# Assignment 2

Algorithms

—

Samer Mohamed El Sayed Abdel Mlek
18010778

# Problem 1

## Time Analysis

$O(n^2)$

## Code

```
class node {
    int start;
    int end;
    int profit;

    public node(int start, int end, int profit) {
        this.start = start;
        this.end = end;
        this.profit = profit;
    }
}

class JobScheduling {
    static node[] nodes;
    static int n;
    public static int jobScheduling() {
        //sorting activities by end time
        Comparator<node> cx = Comparator.comparingInt(o -> o.end);
        Arrays.sort(nodes , cx);

        int[] c = new int[n];
        c[0] = nodes[0].profit;
        for (int i = 1; i < n ; i++)
        {
            //accumulated solution before processing
            c[i] = max(nodes[i].profit, c[i-1]);
            //finding the last compatible activity
            for (int j = i - 1; j >= 0; j--)
            {
                if (nodes[j].end <= nodes[i].start)
                {
                    c[i] = max(c[i], nodes[i].profit + c[j]);
                    break;
                }
            }
        }
        int max = MIN_VALUE;
        for (int i : c)
            max = max(max , i);
        return max;
    }
}
```

# Problem 2

## Analysis

### File 1

**n = 1**

Compression ratio = 50%

Compression time = 1 minutes 32 seconds

Decompression time = 1 minutes 55 seconds

**n = 2**

Compression ratio = 41%

Compression time = 0 minutes 56 seconds

Decompression time = 1 minutes 48 seconds

**n = 3**

Compression ratio = 37%

Compression time = 0 minutes 47 seconds

Decompression time = 1 minutes 58 seconds

**n = 4**

Compression ratio = 35%

Compression time = 0 minutes 52 seconds

Decompression time = 2 minutes 13 seconds

**n = 5**

Compression ratio = 37%

Compression time = 1 minutes 24 seconds

Decompression time = 3 minutes 27 seconds

**7z Compression ratio**

24%

## File 2

**n = 1**

Compression ratio = 94%

Compression time = 0 minutes 0 seconds

Decompression time = 0 minutes 0 seconds

**n = 2**

Compression ratio = 119%

Compression time = 0 minutes 0 seconds

Decompression time = 0 minutes 0 seconds

**n = 3**

Compression ratio = 182%

Compression time = 0 minutes 1 seconds

Decompression time = 0 minutes 0 seconds

**n = 4**

Compression ratio = 161%

Compression time = 0 minutes 1 seconds

Decompression time = 0 minutes 0 seconds

**n = 5**

Compression ratio = 146%

Compression time = 0 minutes 0 seconds

Decompression time = 0 minutes 0 seconds

**7z Compression ratio**

72%

## Code

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.*;

class QueueComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode h1, HuffmanNode h2)
    {
        return Integer.compare(h1.freq, h2.freq);
    }
}
public class Huffman {
    static PriorityQueue<HuffmanNode> Q = new PriorityQueue<>(new
QueueComparator());
    static HashMap<String, String> by = new HashMap<>();
    static String readPath = "";
    static String writePath = "";
    static int n = 1;
    static int m;
    static int overload;
    static boolean b;
    static boolean append = false;
    static String id = "18010778";
    static FileOutputStream out;
    static BufferedOutputStream bout;

    public static void main(String[] args) throws IOException {
        String operation = args[0];
        String path = args[1];
        long start;
        long end;
        int timeSeconds;
        int timeMins;
        if (operation.compareTo("c") == 0)
        {
            int number = Integer.valueOf(args[2]);

            //compression
            start = System.nanoTime();
            huffmanEncoding(number, path);
            end = System.nanoTime();

            //size of the file before and after compression
            Path size = Paths.get(readPath);
            long beforeBytes = Files.size(size);
            size = Paths.get(writePath);
            long afterBytes = Files.size(size);

            //printing compression ratio and time
            int ratio = (int) (((double) afterBytes / beforeBytes) * 100);
            System.out.println("Compression ratio = " + ratio + "%");

            timeSeconds = (int) ((end-start)/(1000 * 1000 * 1000));
            timeMins = 0;
```

```
            if(timeSeconds > 60) {
                timeMins = (int) ((double) timeSeconds / 60);
                timeSeconds = timeSeconds % 60;
            }
            System.out.println("Compression time = " + timeMins + " minutes "
+ timeSeconds + " seconds");


        } else if (operation.compareTo("d") == 0)
        {
            //decompression
            start = System.nanoTime();
            huffmanDecoding(path);
            end = System.nanoTime();

            //printing decompression time
            timeSeconds = (int) ((end-start)/(1000 * 1000 * 1000));
            timeMins = 0;
            if(timeSeconds > 60) {
                timeMins = (int) ((double) timeSeconds / 60);
                timeSeconds = timeSeconds % 60;
            }
            System.out.println("Decompression time = " + timeMins + " minutes
" + timeSeconds + " seconds");
        }
    }

    //initializes huffman encoding
    public static void huffmanEncoding(int number,String read) throws
IOException {
        b = true;
        readPath = read;
        n = number;
        writePath = path();
        out = new FileOutputStream(writePath);
        bout = new BufferedOutputStream(out);
        encodingRead();
        writeToStream("0\\" + m + "\\" + n + "\\");
        huffman();
        encodingWrite();

    }

    //reads from the file and constructs hashmap with frequencies
    public static void encodingRead() {
        try{
            FileInputStream in = new FileInputStream(readPath);
            BufferedInputStream bin = new BufferedInputStream(in);
            m = bin.available() % n;
            if(m == 0)
                m = n;
            byte[] bytes = new byte[n];
            while (bin.read(bytes) != -1) {
                String c = new String(bytes, StandardCharsets.ISO_8859_1);
                if (by.containsKey(c)) {
                    String v = by.get(c);
                    by.put(c, Integer.toString(Integer.parseInt(v) + 1));
```

```
                } else
                    by.put(c , Integer.toString(1));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //constructs huffman tree
    public static void huffman(){
        for (String i : by.keySet()) {
            HuffmanNode h = new HuffmanNode();
            h.c = i;
            String v = by.get(i);
            h.freq = Integer.parseInt(v);
            Q.add(h);
        }
        while (Q.size() > 1)
        {
            HuffmanNode h = new HuffmanNode();
            h.left = Q.poll();
            h.right = Q.poll();
            assert h.right != null;
            h.freq = h.left.freq + h.right.freq;
            Q.add(h);
        }
        try  {
            assert Q.peek() != null;
            encodingTraversal(Q.peek(),"" );
            writeToStream("\\");
            assert Q.peek() != null;
            if (Q.peek().left == null && Q.peek().right == null)
                by.put(Q.peek().c, "1");


        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //writes encoded data
    public static void encodingWrite(){
        try {
            FileInputStream in = new FileInputStream(readPath);
            BufferedInputStream bin = new BufferedInputStream(in);
            RandomAccessFile writer = new RandomAccessFile(writePath, "rw");

            StringBuilder s = new StringBuilder();
            byte[] bytes = new byte[n];
            int maxString = 20 * 1000 * 1000;
            StringBuilder str = new StringBuilder();
            while (bin.read(bytes) != -1) {
                String y = new String(bytes, StandardCharsets.ISO_8859_1);
                String v = by.get(y);
                s.append(v);
                while(s.length() >= 8) {
                    str.append((char) (Integer.parseInt(s.substring(0, 8),
```

```
2)));
                    s = new StringBuilder(s.substring(8));
                }
                if ( str.length() >= maxString )
                {
                    writeToStream(str.toString());
                    str.delete(0, str.length());
                }
            }
            if (str.length() > 0)
            {
                writeToStream(str.toString());
                str.delete(0, str.length());
            }
            StringBuilder str2 = new StringBuilder(String.valueOf(s));
            if (str2.length() > 0)
            {
                overload = 8 - s.length();
                int temp = overload;
                while (temp > 0)
                {
                    str2.append('0');
                    temp--;
                }
                String tempS = "";
                tempS += (char) Integer.parseInt(str2.toString(), 2);
                writeToStream(tempS);
                writer.seek(0);
                writer.write(String.valueOf(overload).charAt(0));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //sets codewords to each group of bytes
    public static void encodingTraversal(HuffmanNode h, String codeword)
throws IOException {
        if (h.c != null)
        {
            writeToStream("1" + h.c);
            by.put(h.c, codeword);
        }
        else {
            writeToStream("0");
            if (h.left != null)
                encodingTraversal(h.left, codeword + '0');
            if (h.right != null)

                encodingTraversal(h.right, codeword + '1');
        }

    }

    //writes string str to the output file
    public static void writeToStream(String str) throws IOException {
        for (int i = 0; i < str.length(); i++)
```

```
        {
            bout.write(str.charAt(i));
        }
        bout.flush();
        append = true;
    }

    //initializes huffman decoding
    public static void huffmanDecoding(String read) throws IOException {
        b = false;
        readPath = read;
        writePath = path();
        out = new FileOutputStream(writePath , append);
        bout = new BufferedOutputStream(out);
        FileInputStream in = new FileInputStream(readPath);
        BufferedInputStream bin = new BufferedInputStream(in);
        overload = decodingRead(bin);
        m = decodingRead(bin);
        n = decodingRead(bin);
        decodingTraversal(bin ,"");
        bin.skip(1);
        if (by.get("") != null)
            by.put("1",by.get(""));
        decodingWrite(bin , bin.read());
    }

    //reads statistics from the encoded file
    public static int decodingRead(BufferedInputStream bin) throws
IOException {
        int n;
        char c;
        StringBuilder number = new StringBuilder();
        do {
            c = (char) bin.read();
            if (c!='\\')
                number.append(c);

        } while (c != '\\');
        n = Integer.parseInt(number.toString());
        return n;
    }

    //constructs the hashmap with codewords and its corresponding data
    public static void decodingTraversal(BufferedInputStream bin ,String
codeword) throws IOException {
        char c = (char) bin.read();
        if (c!= '\\')
        {
            if (c == '0') {
                decodingTraversal(bin ,codeword + "0");
                decodingTraversal(bin, codeword + "1");

            }
            else if(c == '1')
            {
                StringBuilder data = new StringBuilder();
                int temp = n;
```

```
                do {
                    c = (char) bin.read();
                    data.append(c);
                    temp--;
                }
                while (temp > 0);
                by.put(codeword , data.toString());
            }
        }

    }

    //writes the decoded data to the output file
    public static void decodingWrite(BufferedInputStream bin, int pointer)
throws IOException {
        int maxString = 20 * 1000 * 1000;
        StringBuilder str = new StringBuilder();
        StringBuilder keyBuilder = new StringBuilder();
        do {
            char reader = (char) pointer;
            pointer = bin.read();
            StringBuilder binary = new
StringBuilder(Integer.toBinaryString(reader));
            int zeros = 8 - binary.length();
            while (zeros > 0) {
                binary.insert(0, "0");
                zeros--;
            }
            if (pointer == -1) {
                zeros = 8 - overload;
                binary = new StringBuilder(binary.substring(0, zeros));
            }
            keyBuilder.append(binary);
            String value;
            int i = 1;
            do {
                value = by.get(keyBuilder.substring(0, i));
                if (value != null) {
                    keyBuilder = new StringBuilder(keyBuilder.substring(i));
                    if (pointer == -1 && (keyBuilder.length() == 0)) {
                        str.append(value.substring(0, m));
                    } else {
                        str.append(value);
                    }
                    i = 1;
                } else
                    i++;
            } while (i <= keyBuilder.length());
            if ( str.length() >= maxString )
            {
                writeToStream(str.toString());
                str.delete(0, str.length());
            }
        } while (pointer != -1);
        if (str.length() > 0)
        {
            writeToStream(str.toString());
```

```
            str.delete(0, str.length());
        }
    }

    //sets the path of the write file
    public static String path()
    {
        String location = "";
        String fileName = "";
        for (int i = readPath.length() - 1;i > 0;i--)
        {
            if (readPath.charAt(i) == '\\')
            {
                fileName = readPath.substring(i + 1);
                location = readPath.substring(0, i+1);
                break;
            }
        }
        if(b)
            fileName = id + "." + n + "." + fileName + ".hc";
        else
            fileName = "extracted." + fileName.substring(0,fileName.length()-
3);

        return location + fileName;
    }



}
class HuffmanNode{
    HuffmanNode left;
    HuffmanNode right;
    int freq;
    String c;
}
```