting inversions within A[0...n/2-1] and A[n/2...n-1]constitutes two subproblems.

Conquer: Counting inversions within each half by calling countinginversion itself.

Combine:Note that elements in both halves are in increasing order.We can easily calculate the inversions in O(n) time.

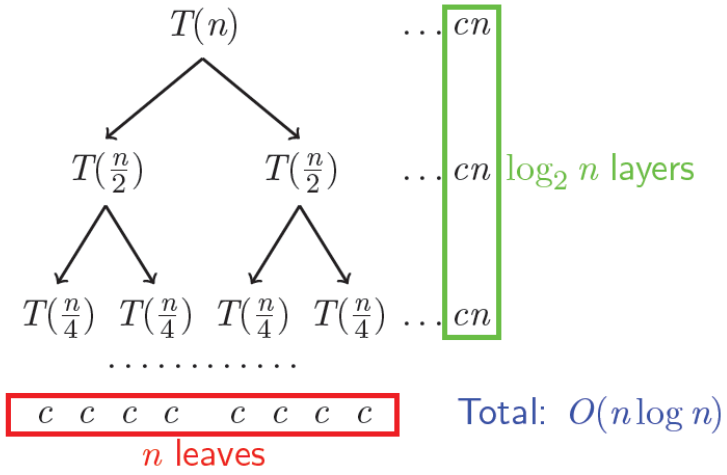
Input: *Array*

Output: *Number of Significant Inversions*

```
1:  function SIGNIFICANT-INVERSIONS(Array, left, right)
2:      result ← 0
3:      if left < right then
4:          middle ← (left + right)/2
5:          result ← result+ SIGNIFICANT-INVERSIONS(Array, left, middle)
6:          result ← result+ SIGNIFICANT-INVERSIONS(Array, middle, right)
7:          result ← result+ MID-INVERSION(Array, left, middle, right)
8:      end if
9:      return result
10: end function
11:
12: function MID-INVERSION(Array, left, middle, right)
13:     i ← left
14:     j ← middle
15:     k ← 0
16:     result ← 0
17:     while i < middle and j < right do
18:         if Array[i] < Array[j] then
19:             B[k++] ← Array[i++]
20:         else
21:             B[k++] ← Array[j++]
22:             if Array[i] > 3 * Array[j] then
23:                 result ← result + (j − k)
24:             end if
25:         end if
26:     end while
27:     while i < middle do
28:         B[k++] ← Array[i++]
29:     end while
30:     while j < right do
```

```
        B[k++] ← Array[j++]
    end while
    for i = 0 → k − 1 do
        Array[left + i] ← B[i]
    end for
    return result
end function
```

2.2 Subproblem Reduction Gragh



2.3 Correctness Validation

With the technique of loop-variant,we can prove it in following steps:

Initialization: When n=1,there is no significant inversions.

Maintenance: Assume that two subarray lists are all ordered and inversions of each list are calculated.Putting the first smaller number of two lists into the new array ensures that the new array is ordered too.We thus can calculate the inversions by adding the left number , the right number and the inversions between subarrays.The sum is the inversions of new arrays.

Termination: When combine the last two lists together,similarly we get the sum of inversions and that is the total number of significant inversions.

2.4 Complexity Analysis

Time complextity :T(n)=2T(n/2)+O(n)=O(nlogn)

3 Problem Three

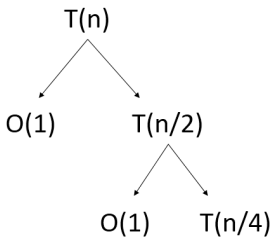
3.1 Solution

In this problem,we define probe() to get the value of one node.We recursively find the smallest one in its children and itself.If itself has the smallest value,then we return its value or we recursively compare the smallest child and its subtree.

Input: T
Output: *local minimum*

```
1: function FIND MIN(T)
2:     if min(probe(T.root(), probe(T.left()), probe(T.right()) == probe(T.root()) then
3:         return probe(T.root())
4:     else
5:         if probe(T.left()) > probe(T.right()) then
6:             return FIND MIN(T.left())
7:         else
8:             return FIND MIN(T.right())
9:         end if
10:    end if
11: end function
```

3.2 Subproblem Reduction Gragh



3.3 Correctness Validation

With the technique of loop-variant,we can prove it in following steps:

Initialization: At first step,the initial node we input is the root,we compare the value of it and its two children.The node we may return is the smallest of them.

Maintenance: Suppose the right child is smallest,we compare the values of this new node and its children, this process recursively goes forward until one node whose root has the smallest value.

Termination: At termination,since the root's value is the smallest,the local minimum is found and it is the smallest in the path through.

3.4 Complexity Analysis

The time complexity is T(n) = T(n/2)+O(1)=O(logn)