CS3130 Homework 3

Phillip Janowski (pajmc2@mail.umsl.edu)

November 3, 2018

Question 1.

6.1-1
What are the minimum and maximum numbers of elements in a heap of height h
Minimum will be a heap with only one node in its bottom level

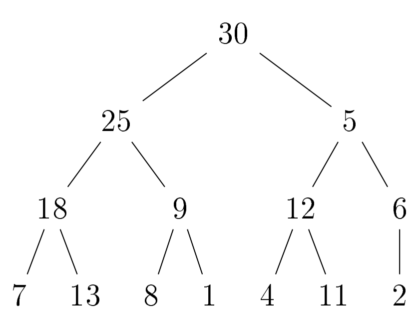
$$2^0 + 2^1 + 2^2 + ... + 2^{h-1} + 1$$
$$\frac{2^h - 1}{2-1} + 1 = 2^h$$

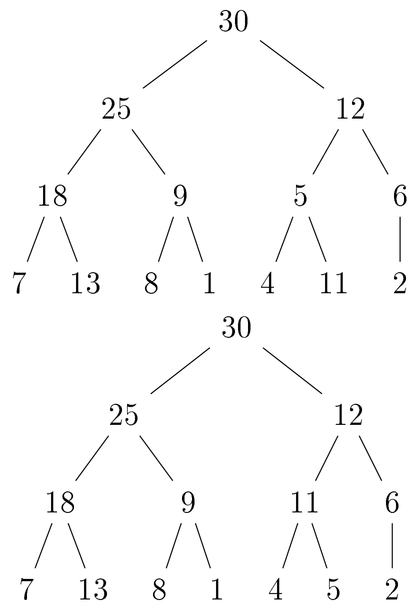
Maximum is a heap with the bottom node full

$$2^0 + 2^1 + 2^2 + ... + 2^h$$
$$\frac{2^{h+1} - 1}{2-1} = 2^{h+1} - 1$$

Question 2.

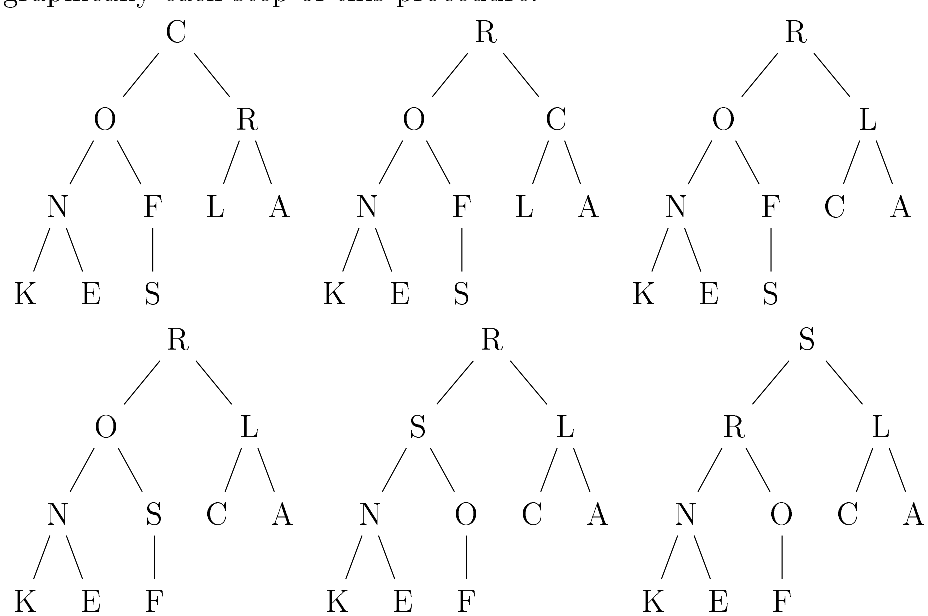
Using Figure 6.2 (p. 155) as a model, illustrate the operation of the function MAX-HEAPIFY (A, 3) on the array A=<30, 25, 5, 18, 9, 12, 6, 7, 13, 8, 1, 4, 11, 2>
Here $i = 3$ and $A[3] = 5$





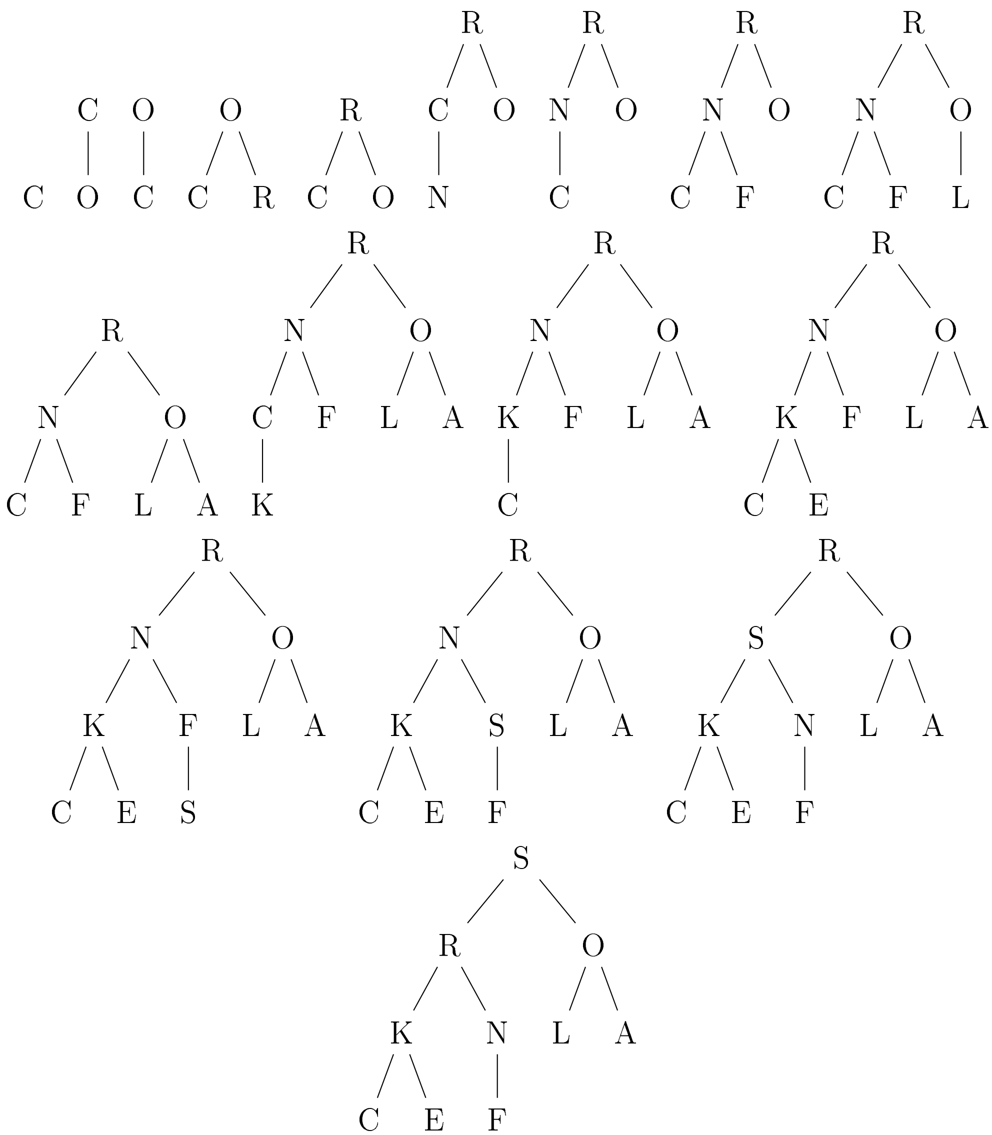
Question 3.

Given a sequence: <C, O, R, N, F, L, A, K, E, S >. Build a heap using a bottom-up approach (see the function BUILD-MAX-HEAP, p.157); show graphically each step of this procedure!



Question 4.

Given a sequence: $<$ C, O, R, N, F, L, A, K, E, S $>$. Build a heap using a top-down approach (see the handout posted on MyGateway site); show graphically each step of this procedure!



Question 5.

6.2-3
What is the effect of calling Max-Heapify(A,i) when the element A[i] is larger than its children?

No effect. The three if conditions,

$$\text{if } l \leq A.heap - size \text{ and } A[l] > A[i],$$
$$\text{if } r \leq A.heap - size \text{ and } A[r] > A[largest]$$
$$\text{if } largest \neq i$$

all fail and nothing changes.

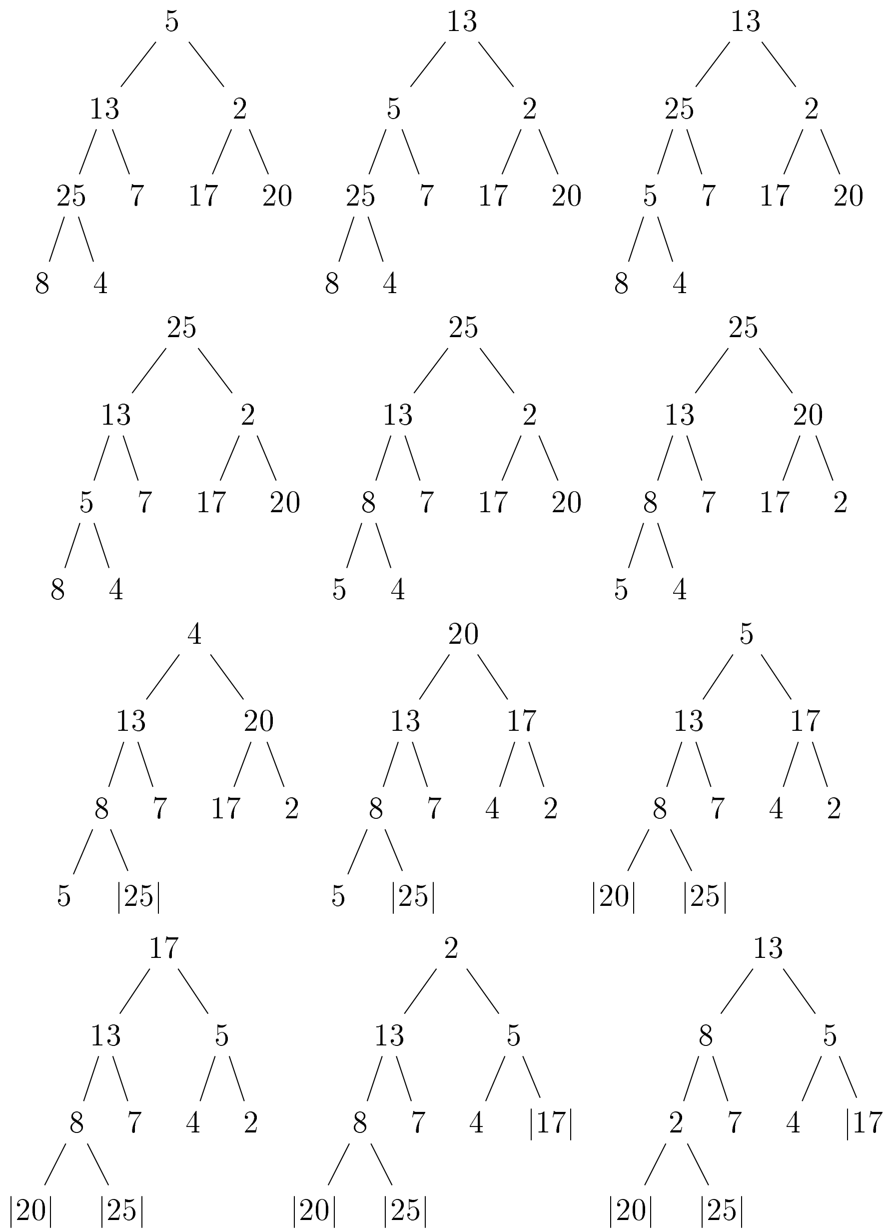
Question 6.

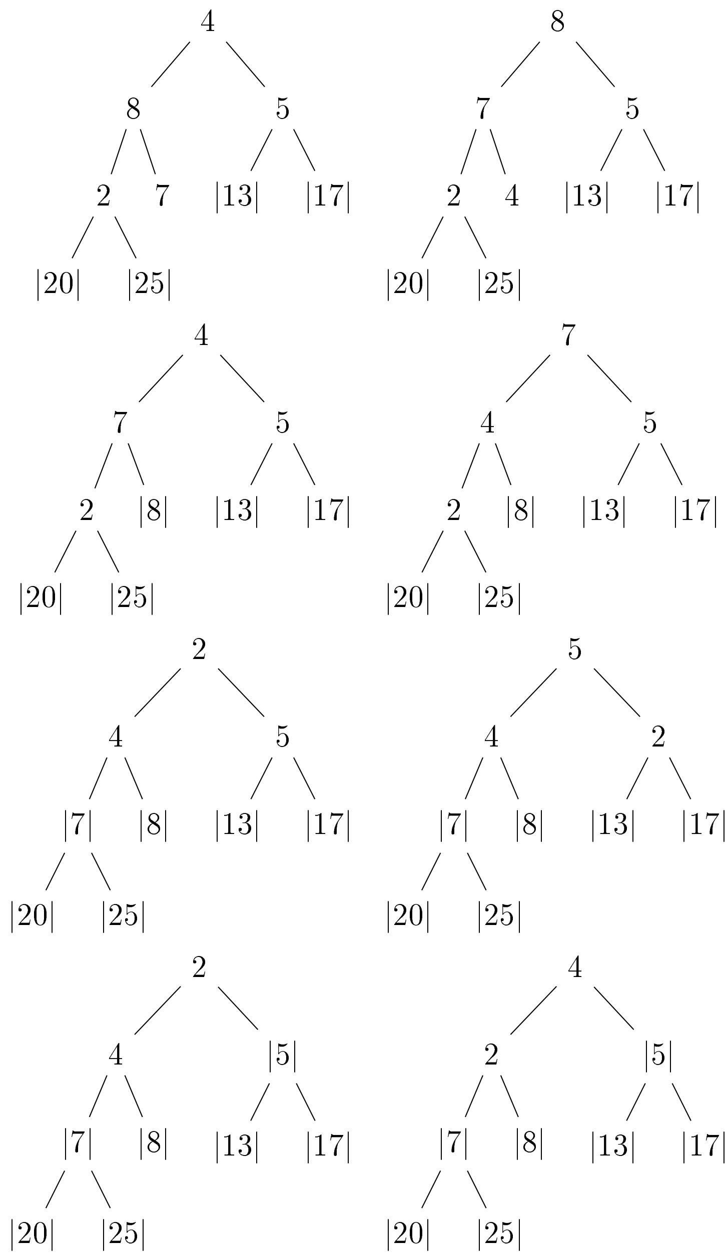
6.2-4
What is the effect of calling Max-Heapify(A, i) for $i > A.heap - size/2$?

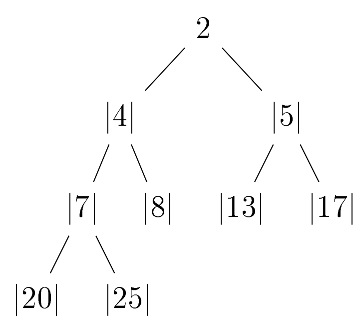
If $i > A.heap - size/2$, then node i is either in the lowest or second lowest level of the tree and does not have any children. Whene LEFT(i) and RIGHT(i) are called the array index that gets called will be out of bounds.

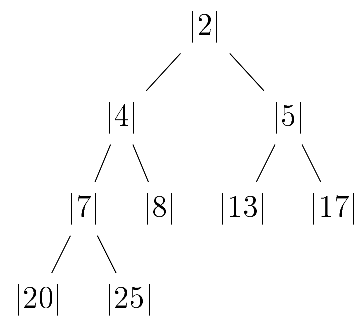
Question 7.

6.4-1
Using Figure 6.4 as a model, illustrate the operation of HEAPSORT on the array A $<$5, 13, 2, 25, 7, 17, 20, 8, 4$>$


5 13 2 25 7 17 20 8 4
13 5 2 25 7 17 20 8 4
13 25 2 5 7 17 20 8 4
25 13 2 5 7 17 20 8 4
25 13 2 8 7 17 20 5 4
25 13 20 8 7 17 2 5 4
4 13 20 8 7 17 2 5 |25|
20 13 17 8 7 4 2 5 |25|
5 13 17 8 7 4 2 |20| |25|
17 13 5 8 7 4 2 |20| |25|
2 13 5 8 7 4 |17| |20| |25|
13 8 5 2 7 4 |17| |20| |25|


4
8
5
2
7
|13|
|17|
|20|
|25|
8
7
5
2
4
|13|
|17|
|20|
|25|
4
7
5
2
|8|
|13|
|17|
|20|
|25|
7
4
5
2
|8|
|13|
|17|
|20|
|25|
2
4
5
|7|
|8|
|13|
|17|
|20|
|25|
5
4
2
|7|
|8|
|13|
|17|
|20|
|25|
2
4
|5|
|7|
|8|
|13|
|17|
|20|
|25|
4
2
|5|
|7|
|8|
|13|
|17|
|20|
|25|





Sorted A = <2, 4, 5, 7, 8, 13, 17, 20, 25>

Question 8.

6.4-3

What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

HEAPSORT will be $O(n\log(n))$ for both increasing and decreasing order sorted order.

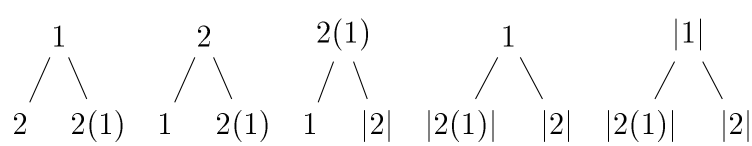
For increasing, BUILD-MAX-HEAP will have a max run time of $\Theta(n)$ and each MAX-HEAPIFY(A,1) will take at most $O(\log(n))$ which results in $O(n\log(n))$

For decreasing, MAX-HEAPIFY(A, i) will result in $O(1)$ but the BUILD-MAX-HEAP will result in $\Theta(n)$ run time because there will be $[n/2]$ calls of MAX-HEAPIFY. Also, there will be $n-1$ calls of MAX-HEAPIFY(A, 1) will result in $O(\log(n))$ resulting in $O(n\log(n))$ run time.

Question 9.

Give an example to show that a heapsort algorithm is not stable.

A = <1, 2, 2(1)>, here 2 and 2(1) will be equal.

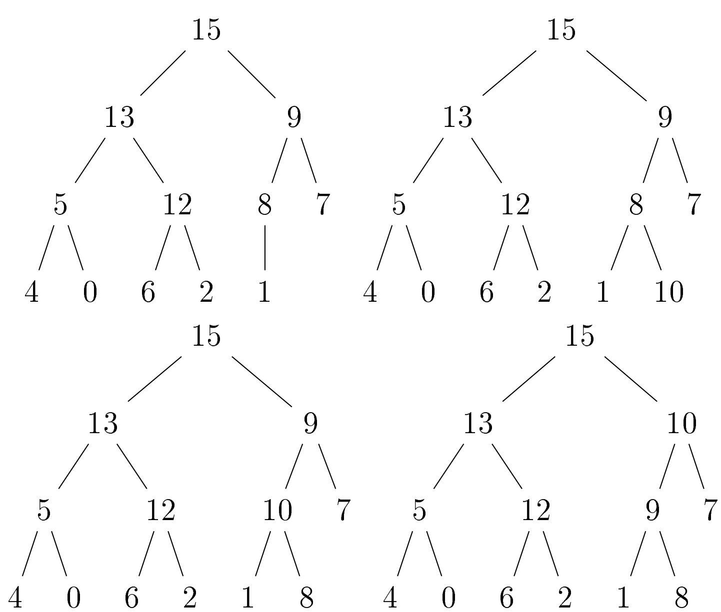


This shows A = <1, 2, 2(1)>, when sorted by HEAPSORT, will result in

<1, 2(1), 2>

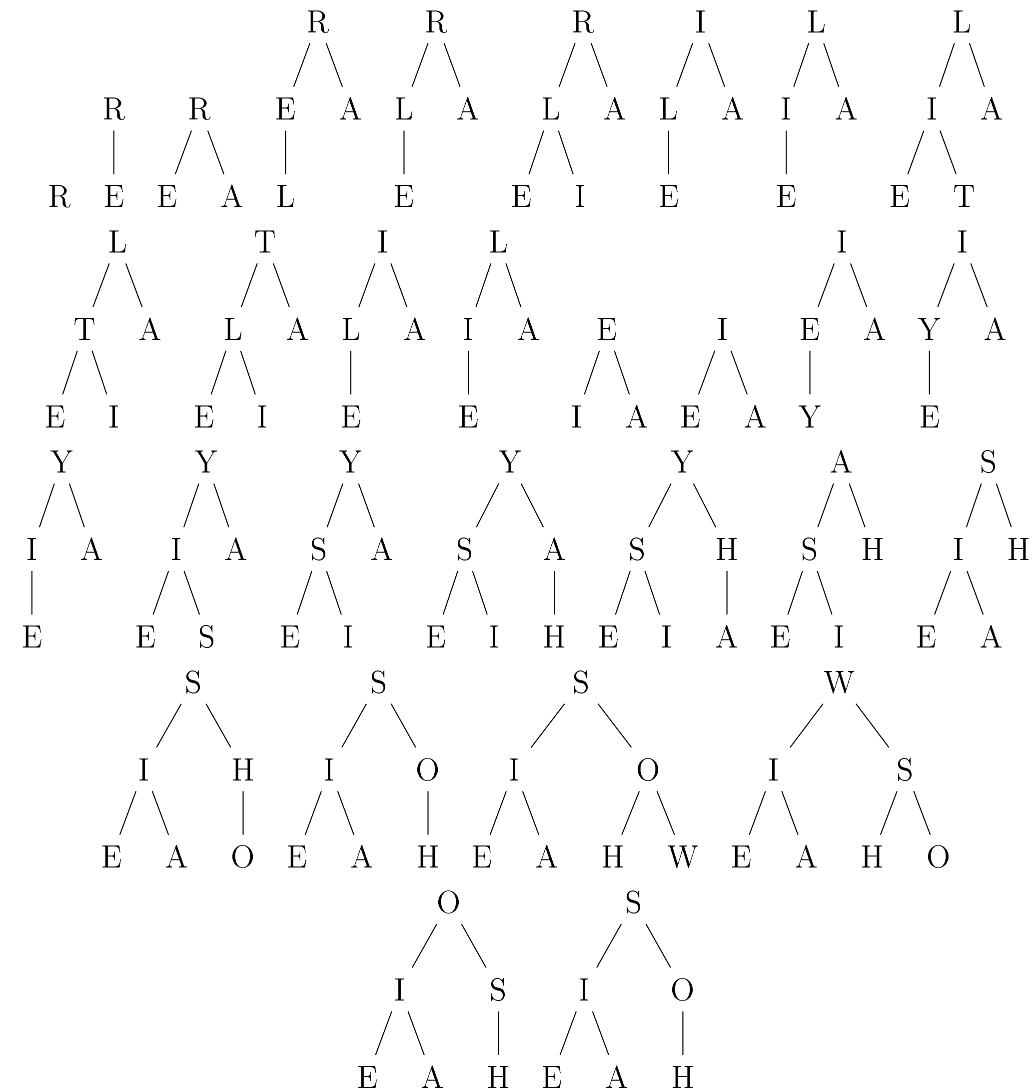
Question 10.

6.5-2
Illustrate the operation of MAX-HEAP-INSERT(A, 10) on the heap A <15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2>.



Question 11.

In the sequence below, the character means âĂŸinsert into priority queueâĂŹ operation; the asterisk âĂŸ*âĂŹ means âĂŸextract the maximum from a priority queueâĂŹ operation. Show the heap in the end of each operation: R E A L I * T * * Y S H * O W * *



Question 12.

You are given a collection of n bolts of different diameters and n corresponding nuts. You are allowed only to try a nut and a bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut. Design an algorithm to solve this problem that will be similar to a

quicksort (use the idea of partitioning).
Select a bolt and test every nut to find its match while also partitioning the nuts into a set of nuts that is too large and another for nuts that are too small. When the matching nut is found use that nut to partition the other bolts.