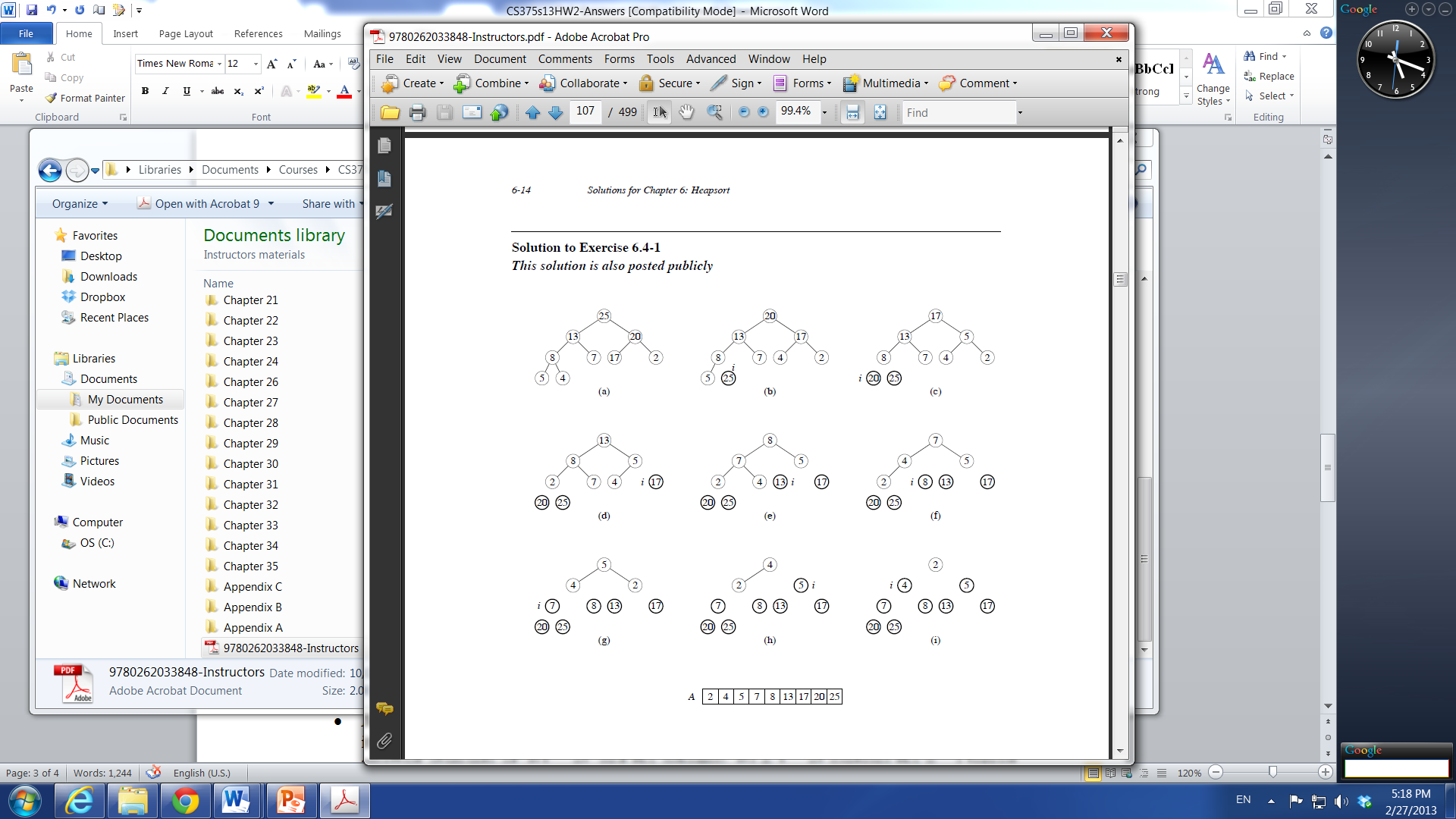
## Design and Analysis of Algorithms

## CS575, Spring 2020

1. (13 points) Illustrate the operation of Heapsort on the input array A = <5, 13, 2, 25, 7, 17, 20, 8, 4>. Draw the heap just after Build-Max-Heap was executed. Then draw a new heap after another (the next) element has been sorted; the last heap you draw has a single element.

Answer:



1. (13 points) Argue for the correctness of Heapsort using the following loop invariant: At the start of the iteration with an *i* of the for loop, (a) the subarray *A*[1 .. *i*] is a max-heap containing the *i* smallest elements of *A*[1 .. *n*], and (b) the subarray *A*[*i*+1 .. *n*] contains the *n* – *i* largest elements of *A*[1 .. *n*] in correctly sorted order (i.e., in ascending order).

**Answer**: We consider the three steps one by one.

* *Initialization*: Before the first iteration starts, *i* = *n*. At this time, *A*[1 .. *i*] contains all elements of *A* and *A*[*i*+1 .. *n*] is empty. Both (a) and (b) are trivially satisfied.
* *Maintenance*: Suppose the two conditions in the loop invariant hold at the start of the iteration with a given *i*. That is, that is, subarray *A*[1 .. *i*] is a max-heap containing the *i* smallest elements of *A*[1 .. *n*], and the subarray *A*[*i* + 1 .. *n*] contains the *n* – *i* largest elements of *A*[1 .. *n*] in correctly sorted order. During the iteration with this *i*, the algorithm first exchanges *A*[1] with *A*[*i*], where *A*[1] was the largest element in *A*[1 .. *i*] before the exchange because it is a max-heap. After the exchange, *A*[*i*] becomes the largest element among *A*[1 .. *i*], but it is still ≤ the elements in *A*[*i* + 1 .. *n*]. Now *A*[*i*] will be added to the sorted subarray and removed from *A*[1 .. *i*] (by not being included in the next Max-Heapify), which will make *A*[1 .. *i*] one element fewer (i.e., the new *i* for the next iteration was actually *i* – 1 for the previous iteration) and *A*[*i* + 1 .. *n*] one element more. Since *A*[*i*] was the largest element among *A*[1 .. *i*] before its move, it is clear that both properties (a) and (b) still hold after this iteration.
* *Termination*: The “for loop” terminates when *i* = 1. In this case, *A*[1] contains the smallest element of *A*[1 .. *n*] and the subarray *A*[2 .. *n*] contains the *n* – 1 largest elements of *A*[1 .. *n*] in correctly sorted order. This means that the entire array *A* is sorted correctly.

1. (14 points) We want to find the largest item in a list of n items.
   1. Use the divide-and-conquer approach to write an algorithm (pseudo code is OK). Your algorithm will return the largest item (you do not need to return the index for it). The function that you need to design is *int maximum (int low, int high)*, which *low* and *high* stands for low and high index in the array (8 points).
   2. Analyze your algorithm and show its time complexity in order notation (using .

Answer:

a)

int maximum (int low, int high) {

if (low == high)

return a[low];

int mid = (low +high)/2; // integer division

int L1 = maximum(low, mid);

int L2 = maximum (mid+1, high);

if A[L1] > A[L2] then

return L1

else

return L2

}

b)complexity analysis: T(n) = 2T(n/2) + 1, which yields