Algorithm Assignment5

Xinyu Jiang

October 2018

1

(25 pts) Solve the following recurrence relations. For each case, show your work.

1. $T(n) = T(n-1) + c\,n$ if $n > 1$, and $T(1) = c$,
2. $T(n) = 2T(n-1) + 1$ if $n > 1$, and $T(1) = 3$,
3. $T(n) = T(n/2) + 3$ if $n > 1$, and $T(1) = 2$,
4. $T(n) = T(n-1) + 2^n$ if $n > 1$, and $T(1) = 2$,
5. $T(n) = T(\sqrt{n}) + 1$ if $n > 2$, and $T(n) = 0$ otherwise.

a).
$T(n) = T(n-1) + cn$
$T(n-1) = T(n-2) + c(n-1)$
$T(n) = T(n-2) + c(n-1) + cn$
$T(n-2) = T(n-3) + c(n-2)$
$T(n) = T(n-3) + c(n-2) + c(n-1) + cn$
$T(n) = T(0) + c\Sigma_1^i n$
$T(n) = T(0) + c * \frac{(n+1)n}{2}$
Since $T(1) = c$, therefore, $T(n) = O(n^2)$
b).
$T(n) = 2T(n-1) + 1$
$T(n-1) = 2T(n-2) + 1$
$T(n) = 2 * (2T(n-1) + 1) + 1$
$T(n-2) = 2T(n-3) + 1$
$T(n) = 2 * (2(T(n-3) + 1) + 1) + 1$
$T(n) = (2^k)T(n-k) + \Sigma_1^n 1$
$T(n) = (2^k)T(0) + n$
$T(n) = 2^k n + n$
Since T(1)=3, $3 = 2^k T(0) + 1$, therefore, $T(n) = O(2^n)$
c).
$T(n) = T(n/2) + 3$
$T(n/2) = T(n/4) + 3$
$T(n) = T(n/4) + 3 + 3$

$T(n/4) = T(n/8) + 3$
$T(n) = T(n/8) + 3 + 3 + 3$
Pattern: $T(n) = T(n/2^k) + 3k$
Since $T(1) = 2$, so $n/2^k = 1k = log_2 n$
$T(n) = T(1) + 3log_2 n = 1 + 3logn$, therefore,
T(n)=$logn$.
d).
$T(n) = T(n-1) + 2^n$
$T(n-1) = T(n-2) + 2^{n-1}$
$T(n) = (T(n-2) + 2^{n-1}) + 2^n$
$T(n-2) = T(n-3) + 2^{n-2}$
therefore $T(n) = ((T(0) + 2^{n-1}) + 2^{n-2} + ...) + 2^n$
$T(n) = T(0) + \Sigma_1^n 2^n$
Since $T(1) = 3$, $T(1) = T(0) + 2^1$, so $T(n) = O(2^n)$
e).
$T(n) = T(\sqrt{n}) + 1$
$T(n) = T(n^{\frac{1}{2}}) + 1$
$T(n^{\frac{1}{2}}) = T(n^{\frac{1}{4}}) + 1$
$T(n) = T(n^{\frac{1}{4}}) + 2$
$T(n^{\frac{1}{4}}) = T(n^{\frac{1}{8}}) + 1$
$T(n) = T(n^{\frac{1}{8}}) + 3$
Pattern: $T(n) = T(n^{\frac{1}{2^k}}) + k$
For some term $T(x) = T(2) + a$, so $n^{\frac{1}{2^k}}$=2, $2^{2k} = 2$, therefore $k = log_2(log_2 n)$
Since $T(n) = 0$, so $T(n) = T(2) + log_2(log_2 n)$, so that $T(n) = O(loglogn)$

2

(10 pts) Consider the following function:

```
def foo(n) {
     if (n > 1) {
        print( 'hey' 'hey' )
        foo(n/4)
        foo(n/4)
}}
```

In terms of the input n, determine how many times is 'hey' printed. Write down a recurrence and solve using the Master method.

Base on the function above, the run time
$T(n) = 2T(n/4) + 2$
Base on Master Theorem:
Since a=2 b=4 c=0
so that the run time
$T(n)$=$O(n^(log_2 4))$=$O(n^{\frac{1}{2}})$

3

(30 pts) Consider a valleyed array $A[1, 2, \cdots, n]$ with the property that the subarray $A[1..i]$ has the property that $A[j] > A[j + 1]$ for $1 \leq j < i$, and the subarray $A[i..n]$ has the property that $A[j] < A[j + 1]$ for $i \leq j < n$. For example, $A = [16, 15, 10, 9, 7, 3, 6, 8, 17, 23]$ is a valleyed array.

1. Write a recursive algorithm that takes asymptotically sub-linear time to find the minimum element of A.
2. Prove that your algorithm is correct. (Hint: prove that your algorithm's correctness follows from the correctness of another correct algorithm we already know.)
3. Now consider the *multi-valleyed* generalization, in which the array contains k valleys, i.e., it contains k subarrays, each of which is itself a valleyed array. Let $k = 2$ and prove that your algorithm can fail on such an input.
4. Suppose that $k = 2$ and we can guarantee that neither valley is closer than $n/4$ positions to the middle of the array, and that the "joining point" of the two singly-valleyed subarrays lays in the middle half of the array. Now write an algorithm that returns the minimum element of A in sublinear time. Prove that your algorithm is correct, give a recurrence relation for its running time, and solve for its asymptotic behavior.

a).

```
findMin(int arr[],int startIndex, int endIndex){
    int index=(startIndex+endIndex)/2
    if(arr[index-1]>arr[i] && arr[index+1]<arr[index]){
        finMin(arr,index+1,endIndex);
    }else if(arr[index-1]<arr[index] && arrr[index+1]>arr[index]){
        findMin(arr,startIndex,index-1);
    }else{
        return arr[index];
        //the number before and after are both smaller then arr[index]
    }
}
```

Base on the function above, $T(n) = 2T(n/2) + 1$
Base on the run-time analysis in 1c, $T(n) = O(logn)$
b).
Prove:
Loop Invariant: For all elements in the array, the target value is in the array (starts from startIndex, and end at endIndex)
Initialization: The startIndex=0 and the endIndex=arr.length-1, which is the whole array, so the min value is for sure in the array
Maintenance: For each iteration of the loop, because the property of the valleyed

array, the length of array is half by shrinking by modify the index of the array, and the target value is still in the array.
Termination: If the value we find is both smaller than the value arr[index-1] and greater than arr[index+1], then by the definition of valleyed array, it is the minimum value of the array.
c). For multi-valleyed array, the algorithm falls because it is possible that the value find is the the local min value, not the minimum value of whole array. For example, $A = [16, 15, 10, 9, 7, 3, 6, 8, 17, 23, 2, 4]$, in this case, the value returns by the algorithm above is the local minimum which is 3, not the minimum of the whole array which is 2.
d).

```
findMin(int arr[],int startIndex, int endIndex){
    firstIndex=endIndex*(1/4);
    secondIndex=endIndex*(3/4);
    if(arr[firstIndex-1]>arr[firstIndex] && arr[firstIndex+1]<arr[firstIndex]){
        finMin(arr,firstIndex+1,firstIndex);
    }else if(arr[firstIndex-1]<arr[firstIndex] && arrr[firstIndex+1]>arr[firstIndex]
        findMin(arr, firstIndex,firstIndex-1);
    }else{
        int firstMin=arr[firstIndex];
    }
    if(arr[secondIndex-1]>arr[secondIndex] && arr[secondIndex+1]<arr[secondIndex]){
        finMin(arr,secondIndex+1,secondIndex);
    }else if(arr[secondIndex-1]<arr[secondIndex] && arrr[secondIndex+1]>arr[secondIn
        findMin(arr, secondIndex,secondIndex-1);
    }else{
        int secondMin=arr[secondIndex];
    }
    if(firstMin<secondMin){
        return firstMin;
    }else{
        return secondMin;
    }
}
```

Proof:
Loop Invariant: For all elements in the array, the two target values(two local min) are in the two sub-array.
Initialization: The startIndex=0 and the endIndex=arr.length-1, which is the whole array, so the two min values are for sure in the array
Maintenance:For each iteration of the loop, because the property of the valleyed array, the length of array is half by shrinking by modify the index of the array, and the target values are still in the array.
Termination: If the values we find are both smaller than the value arr[index-1]

and greater than arr[index+1](which are the two local minmum), then by the definition of valleyed array, they are the local minimum values of the array, then compare the value, return the smaller one.
Base on the algorithm
$T(n) = T(n/2) + 1$
$T(n/2) = T(n/4) + 1$
$T(n) = T(n/4) + 2$
$T(n/4) = T(n/8) + 1$
$T(n) = T(n/8) + 3$
Pattern: $T(n) = T(\frac{n}{2^k}) + k$,therefore the run time $T(n) = O(logn)$