Problem Set 2

2021年9月19号

Problem 1

(a)

$REVERSE(head)$

$next = NULL$

while $head \ != \ NULL$

$tmp \ = \ head \rightarrow next$

$head \rightarrow \ next \ = \ next$

$next = head$

$head = tmp$

$T(n) = \Theta(n)$

(b)

Gerneral Explanation:

因为$x.np$被定义为$x.prev \bigoplus x.next$，所以如果知道$x.prev$或$x.next$中的一个就可以知道另一个。所以建立一个列表以后，由于已知头节点$head.prev = 0$，所以$head.np = head.prev \bigoplus head.next = head.next$，即第二个节点的地址，所以当我们知道头节点的地址Head时，可以根据第二个节点的np知道第三个节点的位置，从而可以访问整个链表。

若有链表A，由头节点的指针可以访问A的每个节点，即可以知道所有节点的指针。要把新元素x插入，先把x存放到一个新的node中，然后根据前一个节点的指针和后一个节点的指针确定x.np，但是要注意此时前后的节点的np都会改变。

要删除第i个节点，只要断开它与前后节点之间的联系，修改前后节点的np。

Pseudocode:

$INSERT(x, i)$

$New\ node(X)$

$X.value = x$

$X.np = A_{i-1}.pointer \bigoplus A_i.pointer$

$A_{i-1}.np = A_{i-2}.pointer \bigoplus x.pointer$

$A_i.np = A_{i-1}.pointer \bigoplus X.np$

$DELETE(i)$

$A_{i-1}.np = A_{i-2}.pointer \bigoplus A_{i+1}.pointer$

$A_{i+1}.np = A_{i-1}.pointer \bigoplus A_{i+2}.pointer$

Problem 2

Brief Overview:

有两个栈A，B，B用来存放出现的最大数字，A用来存放所有的数字。

执行push时，先将要添加的数字与栈B的顶部的元素做比较，如果大于栈顶元素就把其压入栈B中，再压入栈A中，如果小于等于就直接压入栈A中。一开始栈B为空，将新添加的元素压入两个栈中。

执行pop时，不能直接只在栈A中操作，B中的相应元素也要移除。

执行max时，return栈B的栈顶元素。

Pseudocode:

$PUSH(x)$

if $B\ is\ not\ empty$

  if $x > B.top$

    $B.push(x)$

    $A.push(x)$

  else

    $A.push(x)$

  end if

else

$B.push(x)$

$A.push(x)$

end if

$POP()$

if $B\ is\ not\ empty\ and\ A.top == B.top$

$B.pop()$

$res = A.pop()$

return res

$MAX()$

return $B.top$

Space Complexity:
$S(n) = O(1)$

Problem 3

Brief Overview:

用一个栈A来保存符号。

如果输入的是一个操作数，则直接输出。

如果输入是一个运算符号，当A中为空时压入A，若A不为空，则比较与A中元素的优先级，拿出A中所有优先级更高或者优先级相同的元素输出，再将符号压入A中。

当操作数全部输出后，再按次序取出A中的元素输出。

Pseudocode:

for $i = 1\ to\ n$	$c_1 * (n+1)$
$x = input()$	$c_2 * n$
if $x\ is\ a\ operand$ **or** $x ==" ! "$	$c_3 * \sum t_3$
$print(x)$	
else	
if $A\ is\ not\ empty$	
if $x =='' +''$	
do $print(A.pop)$ **untill** $A\ is\ empty$	
else	
do $print(A.top)$ **until** $A.top! ='' *''$	
$A.push(x)$	
while $A\ is\ not\ empty$	
$print(A.pop())$	

Time Complexity:

$Best\ case$：当表达式只做了阶乘运算的时候，$T(n) = c_1 * (n+1) + c_2 * n + c_3 * n + c_4 * n = \Theta(n)$

$Worst\ case$：当表达式做了三种运算，

$T(n) = \Theta(n)$

Problem 4

Algorithm A:

$\because A$采用了分治策略将问题划分成五个子问题且每个问题的大小为$\frac{n}{2}$，且最后以$O(n)$的时间复杂度结合

$\therefore T(n) = 5T(\frac{n}{2}) + O(n)$

$T(n)$可以表示成$T(n) = aT(\frac{n}{b}) + O(n^c)$

$a = 5, b = 2, c = 1$

根据主定理，$c < log_b a$，所以得出：

$T(n)=O(n^{log_b a})=O(n^{log_2 5})$

Algorithm B:

$\because B$将问题分成大小为$(n-1)$的两个子问题，且最后以$O(1)$的时间复杂度结合

$\therefore T(n)=2T(n-1)+O(1)$

画出递归树解决

$T(n)=2*2T(n-2)+2*O(1)=2^2T(n-2)+2*O(1)$

$\cdots$

$T(n)$可以写成$T(n)=2^{n-1}T(0)=O(2^n)$

Algorithm C:

$\because C$将问题分成9个大小为$\frac{n}{3}$的问题

$\therefore T(n)=9T(\frac{n}{3})+O(n^2)$

若表示为$T(n)=aT(\frac{n}{b})+O(n^2)$

$a=9,b=3,c=2$

根据主定理：

$T(n)=O(n^{log_b a}lgn)=O(n^2lgn)$

经过比较后，选择C，因为效率更高。

Problem 5:

Brief Overview:

采用分而治之的策略，多次将数组一分为二直至只有一个元素，然后两两比较删去相同数字形成新数组，递归得出整个原数组删除重复元素的结果。

Pseudocode:

$MergeDelete(A[1\cdots n])$

if $n==1$

  return $A[1\cdots n]$

else

  $Left=MergeDelete[1\cdots 2/n]$

  $Right=MergeDelete[2/n+1\cdots n]$

  return $NewMerge(Left,Right)$

$NewMerge(A,B)$

$new\ array\ S[1\cdots k]$

$i=j=0$

for $p=0\ to\ k-1$

  if $A[i]==B[j]$

    $S.add(A[i])$

    $B[j]=\infty$

    $i+1$

  else $j+1$

for $q=0\ to\ B.length-1$

  if $B[q]!=\infty$

    $S.add(B[q])$

  return S

Time Complexity:

$n=1$时$T(1)=O(1)$

$n!=1$时$T(n)=2\cdot T(n/2)+O(n)$

故总的$T(n)=O(nlgn)$

Correctness:

找出循环不变式：数组中没有重复的数字。

$Basis$: 当$n=1$时，显然没有重复元素，故成立。

$InductiveSteps$: 假设$n=k$时不变式也成立，所以第k次循环结束后，数组内已经没有重复的数字

当$n=k+1$时，新添加的数字在如果与已有数字重复，在比较过程中过就会被删去，所以不变式依然成立。

所以算法的正确性可证。

Problem 6

(a)

$(2,1),(3,1),(8,1),(6,1),(8,6)$

(b)

结论：插入排序的第二层循环就是找逆序对的一个过程，所以逆序对的个数和插入排序的时间复杂度的数量级应该是相同的，但是相比插入排序还要改变元素

证明：

找逆序对的伪代码：

$count=0$

for $j=2\ to\ A.length$

  $key=A[j]$

  $i=j-1$

  while $(i>0\ and\ A[j]>key)$

    $i=i-1$

    $count=count+1$

return $count$

假设$D(i)$是固定i之后找到的逆序对的个数

$\therefore$ 总的逆序对的个数是 $\sum_{i=1}^{n}D(i)$

而对于插入排序的第一层循环，每次取一个i值，时间复杂度就为$D(i)+T(n),T(n)$为进行位置互换及增值操作需要的时间

$\therefore$ 总的时间为 $\sum_{i=1}^{n}D(i)+T(n)$

所以两值的数量级是相等的

(c)

Brief Overview:

利用归并排序（时间复杂度为$O(nlgn)$, 在$Merge$函数上做一些修改。在归并排序的$Merge$操作中，每一次将左右两个数组扫描重组，左边的元素的下标一定

Pseudocode:

$Inversion(A[1\cdots n])$

if $n==2$

  if $A[0]>A[1]$

    $count=1$ **else** $count=0$

else

  $l=Inversion(A[1\cdots n/2])$

  $r=Inversion(a[n/2+1\cdots n])$

  $L=A[1\cdots n/2]$

  $R=A[n/2+1\cdots n]$

  $count=NewMerge(L,R)+l+r$

  end if

$NewMerge(A,B)$

$new\ array\ S[1\cdots A.length+B.length]$

$count=0$

$i=j=0$

for $k=0\ to\ A.length$

  if $A[i]<=B[j]$

    $S[k]=A[j]$

    $i=i+1$

else

$S[k] = B[j]$

$j = j + 1$

$count = count + 1$

return $count$