Advanced Data Structure programming assignment

COP 5536 Spring 2017
Instructor : Dr. Sartaj K Sahni

Name : Tanya Pathak
UF ID-18373292
Email Id- tanya.pathak@ufl.edu

BinaryHeap:

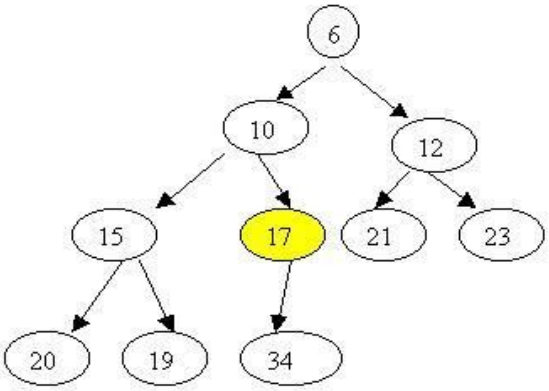
A binary heap is a complete binary tree, where each node has a higher priority than its children.This is called heap-order property.

- the *min-heap property*: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
- the *max-heap property*: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.

Complete binary tree: Each node has two children, except for the last two levels. (The nodes at the last level do not have children.)
New nodes are inserted at the last level from left to right.

This is called heap-structure property.

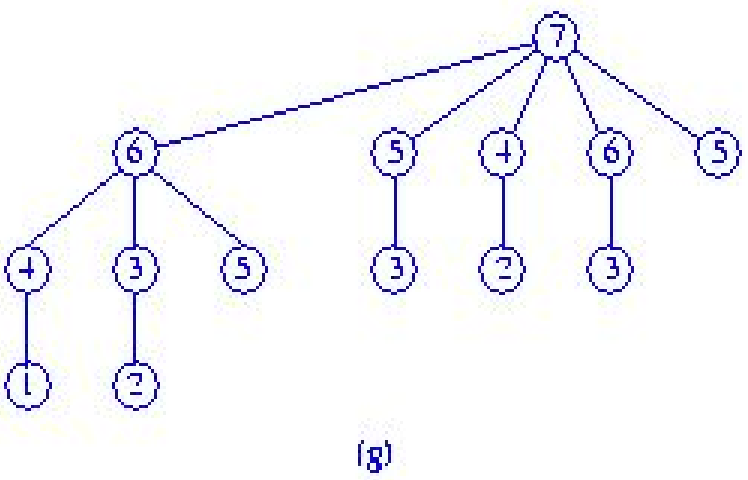
Since a heap is a complete binary tree, it has a smallest possible height - a heap with N nodes always has O(log N) height. The tree below is a Complete Binary Heap Tree:



Pairing Heap:

A min (max) pairing heap is a min (max) tree in which operations are done in a specified manner.

Here is an example of max pairing heap:



(g)

Amortized complexity of pairing heap operations insert,remove,decrease key,meld,remove min is O(log n).

Node Structure includes a child pointer,left and right sibling,data fields.

Left and Right siblings:Used for doubly linked linked list (not circular) of siblings.Left pointer of first node is to parent.

There is no parent,childcut or degree fields.

FourWayHeap

It is a complete n node tree whose degree is 4 .It is a Min (max) tree.

Number nodes in breadth-first manner with root being numbered 1 .

Parent(i) = ceil((i – 1)/4) .

Children are 4*(i – 1) + 2 , ..., min{4*i + 1, n} .

Height is $\log_4 n$.

Height of 4 -ary heap is half that of 2 -ary heap.

Worst-case insert moves up half as many levels as when d = 2 .

Average remains at about 1.6 levels.

Remove-min operations now do 4 compares per level rather than 2 (determine smallest child and see if this child is smaller than element being relocated).

But, number of levels is half.

Other operations associated with remove min are halved (move small element up, loop iterations, etc.)

4-heap cache utilization

Shift 4-heap by 2 slots.

Siblings are in same cache line.~$\log_4 n$ cache misses for average remove min (max).

4-ary heap performance:

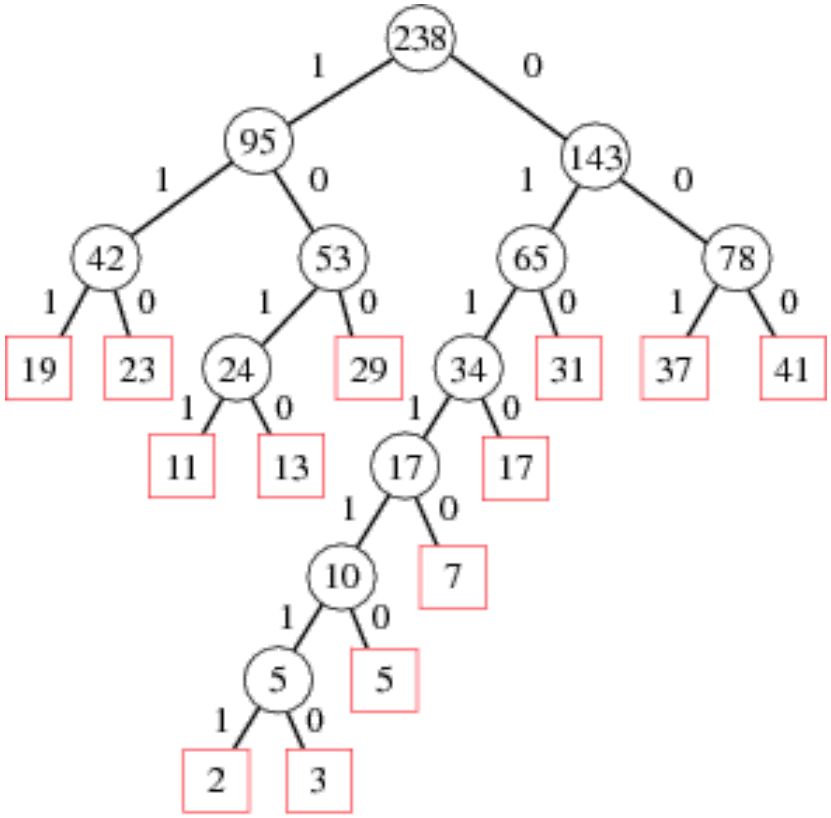
Speedup of about 1.5 to 1.8 when sorting 1 million elements using heapsort and cache- aligned 4 -heap vs. 2 -heap that begins at array position 0 .Cache-aligned 4 -heap generally performs as well as, or better, than other d-heaps.

Huffman Encoding

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.
The variable-length codes assigned to input characters are prefix codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.

Why huffman encoding is used?

Huffman coding is a technique used to compress files for transmission.Uses statistical coding.More frequently used symbols have shorter code wordsWorks well for text and fax transmissions



Which one is the best?

After implementing all the above three data structures we find out that fourway heap comes out to be the best data structure for huffman encoding

FourWayHeapImplementation

Node Structure

- **Data:** This field contains the elements stored in the files.This is later used for huffman encoding
- **Frequency** :This field contains the frequency of each element stored in the file.
- **Left:**This field stores the left child of the node.
- **Right:**This field stores the right child of the node.

Inbuilt Data Structure used:

- HashMap<Integer, Integer> ftable = new HashMap<>();
  This hashmap stores the frequencies of elements in the files.
- Iterator iter = freqtab.entrySet().iterator();
  An iterator class is used to iterate over each hashmap used in the program
- Map.Entry pair = (Map.Entry)it.next();
  The allows you to iterate over **Map**.entrySet() used in the program.
- ArrayList<HuffmanNode> Heap = new ArrayList<HuffmanNode>();
  This stores an arraylist containing the nodes of fourwayheap.
- HashMap<Integer,String> hcodes = new HashMap<>();
  This hashmap stores the integer and its codes.

Associated Files

My files :

1. **Encoder.java:** This is the file needed to calculate the frequencies of the elements and store in code_table.txt file then calculate the huffman encoding using the 4 ary optimized heap structure and write to the file code_table.txt.Then use the code_table.txt file to create a hashmap containing the elements and their respective codes.Then it uses the sample_input_large.txt file and the hashmap generated to write the encoded string to the encoded.bin file.

2. **Decoder.java:** This file builds the decodertree using the huffman code generated and stored in code_table.txt file and encoded.bin.Then traverses the tree to write to the decoded.txt file.

3**.Makefile :** A makefile is an easier way to organize code compilation. A makefile to compile and run encoder.java and decoder.java is created.

Files given for evaluation :

sample_input_small.txt: The file is used for the huffman tree compression and generating the encoder decoder for a small set of data.
sample_input_large.txt: This file is primarily given for q ivn for performance analysis.and checking the efficiency and complexities of our encoding and decoding logic.

Compiler Description

The project has been compiled and tested under the following platforms :

Platform/ OS	Compiler	Test Result
● Ubuntu (14.04)	jdk 1.8	Test passed
● Mac OS Sierra 10.12.3	jdk 1.8	Test passed

Steps to execute the project assignment is as following :

1. Copy all the files including the make file in a separate directory ads_p zip submitted.

2. Reach to that directory through terminal.

3. Copy the data files sample_input_small.txt and sample_input_large.txt

4. Run make.

5. Now, the project is ready to be executed. The project can be executed by this way:
    a. Type make

    b. This command should produce two binary files: encoder and decoder
    c. We will run it using following command:
        i. $ java encoder <input_file_name>
    d. Running encoder program must produce the output files with exact name "encoded.bin" and "code_table.txt".
    e. On the other hand, decoder will take two input files. We will run it using following command:
    f. $ java decoder <encoded.bin> <code_table.txt>
    g.Running decoder program must produce output file with exact name "decoded.txt".

Function Description and Structure overview

There are 2 files Encoder.java,Decoder.java and the functionalities are listed below:

The file **Encoder.java** includes the following functions needed for the implementation of the given project assignment **:**

FourWayHeap.class :This class contains the heap structure required for huffman encoding.This includes all the heap operations like

upheapfy,downheapify,removemin and insert.The functions used in this class are:

- **public void insert(Node node)():**The function has an argument which is defined as Node node. This function inserts a new node into the fourway heap and calls the heapifyUp operation on the last node.
- **public void heapifyDown(int f):** This function takes the argument index at which heapifydown needs to be done.This finds index with smallest frequency and if the index !=the smallest,swaps with minimum and recurses.
- **public void heapifyUp(int child):**This function takes the child index argument and performs heapify up operation on it.This function calculates the parentnode with the given child index and then checks the minimum of the two and then swaps with minimum and recurses.
- **public Node removeMin():**This function removes the minimum node by removing the first node of the cache optimized heap if heap size ==4 and if heap size > 4 ,this swaps with the last node and removes the last node and performs heapifyDown on the first index of the cache optimized heap.

class Encoder contains the following functions for the huffman encoded file generation.The following functions are included in the Encoder.class

- **static void buildCodeTable(Node root, String code, PrintWriter of):**This function contains the argument root of the tree using which the code table has to be built.

  Algorithm used is :

  Traverse the root till every leaf and assign 0 to the left branch of the heap and 1 to the right branch of the heap until the leaf is encountered.The resulting codes are added to the output file code_table.txt using printwriter operations.

  Main(): This function does all the major operations-
  creating frequency table,implementing the 4wayheap,huffman encoding and writing to final encoded.bin and code_table.txt file.

The function creates a hashmap of value and count pairs using the input files and then uses the hashmap to write to output frequency file called freq.txt. Then this function reads the frequencies from the file freq.txt and inserts the frequencies and the elements in the fourwayheap structure.after inserting the algorithm for the calculation of huffman tree is executed. We start deleting 2 elements with the minimum frequencies from the fourwayheap and calculate the sum of the frequencies and add it to the heap with frequency of the new node equal to the sum of the two frequencies.This new node also contains a left and a right child pointer which is assigned the values of the two values coming from the removeMin().This operation is continued until we reach the last node in the fourwayheap and that is stored.The buildcodetable is then called with this last node acting as the root of the huffman tree and the output code file code_table.txt.This function reads the code_table.txt file and creates a hashmap of the element and the code.Then I use the sample_input_large.txt file and the hashmap created to write to the new file encoded.txt file.

The file **Decoder.java** contains the following function.

- **static void decodertree(String s):**This function takes the element code as argument and generates the decode tree.if there is 1 encountered in the code a right branch to the root is created and if there is a 0 encountered in the code a left branch is created and this is how the decoder tree is generated using the huffman codes.

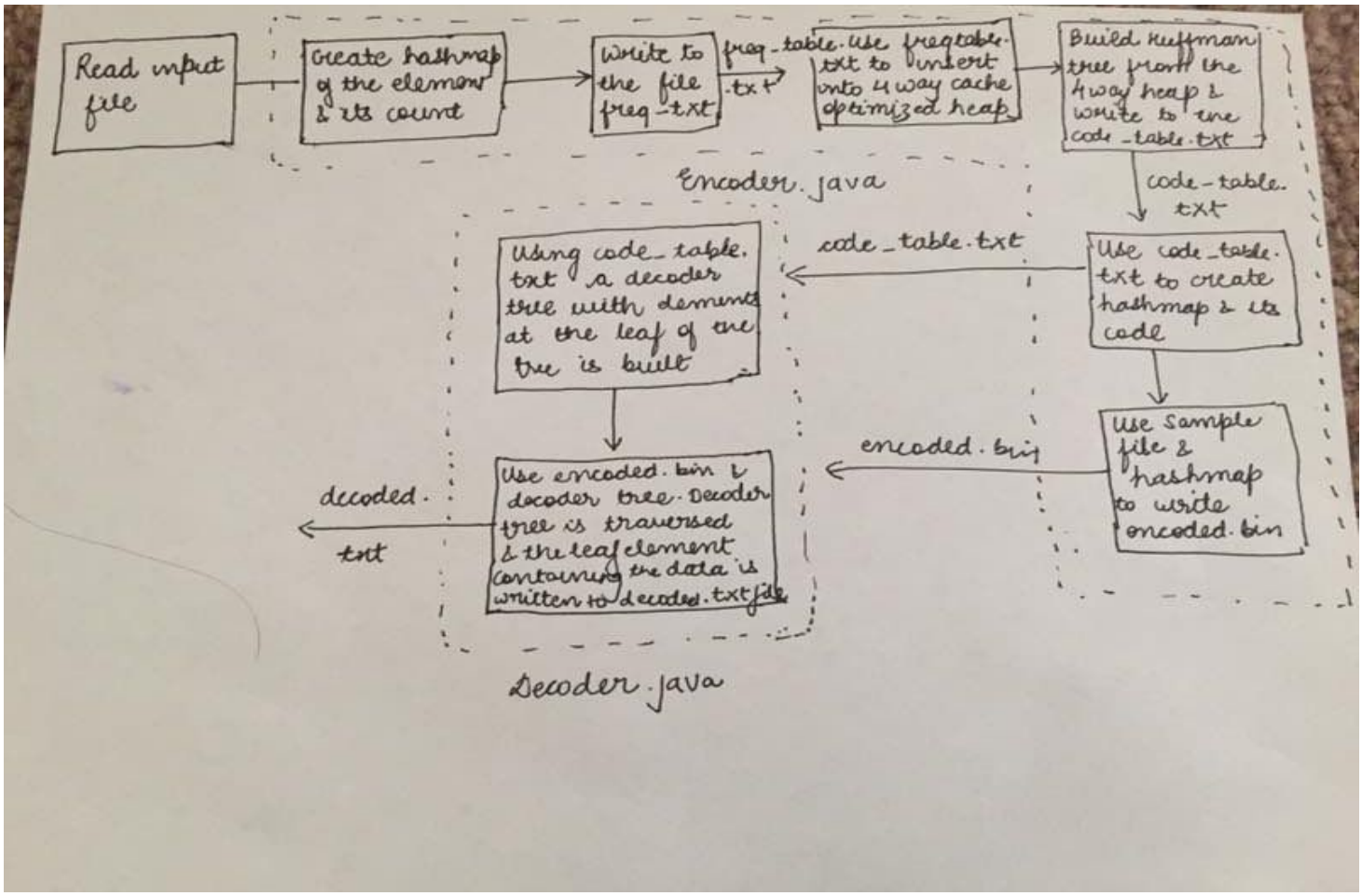
- **static void traversal(BitSet b, PrintWriter pwr):**This traverses down the tree containing the elements and writes the elements to the decoded.txt file.
- **Main():**This function reads the code_table.txt file and builds decodertree and then reads encoded.bin and calls traversal() on it to write the decoded.txt file.

DECODING ALGORITHM USED AND COMPLEXITY:

The decoding algorithm is same like creating huffman tree.We are reading data and its code from code_table.txt file and building decoder tree at root for the elements by moving right if we encounter 1 in the code of the element and moving left when we encounter 0 in the code of the element.Thus the elements are stored at the leaf of the tree.Then we traverse this tree and use encoded.bin.Whenever we encounter 1 we move to the right of the tree and move to the left of the tree when we
encounter a 0.If we have reached the leaf that contains the element so that is added back to decoded.txt file.

The complexity of the decoding algorithm is O(bn) where b is bit length and n is the no of elements in the sample file

Structure of the program

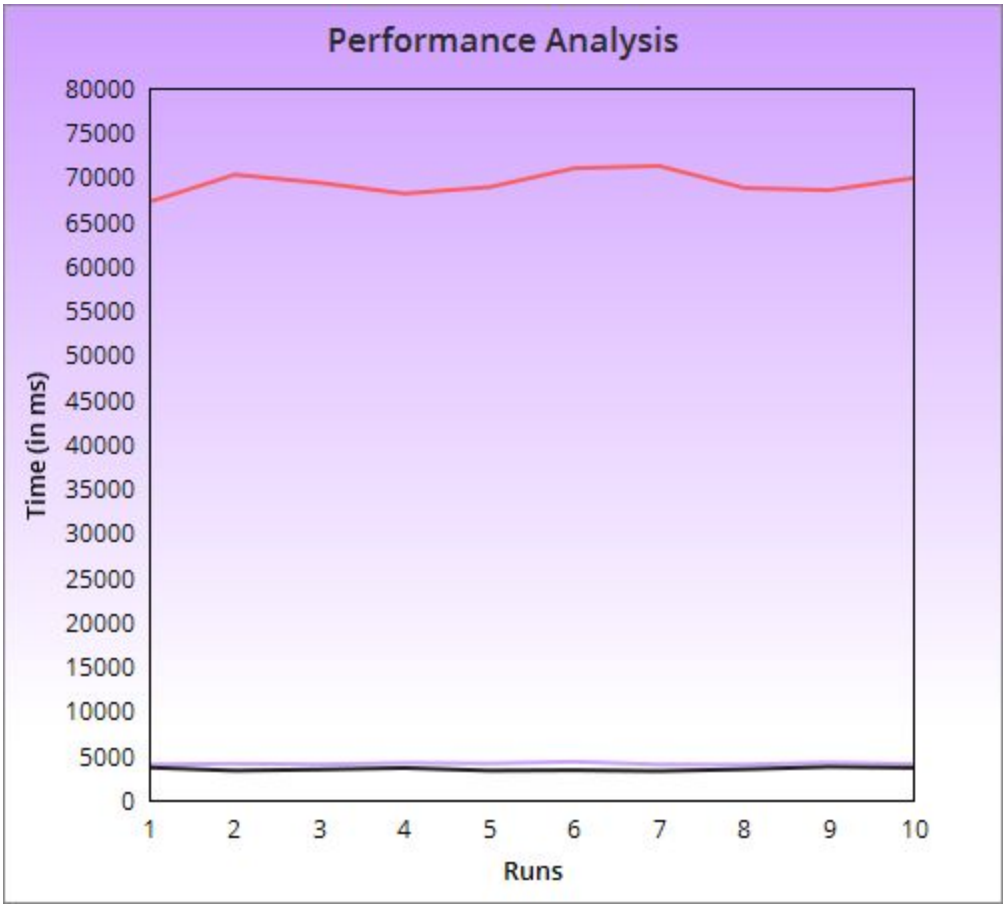


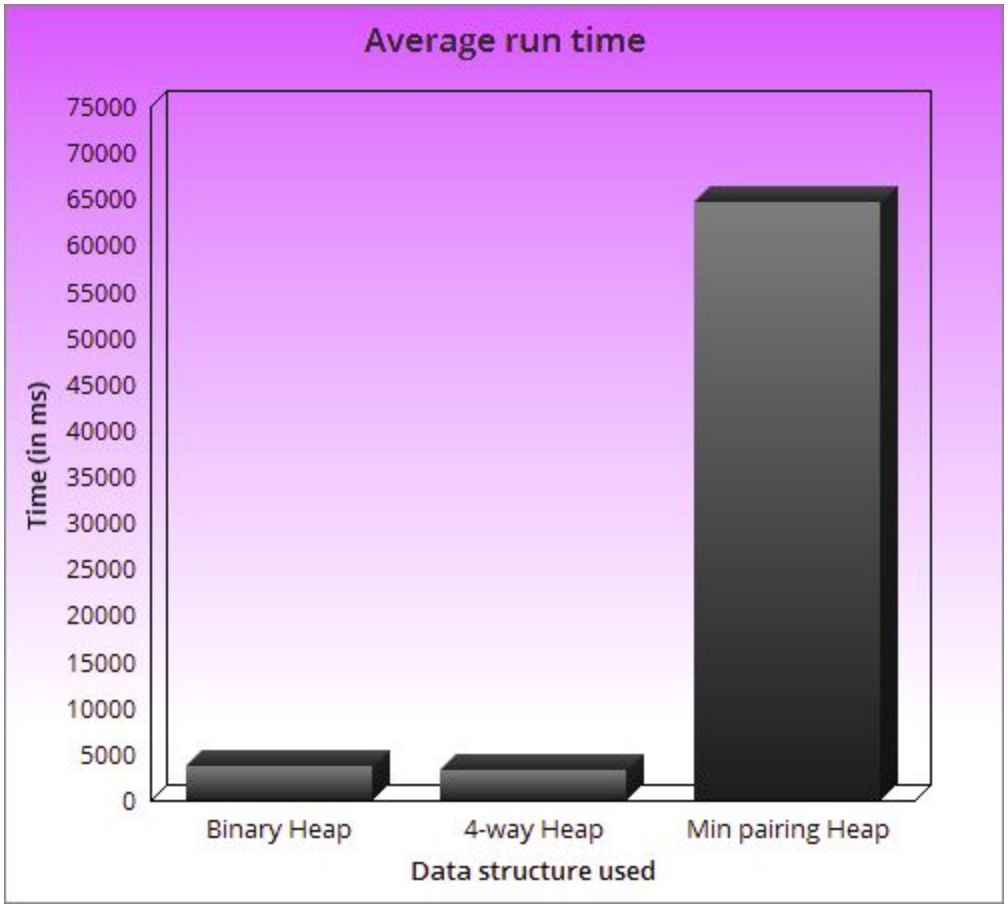
Result Comparision

Performance Analysis

We were provided with 2 input files each having different number of elements.The files were to be implemented using three different data structures namely binaryheap,pairing heap and 4waycacheoptimized heap and build the encoded.bin and decoded.bin using these three itself.

The following graph shows the performance of these different input files:


Performance Analysis
80000
75000
70000
65000
60000
55000
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Time (in ms)
1
2
3
4
5
6
7
8
9
10
Runs



Average Time to execute the given were different for the above 3 implementations.

They were as following :

1. BinaryHeapImplementation – 3684 milli sec
2. PairingHeapImplementation– 72226 milli sec
3. 4waycacheOptimized– 3294 milli sec

Conclusions

Overall, The four way cache optimized heap is the fastest of the three for generating huffman encoder and decoder :

Insert : O(log n)

Remove-min : O(log n) with log4n cache misses for average remove min

And the **decoding** algorithm has complexity **O(bn)** where b is bit length and n is the no of elements.