Homework 2

1. Question 2.4.2

Criticise the following idea: To implement *find the maximum* in constant time, why not use a stack or a queue, but keep track of the maximum value inserted so far, then return that value for *find the maximum* (assume we'd also like to support delete operations)?

If the queue or stack supports any sort of remove or delete operation (such as pop or dequeue), the structure would also need to check at that point if the maximum value is being removed and then iterate through the entire structure to find the new maximum. Simply keeping track of the largest value inserted would not work in either case.

2. Question 2.4.4

Is an array that is sorted in decreasing order a max-oriented heap?

Yes. The root is the max value, its children are second and third largest, and their children continue that pattern. The final value would be the smallest value of the heap and also the final leaf in the deepest level of the tree.

3. Question 2.4.11 and 2.4.12

Suppose that your application will have a huge number of *insert* operations, but only a few *remove the maximum* operations. Which priority-queue implementation do you think would be most effective: heap, unordered array, or ordered array?

An unordered array would be the most effective as it can do insertions in constant time and the cost of the linear operation to remove a maximum value would be outweighed by the benefit to insert.

Suppose that your application will have a huge number of *find the maximum* operations, but a relatively small number of *insert* and *remove the maximum* operations. Which priority-queue implementation do you think would be most effective: heap, unordered array, or ordered array?

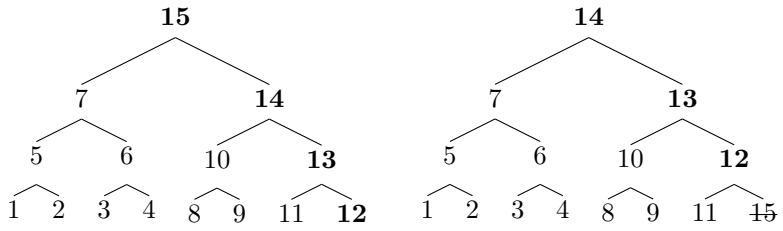
A max-oriented heap would be the most beneficial in this case, as the maximum value is always the first element of the heap, far outweighing the cost of

the logarithmic *insert* and *remove the maximum* operations.

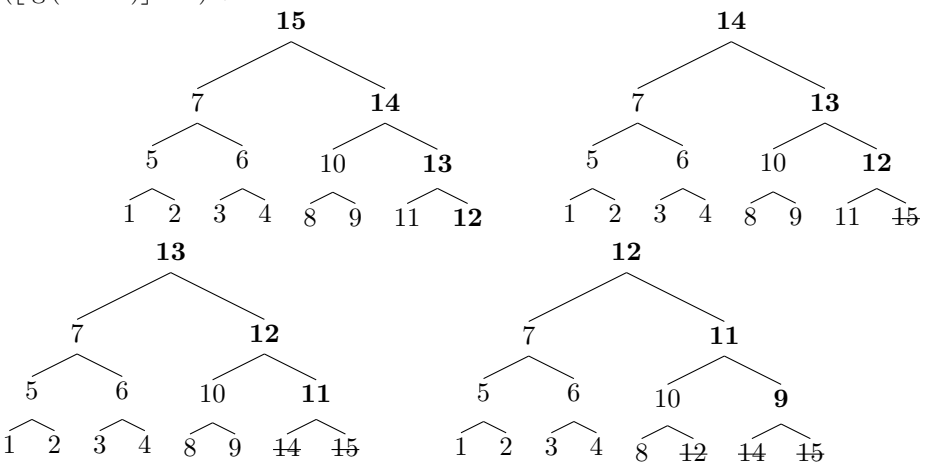
4. 2.4.14

What is the minimum number of items that must be exchanged during a *remove the maximum* operation in a heap of size N with no duplicate keys? Give a heap of size 15 for which the minimum is achieved. Answer the same questions for two and three successive *remove the maximum* operations.

The minimum number of exchanges $X = \lfloor \lg N \rfloor - 1$ for one *remove the maximum* operation.



Successive operations change the formula slightly to $X = (\lfloor \lg N \rfloor - 1) + (\lfloor \lg (N-1) \rfloor - 1) + ...$



5. One of the following Midterm questions: Spring 2008 Question 5, Fall 2008 Question 4, Fall 2009 Question 3, etc.

Spring 2008 Question 5

Consider the following binary heap (i.e., the array-representation of a heap-ordered complete binary tree).

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-	Z	W	Y	T	G	K	V	R	S	F	A	-	-

A. Delete the maximum key. Give the resulting binary heap. Circle those values that changed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-	(Y)	W	(V)	T	G	K	(A)	R	S	F	(Z)	-	-

B. Insert the key X into the original binary heap. Give the resulting binary heap. Circle those values that changed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-	Z	W	Y	T	G	(X)	V	R	S	F	A	(K)	-