Oblig 2 Algoritmer

1a:

oppgave1.tif

1b:

Ascending order (1,2,3,4,5,6 etc). The rightmost element (on level n – 1, or just n if we are in the state with 2n – 1 elements) will be filled up before swapping to the closest left element, then the procedure are done over and over again until we reach 2n – 1. It will then be a perfect binary tree.

1c:

Descending order (10,9,8,7,6,5,4,3,2,1). The new element which is the root of an “one element heap”, will be placed at top, and make the previous root as it’s left child. This will always be the case. Insertion is easy because as explained over, the new element will be made root, and the remaining heap it’s children. Deletemin will just make the root’s left child the new root, and then delete the previous root.

2a:

CODE

2b:

Dijkstra’s data:

Test 1: 27 ms.

Nodes made: 32

Moves made: 17

Level: 5

Path: DLDR

Test 2: 39 ms.

Nodes made: 188

Moves made: 105

Level: 7

Path: RDLURDD

Test 4: 54 ms.

Nodes made: 398

Moves made: 192

Level: 7

Path: RDRRDD

A\* search data:

Test 1: 24 ms.

Nodes made: 9

Moves made: 4

Level: 4

Path: DLDR

Test 2: 28 ms.

Nodes made: 22

Moves made: 12

Level: 7

Path: RDLURDD

Test 4: 29 ms.

Nodes made: 14

Moves made: 6

Level: 6

Path: RDRRDD

If we look at the data output from the two methods, we clearly see that that A \* is both quicker and makes much less nodes to reach the goal. And the basic reason for that is that it will make a balanced tree no matter what. It will find the node with the lowest g(v) (the lowest level in the tree), and try to find a way from there. Every time.

3a:

If the language is touring complete, every algorithm can be implemented.

3b:

Dette skjønner jeg ikke enda. Så leverer uten, og om den ikke blir godkjent uten denne oppgaven, leverer jeg på nytt. Håper det er greit.