Aikavaativuudet:

Huffman Enkoodaus

O(n log k), missä k on nodejen määrä ja n on charactereiden määrä tiedostossa

Huffman Dekoodaus

O(n log n), jokainen luettu bitti luetaan ensin jonoon, josta ne luetaan pois ja jokaisen inputin kohdalla kestää
logaritminen aika löytää vastaava character.

LZW Enkoodaus

O(k*n), tietorakenteena yleisesti käytetään hashMappia, mutta minulla on köykäisempi setti, jonka pahin tapaus etsiä string avain on 4096 = k. Tämä hidastaa suoritusta paljon.

LZW Dekoodaus

O(n), tässä ei mitään ihmeellisempää, luetut inputit talletetaan Integer to String - mappiin, josta ne saadaan vakioajassa.

Tilavaativuudet:

Huffman Enkoodaus

O(n+k+512) = O(k), missä n on charactereistä muodostetun binääripuun koko ja k on luettavan tiedoston pituus.
Lisäys 512 tulee kahdesta aputaulusta, mutta niiden tilat ovat vakioita. k on lähes aina selvästi suurin.

Huffman Dekoodaus

O(n+k) = O(n), missä n on tiedoston pituus bitteinä ja k on eri charactereista muodostetun puun koko.

LZW Enkoodaus

O(624288000) eli vaatii käytännössä lähes koko javan virtuaaliheapin (oletusarvoisen) maksimitilan, koska tiedoston (maksimi)pituus on ohjelmassa kovakoodattu ja se luetaan maksimipituiseen charlistaan. Se kuitenkaan harvoin on ongelma,
koska oletusarvoinen keon maksimi ei ole kovin iso nykykoneille. Ikävä kyllä se rajoittaa kompressoitavan tiedoston maksimikokoa.

LZW Dekoodaus

O(n), missä n on 12-bitin jonojen lukumäärä tiedostossa.

LZW:tä pitäisi parantaa paljon, siinä ajanpuutteen takia jäi tyhmyyksiä, kuten kovakoodattu charlista LZW enkoodauksessa.
Se ei myöskään tunnu toimivan oikein isoilla syötteillä.