IMPORTANT

SORTING TECHNIQUES

WEEK-30 | 205-160
WEDNESDAY
24

* Insertion sort

Suppose an array A contain 8 elements

77, 33, 44, 11, 88, 22, 66, 55

Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1	$-\infty$	(77)	33	44	11	88	22	66	55
K=2	$-\infty$	77	(33)	44	11	88	22	66	55
K=3	$-\infty$	33	77	(44)	11	88	22	66	55
K=4	$-\infty$	33	44	77	(11)	88	22	66	55
K=5	$-\infty$	11	33	44	77	(88)	22	66	55
K=6	$-\infty$	11	33	44	77	88	(22)	66	55
K=7	$-\infty$	11	22	33	44	77	88	(66)	55
K=8	$-\infty$	11	22	33	44	66	77	88	(55)

Sorted list :- $-\infty$ 11 22 33 44 55 66 77 88

AUGUST '13
M T W T F S S
1 2 3 4

SEPTEMBER '13
M T W T F S S
30 1

THINGS TO DO

INSERTION (A,N)

This algorithm sorts the array A with N elements

(1) Set A[0] := -∞

(2) Repeat steps 3 to 5 for K = 2, 3, - - N

(3) Set TEMP := A[K] and PTR := K-1

(4) Repeat while TEMP < A[PTR]

a) set A[PTR+1] := A[PTR] [Moves element forward]

b) set PTR := PTR-1

[End of loop]

(5) Set A[PTR+1] := TEMP [Inserts element in proper place]

[End of step 2 loop]

(6) Exit

Complexity : $O(n^2)$

IMPORTANT

WEEK-30 | 207-158

FRIDAY

26

SELECTION SORT :-

Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1, LOC=4	(77)	33	44	(11)	88	22	66	55
K=2 LOC=6	11	(33)	44	77	88	(22)	66	55
K=3 LOC=6	11	22	(44)	77	88	(33)	66	55
K=4 LOC=6	11	22	33	(77)	88	(44)	66	55
K=5 LOC=8	11	22	33	44	(88)	77	66	(55)
K=6 LOC=7	11	22	33	44	55	(77)	(66)	88
K=7 LOC=7	11	22	33	44	55	66	(77)	88
Sorted list	11	22	33	44	55	66	77	88

Complexity = $O(n^2)$

THINGS TO DO

AUGUST '13

M	T	W	T	F	S	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18

SEPTEMBER '13

M	T	W	T	F	S	S
30						1
2	3	4	5	6	7	8

* SELECTION SORT

Procedure: MIN (A, K, N, LOC)

An array A is in memory. This procedure finds the location LOC of the smallest element among A[K], A[K+1], - - - A[N]

(1) Set MIN := A[K] and LOC := K [Initializes pointers]

(2) Repeat for J = K+1, K+2, - N.
If MIN > A[J], then set MIN := A[J], A[LOC] := A[J]
and LOC := J
[End of loop]

(3) Return.

Algorithm: SELECTION (A, N)
This algorithm sorts the array A with N elements.

(1) Repeat steps 2 and 3 for K = 1, 2, - - N-1,.

(2) Call MIN (A, K, N, LOC)

(3) [Interchange A[K] and A[LOC]
Set TEMP := A[K],
A[K] := A[LOC]
A[LOC] := TEMP.
[End of step 1 loop]

(4) Exit.

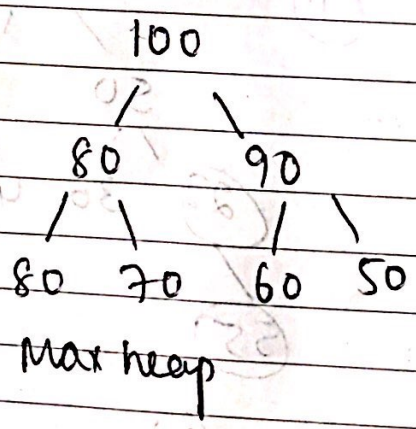
Complexity :- Worst case : $O(n^2)$
Average case : $O(n^2)$

CHAPTER : HEAPS

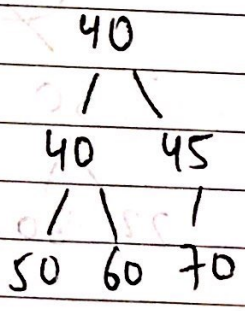
Suppose H is a complete binary-tree with n elements. Then H is called a heap or a maxheap, if each node N of H has the following property:

The value at N is greater than or equal to the value at each of the children of N.

Minheap :- The value at N is less than or equal to the value at any of the children of N



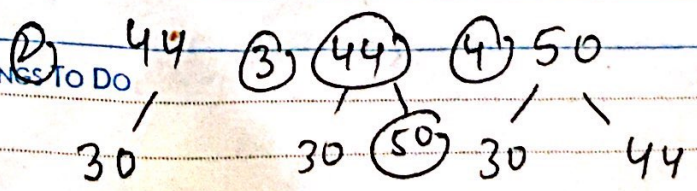
Max heap



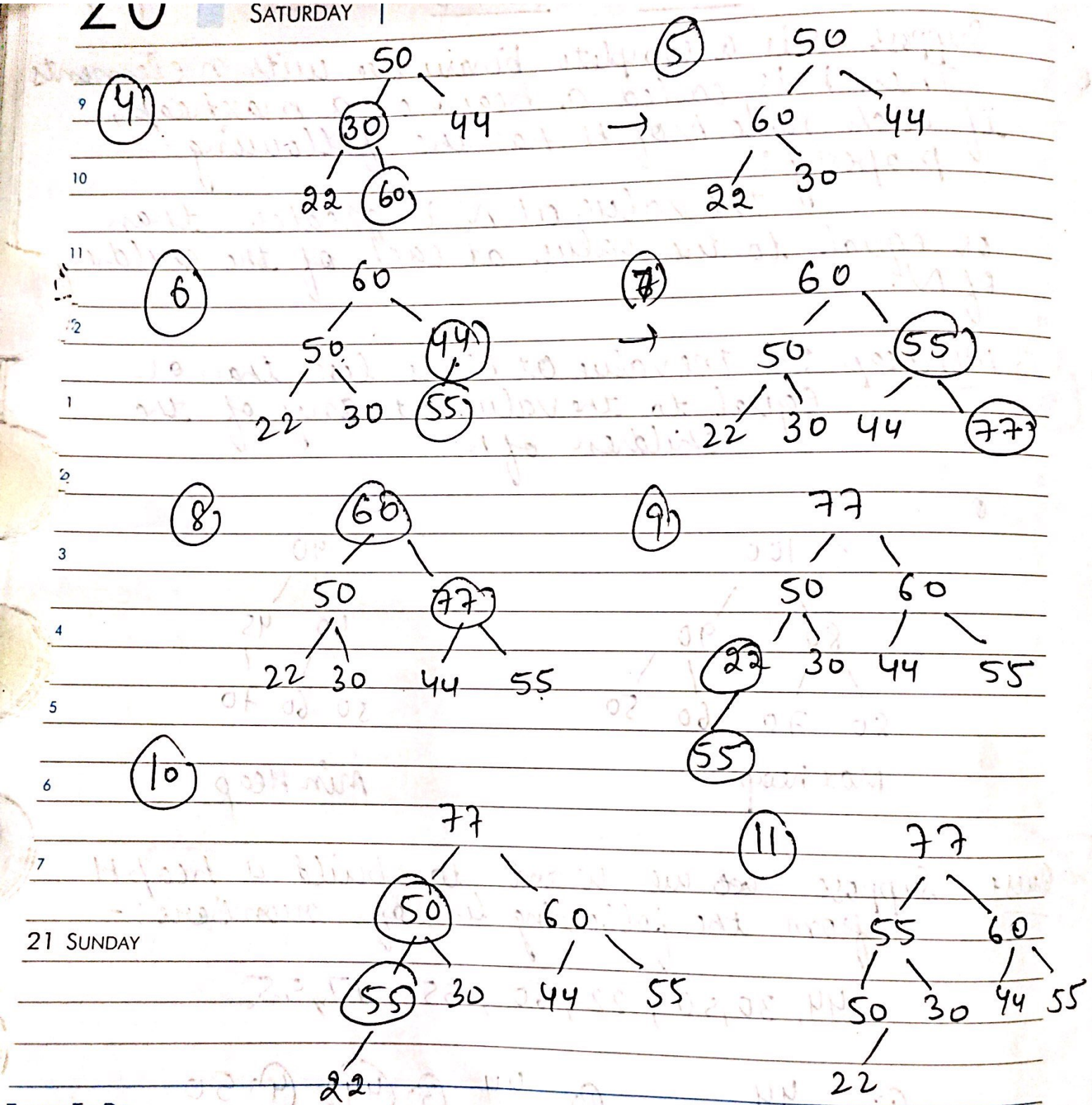
Min Heap

Ques: Suppose we want to build a heap H from the following list of numbers:-

44, 30, 50, 22, 60, 55, 77, 55



20 SATURDAY



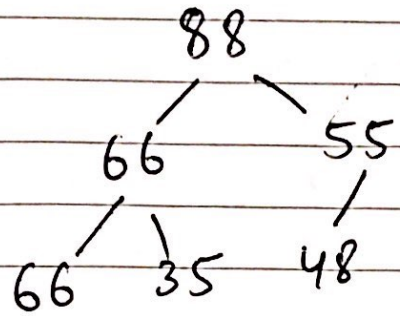
21 SUNDAY

THINGS TO DO

JUNE '13 M T W T F S S

JULY '13 M T W T F S S

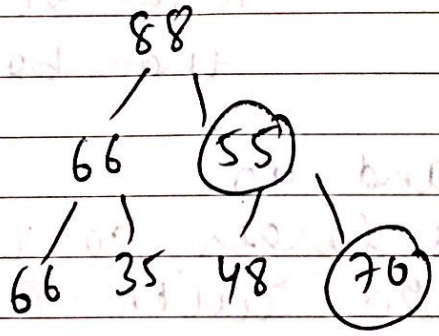
* Insertion into a heap :-



Insert 70 into a heap.

Initially insert 70 as the last element.

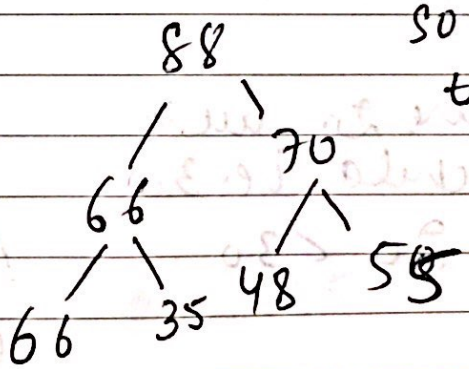
88	66	55	66	35	48	70
1	2	3	4	5	6	7



Compare 70 with its parent node 55

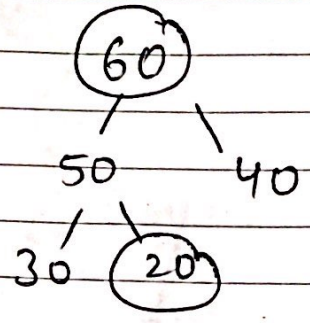
~~70~~ > 55

so interchange them to build maxheap

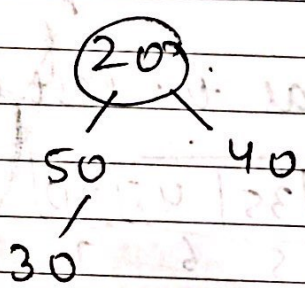


TUESDAY

Deleting the Root of a heap:-



Delete 60 and replace with 20 (last element)



Now compare 20 with 50 & 40

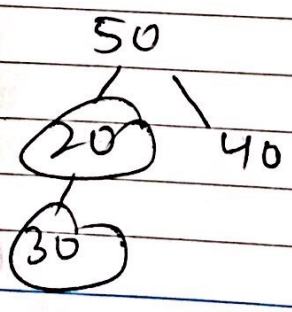
Now 20 is smaller than both of its child

Compare 50 and 40

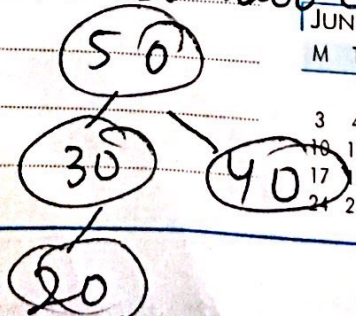
50 is larger so interchange 20 with 50.

Now compare 20 with its child i.e. 30

$20 < 30$



So interchange these two elements



Now heap

THINGS TO DO

Heap Sort

(1) Insertion into HEAP.

INSHEAP (TREE, N, ITEM)

A heap H with N elements is sorted in the array TREE, This procedure inserts ITEM as a new element of H. PTR gives the location of ITEM as it rises in the tree, and PAR denotes the location of the parent of ITEM.

(1) [Add new node to H and initialize PTR]
Set N := N+1 and PTR := N

(2) Repeat while PTR > 1
compare ITEM and TREE[PAR]
If ITEM ≤ TREE[PAR]
then Set TREE[PTR] := ITEM and Return.

(4) If ITEM ≤ TREE[PAR]
then set TREE[PTR] := ITEM and Return.

(3) Else
Set TREE[PTR] := TREE[PAR]
[Moves node down]

(4) Set PTR := PAR [Updates PTR]

(5) [Assign ITEM as the root of H]
Set TREE[1] := ITEM

(6) Exit.

(2) Deletion from Heap or Deleting Root node of Heap

DELHEAP (TREE, N, ITEM)

A heap H with N elements is stored in the array TREE. This procedure assigns the root TREE[1] of H to the variable ITEM and then reheaps the remaining elements. The variable LAST saves the value of the original last node of H. The pointers PTR, LEFT and RIGHT gives the locations of LAST and its left and right children as LAST sinks in the tree.

Step:1 Set ITEM := TREE[1] [Removes root of H]

Step:2 Set LAST := TREE[N] and N := N-1
[Removes last node of H]

Step:3 Set PTR := 1, LEFT := 2 and RIGHT := 3
[Initializes Pointers]

Step:4 Repeat steps 5 to 7 while RIGHT ≤ N.

Step:5 If LAST ≥ TREE[LEFT] and LAST ≥ TREE[RIGHT] then set TREE[PTR] := LAST and Return.
[End of If structure]

Step:6 If TREE[RIGHT] ≤ TREE[LEFT] then
Set TREE[PTR] := TREE[LEFT] and PTR := LEFT.

Else:

Set TREE[PTR] := TREE[RIGHT] and PTR := RIGHT

[End of if structure]

Step:7 Set LEFT := 2 * PTR and RIGHT := LEFT + 1

[End of step 4 loop]

Step:8 if LEFT = N and if LAST < TREE[LEFT] then Set PTR := LEFT.

Step:9 Set TREE[PTR] := LAST

Step:10 Return.

* Heapsort Algorithm

HEAPSORT (A, N)

An array A with N elements is given. This algorithm sorts the elements of A.

(1) [Build a heap H using insertion procedure]

Repeat for J = 1 to N-1

Call INSHEAP

(2) [Sort A by repeatedly deleting the root of H]

Repeat while N > 1

a) Call DELHEAP (A, N, ITEM)

b) Set A[N+1] := ITEM

[End of loop]

Step: 3 Exit.

Complexity of Heap Sort :- $O(n \cdot \log n)$