Programming Problem: Heap Tunable Branching Factor

```
import java.util.ArrayList;
import java.util.Scanner;

public class heap {

	static class KeyValue{
		int key;
		int value;

		KeyValue(int key, int value){
			this.key = key;
			this.value = value;
		}
	}

	static ArrayList<KeyValue> list;
	static int increaseFactor;
	static int branchingFactor;
	static int keyComparisons = 0;
	static boolean debug = false;

	static class RemovedData{
		int key;
		int value;
	}

	static int parent(int i) {
		if(i%branchingFactor == 0)
			return (int) Math.floor((i/branchingFactor) - 1);
		else
			return (int) Math.floor(i/branchingFactor);
	}

	static void percolate(int i) {
		if(i == 0)
			return;
		int p = parent(i);
		keyComparisons++;
		if(list.get(i).key < list.get(p).key) {
			KeyValue temp = list.get(i);
			list.set(i, list.get(p));
			list.set(p, temp);
			percolate(p);

		}
	}
```

```
	static void heapify(int i) {
		//System.out.println("DEBUG: Heapify started");
		int left, shift;
		KeyValue temp;
		shift = i << 1;
		while((shift + 1 + (increaseFactor*i)) < list.size()) {

			left = shift + 1 + (increaseFactor*i);
			int min = list.get(left+0).key;
			int minj = 0;

			for(int j=1; j<branchingFactor && left+j <
list.size(); j++) {
				int k = list.get(left+j).key;
				keyComparisons++;
				if(k < min) {

					min = k;
					minj = j;
				}
			}
			keyComparisons++;
			if(list.get(i).key > list.get(left+minj).key) {
				temp = list.get(i);
				list.set(i, list.get(left+minj));
				list.set(left+minj, temp);
				i = left+minj;
				shift = i << 1;

			}
			else
				break;
		}
	}

	static void removeMin(RemovedData d) {
		KeyValue temp = list.get(0);
		list.set(0, list.get(list.size()-1));
		list.set(list.size()-1, temp);
		temp = list.remove(list.size()-1);
		heapify(0);
		d.key = temp.key;
		d.value = temp.value;

	}

	static void insertValue(int key, int value) {
		KeyValue k = new KeyValue(key, value);
		list.add(k);
		percolate(list.size()-1);

	}
```

```
	public static void main(String[] args) {
		// TODO Auto-generated method stub

		if(args.length == 0)
			branchingFactor = 2;
		else if(args.length > 1) {
			System.out.println("ERROR: More than one parameter
provided.");
			return;
		}
		else{
			if(args[0].matches("\\d+")) {
				if
((int)(Math.ceil((Math.log(Integer.parseInt(args[0])) / Math.log(2))))
!= (int)(Math.floor(((Math.log(Integer.parseInt(args[0])) /
Math.log(2)))))) {
					System.out.println("ERROR: Branching factor
not a power of 2.");
					return;
				}
				else
					branchingFactor =
Integer.parseInt(args[0]);
			}
			else {
				System.out.println("ERROR: Invalid input.");
				return;
			}
		}

		//long start = System.nanoTime();


		/*	Method 1	*/
		/*
		int i=1, sum=0;

		int temp = 2;
		System.out.println(branchingFactor);
		while(temp<branchingFactor) {
			sum+=temp;
			i++;
			temp = (int) Math.pow(2,i);
		}
		increaseFactor = sum;
		*/

		/*	Method 2	*/
		if(branchingFactor == 2)
			increaseFactor = 0;
		else
```

```
            increaseFactor = branchingFactor - 2;

        list = new ArrayList<KeyValue>();
        int num1, num2;
        String line = new String();
        Scanner sc = new Scanner(System.in);

        while(true) {
            try {
                line = sc.nextLine();
                if(line.length() == 0)
                    break;
                String[] nums = line.split(" ");
                num1 = Integer.parseInt(nums[0]);
                if(num1 != -1) {
                    num2 = Integer.parseInt(nums[1]);
                    insertValue(num1,num2);
                }
                else {
                    RemovedData d = new RemovedData();
                    removeMin(d);
                    System.out.println(d.key+" "+d.value);
                }
            }
            catch(Exception e) {
                break;
            }
        }

        //long end = System.nanoTime();

        /*

        System.out.println(increaseFactor);
        for(int j=0; j<list.size(); j++)
        System.out.println(list.get(j).key);

        */
        //System.out.println("runtime: "+((end-start)/1000000)+"
ms");
        System.out.println("key comparisons: "+keyComparisons);


    }

}
```