Deepanshu 2019CS50427

Page :
Date :

Homework 13.1

Data Structure	Insert	Minimum	Delete-Minimum
Array	O(n)	O(1)	O(n)
Singly linked list	O(n)	O(1)	O(1)
Stack	O(n)	O(1)	O(1)
Queue	O(n)	O(1)	O(1)
Hash Map	O(n)	O(1)	O(1)
BST	O(n)	O(n)	O(n)
AVL/2-4/R-B tree.	O(logn)	O(1)	O(logn)

Note that answer may vary depending on following points -

- Whether we keep a ~~to~~ pointer to last element in tail or not.
- Whether we are doing unsorted implementation or sorted implementation. (Above ~~~~ is done assuming sorted implementation).
- Whether to take ~~~~ height of BST as O(n) or O(logn).
  worst case
- for array case, whether we know expected size in advance or not.

Deepanshu 2019CS50427

Homework 13.2

Proof by induction

P(x): Procedure gives correct output if height is x.

Base case: $h = 0$

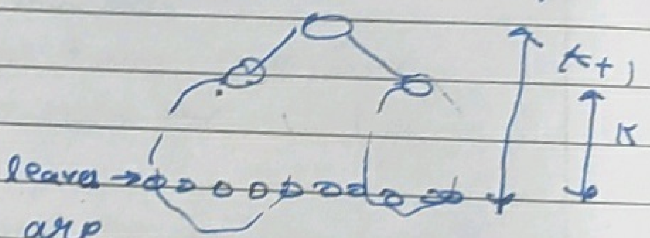
Only one element is present which is a heap.

Induction Hypothesis

Assume that P(k) is true i.e if height is k, the procedure described in lecture works correctly.

for k+1.

Now we start from leaves in the procedure and according to induction hypothesis, heaps will be formed up till height k.



Now the left subtree and right subtree of root are valid heaps as P(k) is true from induction hypothesis.

Now we call heapify(1) ~~and~~ ~~since~~ and since both left subtree and right subtree are heaps, heapify works correctly and finally, ~~PEI~~ heap of height k+1 is built.

Hence P(k+1) is true and by principle of Mathematical Induction, P(x) is true for all x.

Hence Proved.

Deepanshu 2019CS50427

Page :
Date :

Homework 13.3

Ans ⇒ Algorithm heapSort (ar[], n)

```
for( int i ← n/2 -1 ; i >= 0 ; i--)
    heapify (ar, i, n)

for( i ← n-1, i > 0, i--)
    swap(ar[0], ar[i])
    heapify( ar, 0, i)
```

[To rearrange elements in array so that it represents a heap n/2 to n are leaves, so ~~need~~ no need to heapify.]

Above loop puts element at last (effectively removing it from heap as we are decrementing i).
This is done to ensure in place sorting.
Note :- ~~finall~~ final array after this sorted but in reverse.
We can make it ascending order easily.

```
Algorithm heapify (ar[], i, n)
    int min ← ar[i]
    int l ← 2*i + 1
    int r ← 2*i + 2
    if (l<n && ar[l] < ar[min])
        min = l
    if (r<n && ar[r] < ar[min])
        min = r
```

```
Algorithm heapify (ar[], i, n)
    int min ← ar[i]
    int l ← 2i+1
    int r ← 2i+2
    while (True)
        if(
```

Deepanshu 2019CS50427

Page :
Date :

Iterative heapify

Algorithm heapify(ar[], i, n)
int min ← ar[i]
while (i<n)
if (ar[min] ≤ ar[2i+1] && ar[min] ≤ ar[2i+2])
break
else if (ar[min] > ar[2i+1])
swap(ar[min], ar[2i+1])
min = 2i+1
else
swap(ar[min], ar[2i+2])
min = 2i+2

Runtime analysis

- From lecture proof, we know that –

→ ~~building~~ building heap takes $O(n)$ time.

→ heapify takes time no more than $O(\log n)$.

Therefore building heap and the calling heapify repeatedly on all nodes will take.

$$O(n) + O(n\log n) \boxed{= O(n\log n)}$$