8/30/2015

Advance Algorithm HW

95/100

2.2 2 n = A.length
for(i=0; i<n; i++){
find min_value from A[i] to A[n-1]; // another for loop
swap the min_value with A[i]
}

This algorithm has a loop invariant in that after each loop A[i] will be the minimum value from A[i] ~~to~~ A[n-1]
It only needs to run to n-1 because at that point, because of the swapping, A[n] will automatically have the maximum value
Best case: $O(n^2)$
Worst case: $O(n^2)$

2.3 6 Using binary search instead of a linear search will NOT improve the overall worst-case running time of the insertion sort. The reason for this is that the while loop is doing a comparison and swapping and using a binary search will reduce the number of comparisons but will have the same number of swaps.

you need to list the indices for inversions

−5

2.4 a. Given <2, 3, 8, 6, 1>; its inversions are <2,1>; <3,1>; <8,6>; <8,1>; <6,1>
(1,5) (2,5) (3,4) (3,5) (4,5)

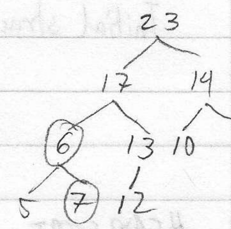
b. <n, n-1, n-2, ..., 1> is from the set <1, 2, ..., n> and has the most inversion.
It has $(n-1)+(n-2)+\ldots+1 = \frac{n(n-1)}{2}$ inversions

c. The relationship between the running time of insertion sort and the number of inversions in the input array is that they have a running time of O(n + number of inversions)
First the outer loop is executed n times b/c there are n elements, now each iteration of the inner loop eliminate one invertion, This means sorting the elements will eliminate all inversions thus having a running time of O(n + # of inversions)

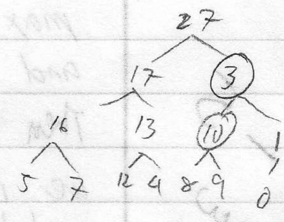
95/100

CMPE 130 HW 2

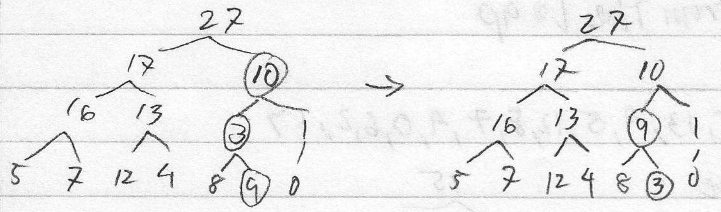
6.1 6. array <23, 17, 14, 6, 13, 10, 1, 5, 7, 12>
This is not max heap because the numbers 6 and 7 need to be interchanged to satisfy the heap property of parent > child



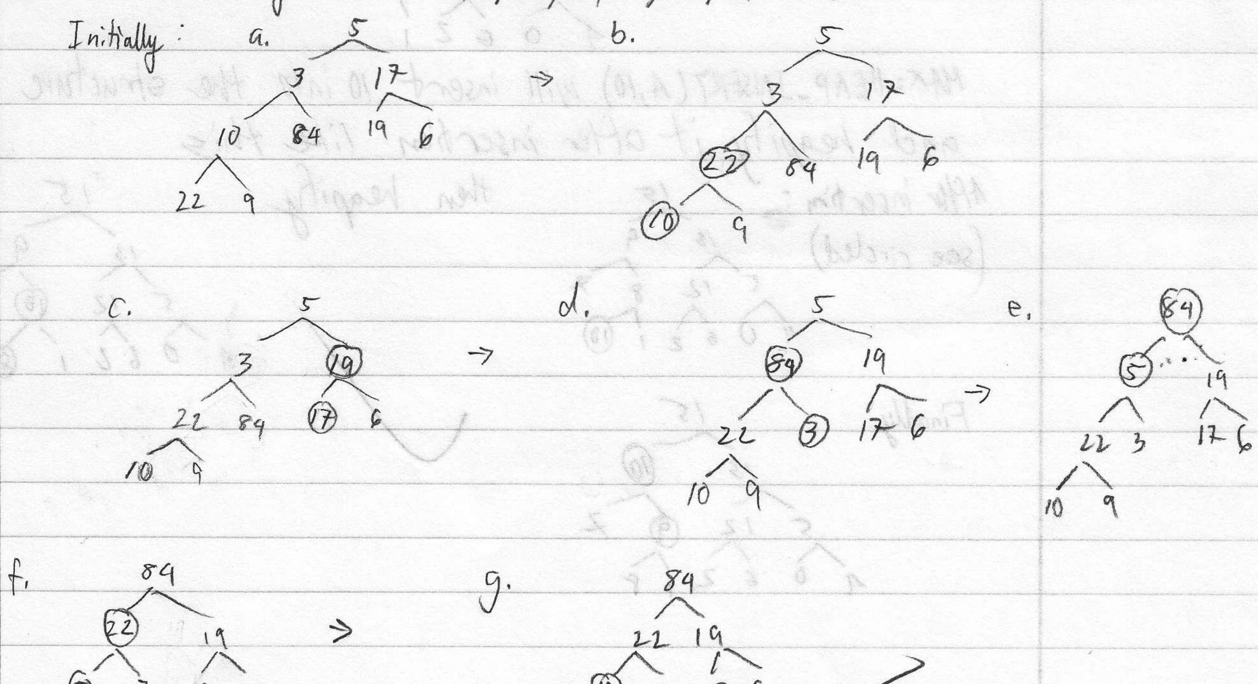
6.2 1. Given A = < 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0>
MAX HEAPIFY (A,3)
Originally the tree structure is like this →
MAX HEAPIFY (A,3) would rearrange the right subtree (see the circled elements)



When tracking for max heapify
Go from right most child to left
Right to left



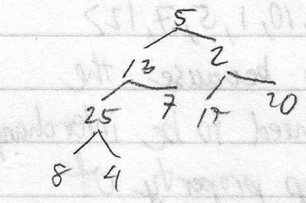
6.3 1. Given the array: A = < 5, 3, 17, 10, 84, 19, 6, 22, 9>



95/100

6.4 | 1. Given A = < 5, 13, 2, 25, 7, 17, 20, 8, 4 >

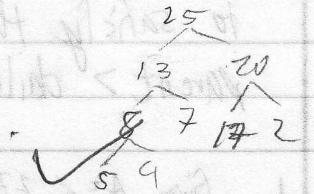
Initial structure :



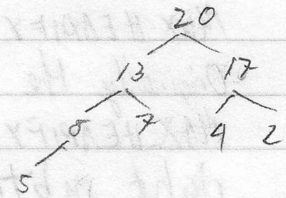
which is NOT a heap

Sorted array : 25, 20, 17, 13, 8, 7, 5, 4, 2

HEAP SORT would make a heap like this:



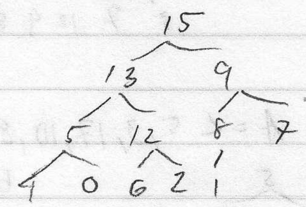
After creating the heap we take the max at the top which is 25 and replace with bottom element 4
Then we call heapify again
we repeat this process until our sorted array is filled with all the elements from the heap



-5 show some steps

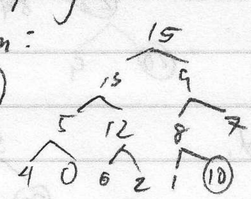
6.5 | 2. Given A = < 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 >

Initial Structure :

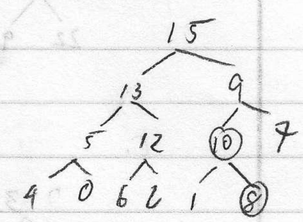


MAX-HEAP_INSERT(A,10) will insert 10 into the structure and heapify it after insertion like this

After insertion : (see circled)



then heapify



...

Finally