Algorithmn HW5

5140379032 JIN YI FAN

Problem 4.5

Require: a given array $A[1 \dots n]$
Ensure: whether this is a heap

1: **function** isHeap($A[], n$)
2:     **if** $A[0] > A[1]$ and $A[0] > A[2]$ **then**
3:         **return** isMaxHeap(A, n)
4:     **else if** $A[0] < A[1]$ and $A[0] < A[2]$ **then**
5:         **return** isMinHeap(A, n)
6:     **elsereturn** false
7:     **end if**
8: **end function**
9: **function** isMaxHeap($A[], n$)
10:     **for** $i \leftarrow 1$ to $\lfloor n/2 \rfloor$ **do**
11:         **if** $k[i] < k[2i+1]$ **then**
12:             **return** false
13:         **end if**
14:         **if** $2i + 2 < size$ and $k[i] < k[2i+2]$ **then**
15:             **return** false
16:         **end if**
17:         **return** true
18:     **end for**
19: **end function**
20: **function** isMinHeap($A[], n$)
21:     basically the same as isMaxHeap
22:     change the "$k[i] <$" in line 11 and 14 to "$k[i] >$"
23: **end function**
24: **return** isHeap(A,n) ▷ main function

The time complexity is $O(n)$

Problem 4.9

Require: an array $A[1 \dots n]$ of a maxHeap
Ensure: the *minimumkey* in A

1: $lstart \leftarrow \lfloor n/2 \rfloor$ ▷ The start of leaf nodes
2: $Leaves[] \leftarrow A[lstart : n-1]$ ▷ pick all the leaves
3: **for** $i \leftarrow lstart$ to n **do**
4:     $res \leftarrow res > A[i]\ ?\ A[i] : res$
5: **end for**

Requires $\lfloor n/2 \rfloor$ comparisons in total. So the algorithmn is $\Theta(n)$

Problem 4.19

Require: two heaps $A[1 \dots n]$ and $B[1 \dots n]$
Ensure: merge B to A

pick one node from B sequentially ▷ n times
insert it to A ▷ need $\log(n)$

The complexity is $O(n \log(n))$

k-merge

Require: k sorted lists $L_1, L_2 \dots L_k$
Ensure: merged one sorted list

$Min[] \leftarrow L_1[0], L_2[0] \dots L_k[0]$ ▷ every element has an index label showing where it comes from
MakeMinHeap(Min) ▷ takes $O(k)$
while lists not all empty **do** ▷ loop n times
    remove($L[] \leftarrow$ the minimum element in the heap)
    insert((next element exist)? next element : first element in next list) ▷ takes $O(\log(k))$
end while

So it takes $O(k) + n \cdot O(\log(k)) \; = \; O(n \log(k))$

Dynamic median

keep a maxHeap and a minHeap

1 Insertion

Each insertion, insert the element in both maxHeap and minHeap
After insertion, adjust heaps if one heap is 2-element larger than the other by moving the top element of larger heap to smaller heap
So it takes $O(\log(n))$ which is the cost of insertion in heap

2 Find

If maxHeap and minHeap are of same size, return $(maxHeap.top + minHeap.top)/2$
Else return the top of the larger heap
So this takes $O(1)$

3 Remove

Remove the median while keep the feature heap
This takes $O(\log(n))$