95/100

CMPE 130 Midterm Exam #1 Fall 2014

15:00—16:15 Thursday Oct 2, 2014

Student Name ______________________________(print)

Student ID__________________________________

(8 points) Problem 1 (A): Fill into line 6 and line 7 of the pseudo code in the **"while loop"** of INSERSION-SORT below. (The INSERSION-SORT sorts the numbers in the array in ascending order.)

INSERTION-SORT (A)

```
1   for j=2 to A.length
2   key=A[j]
3   //insert  A[j] into the sorted sequence A[1, …, j-1]//
4   i= j-1
5   while i>0 and A[i] > key
6       A[i+1] = A[i]
7       i = i-1
8   A[i+1] = key
```

8

(6 points) Problem 1(B): Apply INSERTION-SORT to array

31	41	59	26

and show the intermediate and final result for index j=2, 3 and 4.

6

J=2

1	2	3	4
31	41	59	26

J=3

31	41	59	26

J=4

26	31	41	59

31 and 41 are the only part of the array. It checks if 31 is larger than 41. Since it isn't, the numbers stay in that order. 59 is inserted, and checks if the number before it is larger, which it isn't so it stays in place. When 26 is inserted, it is compared with every number until the number before it is smaller, which in this case none are smaller, so it gets inserted in the beginning.

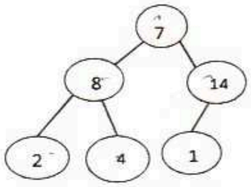
(6 points) Problem 1(C): Create an array out of the elements {41, 26, 59, 31} that has the worst performance under INSERTION-SORT.

6

Array that leads to the worst performance is

59	41	31	26

(6 points) Problem 2(A): Is the data structure below a MAX-HEAP

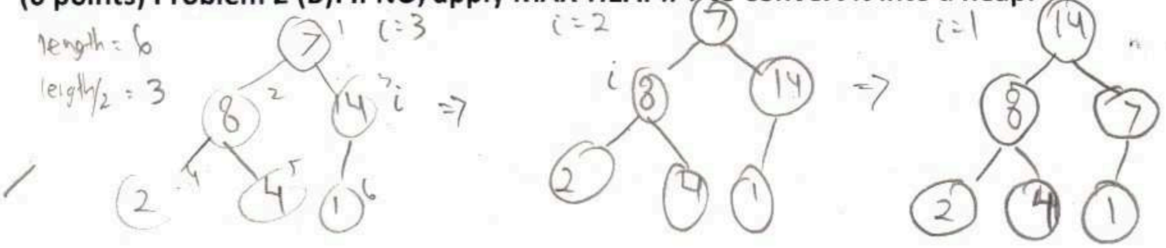


6 ✓

N̲o̲t̲ a max heap because parent node 1 is smaller than both child nodes

(6 points) Problem 2 (B): If NO, apply MAX-HEAPIFY to convert it into a heap.

6 ✓



switch node 3 with node 1

(8 points) Problem 2 (C): create an array corresponding to the result in 1(B)

8 ✓

14	8	7	2	4	1
1	2	3	4	5	6

20 ✓

(20 points) Problem 3: Using Attachment #1 as a model, illustrate the operation of HEAPSORT on the array A<5, 13, 2, 25, 7, 17, 20, 8, 4>

20 ✓

(20 points) Problem 4: Using Attachment #2 as a model, illustrate the operation of PARTITION on the array A<13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11>

Problem 5 The Knuth-Morris-Pratt algorithm is widely used in matching an m-element pattern to an n-element string.

(5 points) 5 (A): What is the complexity of the brute-force string matching algorithm? O(mn)

5 ✓

(5 points) 5 (B): The KMP algorithm reduces the complexity to O(nlogn) **by a pre-processing.** O(n)

0

(10 points) 5 (C): The pre-processing involves the use of prefix in the pattern. Compute the f (pre-processing function) of the KMP algorithm for the pattern

10 ✓

P:

A	T	C	A	C	A	T	C

Fill in the answer f( ) below.

0	0	0	1	0	1	2	3

A T C A C A T C

A T C A C A T C

A T C A C A T C

A T C A C A T C

A T C

Attachment #1

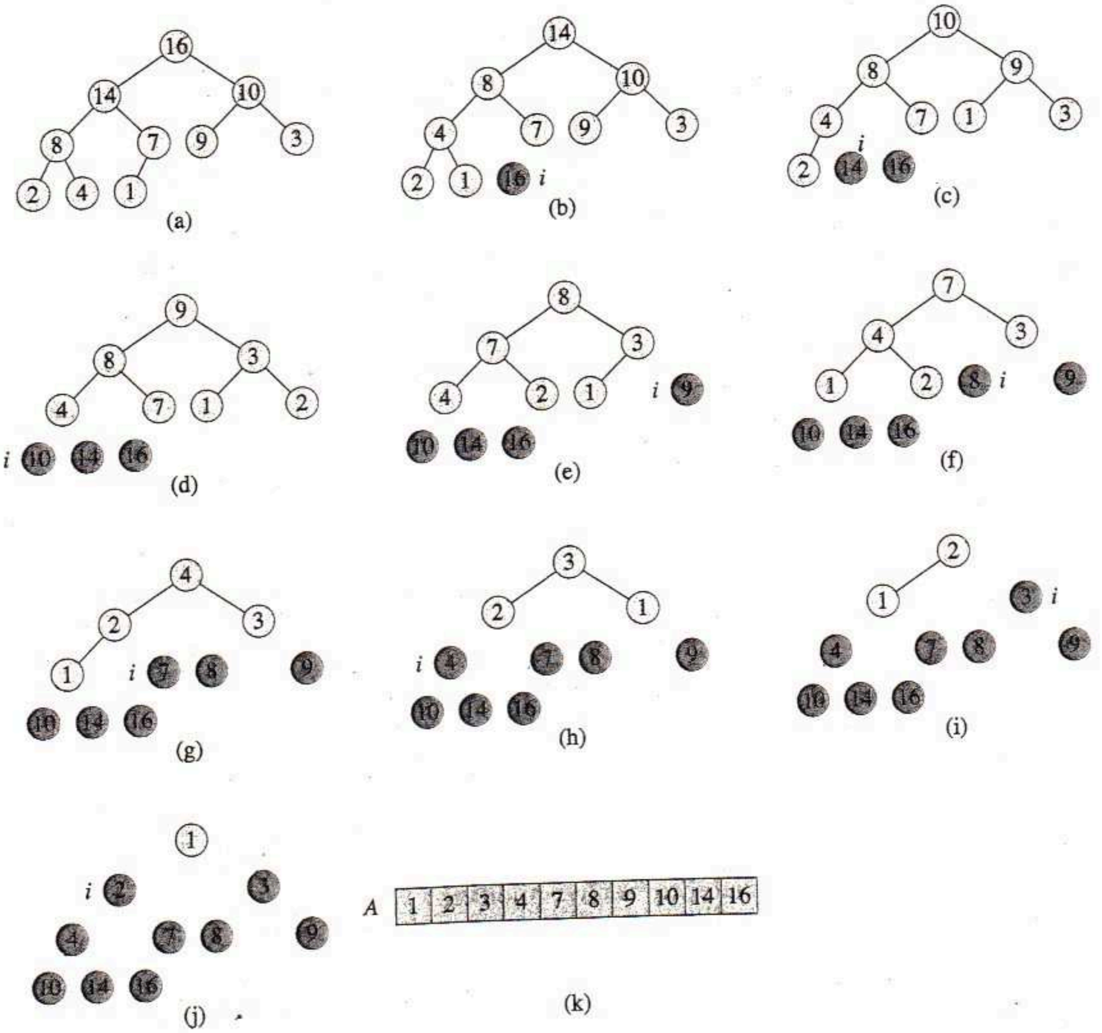


Figure 6.4 The operation of HEAPSORT. **(a)** The max-heap data structure just after BUILD-MAX-HEAP has built it in line 1. **(b)–(j)** The max-heap just after each call of MAX-HEAPIFY in line 5, showing the value of i at that time. Only lightly shaded nodes remain in the heap. **(k)** The resulting sorted array A.

Attachment #2

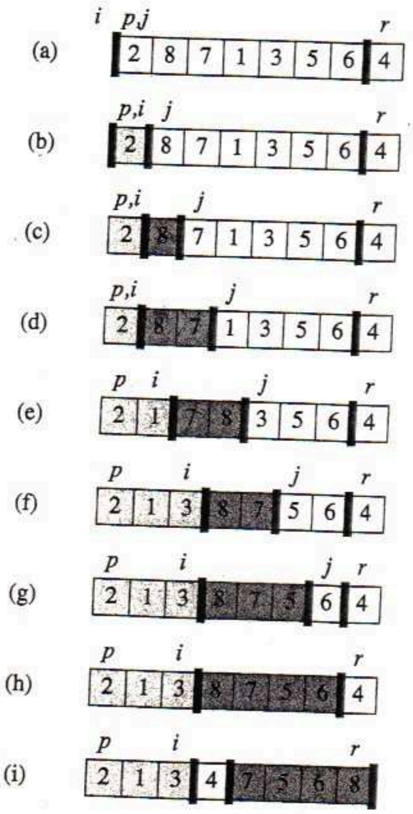
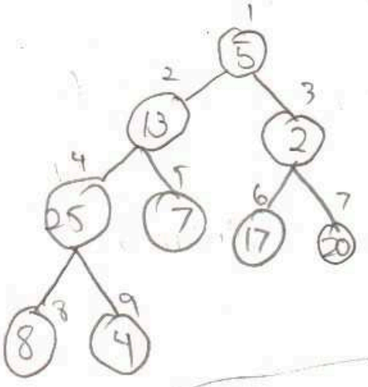


Figure 7.1 The operation of PARTITION on a sample array. Array entry $A[r]$ becomes the pivot element x. Lightly shaded array elements are all in the first partition with values no greater than x. Heavily shaded elements are in the second partition with values greater than x. The unshaded elements have not yet been put in one of the first two partitions, and the final white element is the pivot x. **(a)** The initial array and variable settings. None of the elements have been placed in either of the first two partitions. **(b)** The value 2 is "swapped with itself" and put in the partition of smaller values. **(c)–(d)** The values 8 and 7 are added to the partition of larger values. **(e)** The values 1 and 8 are swapped, and the smaller partition grows. **(f)** The values 3 and 7 are swapped, and the smaller partition grows. **(g)–(h)** The larger partition grows to include 5 and 6, and the loop terminates. **(i)** In lines 7–8, the pivot element is swapped so that it lies between the two partitions.

Problem 3: $A = \langle \overset{1}{5}, \overset{2}{13}, \overset{3}{2}, \overset{4}{25}, \overset{5}{7}, \overset{6}{17}, \overset{7}{20}, \overset{8}{8}, \overset{9}{4} \rangle$

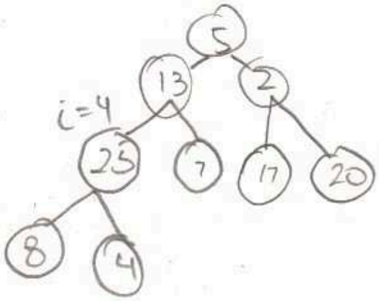
Attachment 1



First you heapify the structure starting at node = length/2

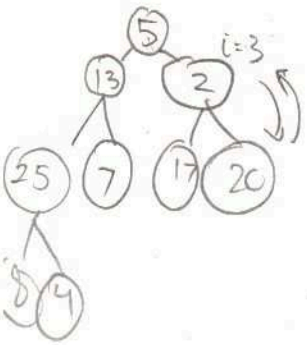
∴ 9/2 = 4

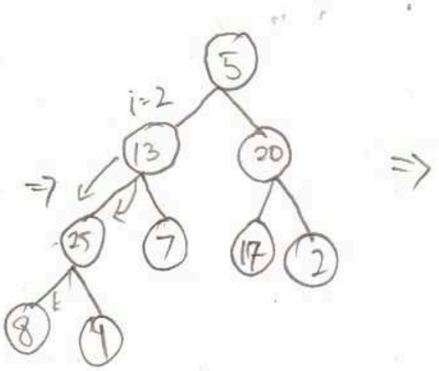
=>



// heapified

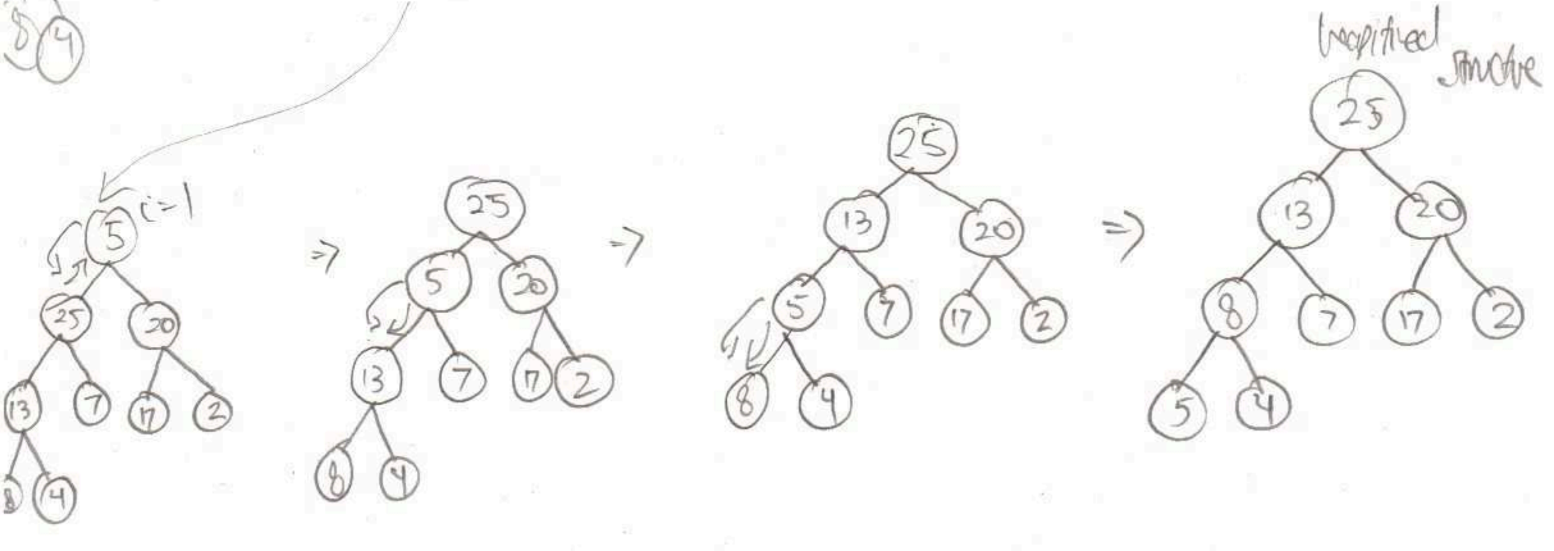
i = 3





=>

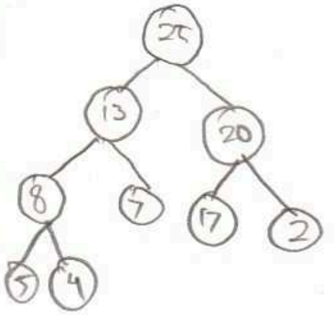
When i=2, you switch the 13 and 25, but have to check if the structure stays heapified. When 13 goes to node 4, it is still larger than the children 8 and 4, thus structure stays heapified

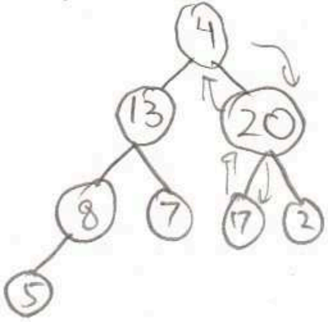


Now that the tree is heapified, we can now apply the heapsort function

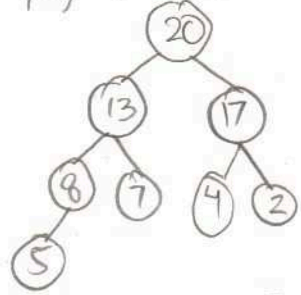
→

step 1: switch last node with first node, decrement size by 1
Then heapify structure



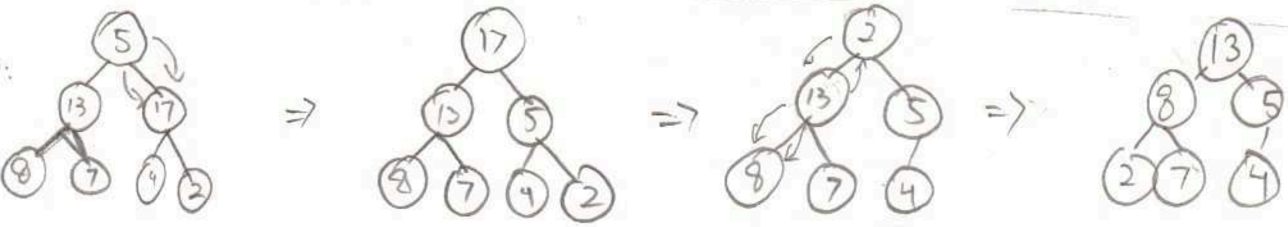


i 25

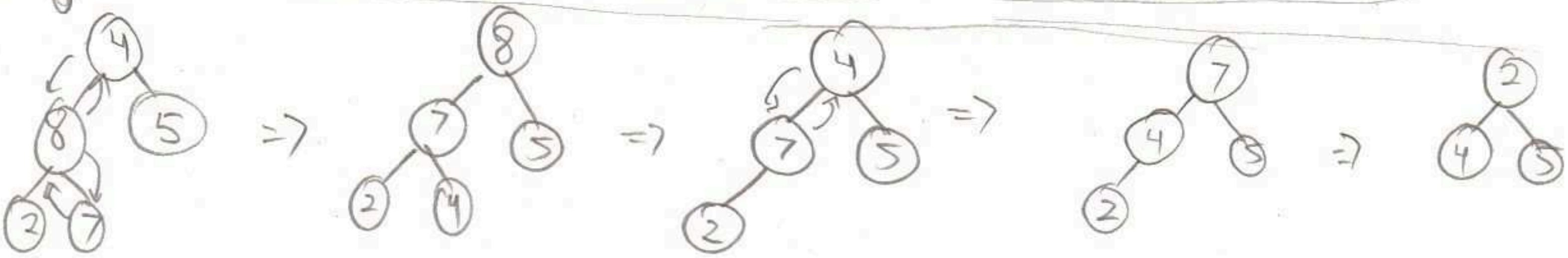


i: 25

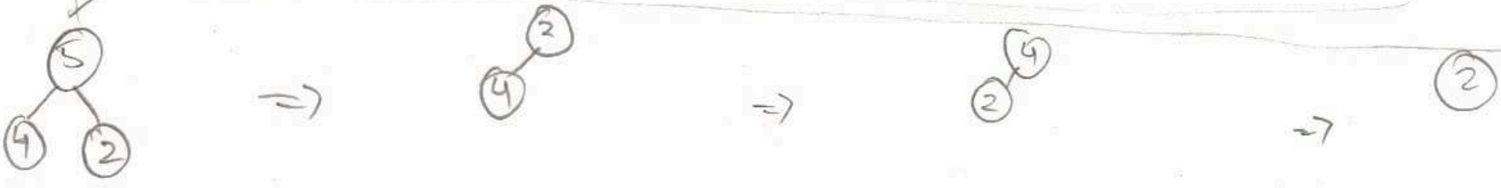
step 2:



i 20,25 i: 20, 25 i: 17, 20,25 i = 17, 20, 25



i = 13, 17, 20, 25 i = 8, 13, 17, 20, 25 i = 7, 8, 13, 17, 20, 25



i= 7, 8, 13, 17, 20, 25 i= 5, 7, 8, 13, 17, 20, 25 i= 5, 7, 8, 13, 17, 20, 25 i= 4, 5, 7, 8, 13, 17, 2

when there is 1 node left, it is placed at the beginning of the array.

The final array is A

2	4	5	7	8	13	17	20	25

Problem 4: Partition

A = < 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 >

▨ larger

p = beginning of lower partition

i = end of lower partition

j = position of pointer

i	p,j											r
	13	19	9	5	12	8	7	4	21	2	6	11

i	p	j										r
	13	19	9	5	12	8	7	4	21	2	6	11

i	p		j									r
	13	19	9	5	12	8	7	4	21	2	6	11

p,i			j								r
9	19	13	5	12	8	7	4	21	2	6	11

p	i			j							r
9	5	13	19	12	8	7	4	21	2	6	11

p	i				j						r
9	5	13	19	12	8	7	4	21	2	6	11

p		i				j					r
9	5	8	19	12	13	7	4	21	2	6	11

p			i				j				r
9	5	8	7	12	13	19	4	21	2	6	11

p				i				j			r
9	5	8	7	4	13	19	12	21	2	6	11

p				i					j		r
9	5	8	7	4	13	19	12	21	2	6	11

p					i					j	r
9	5	8	7	4	2	19	12	21	13	6	11

p						i					r
9	5	8	7	4	2	6	12	21	13	19	11

at the end, you put the r in position i+1

9	5	8	7	4	2	6	11	21	13	19	12