Übungsserie 1 - Datenstrukturen & Algorithmen

Dominik Hiltbrunner, Michael Kohler

Aufgabe 1

a) Maximum und Minimum des Input-Arrays

b)

```
public class MaxMin {

   public static void main(String[] args) {
            int[] inputArray = null; // bzw. Input des Users
            int[] result = minMax(inputArray);
   }

   private static int[] minMax(int[] a) {
            int x = a[0];
            int y = a[0];
            for (int i = 1; i <= a.length - 1; i++) {
                     if (x > a[i]) {
                              x = a[i];
                     }

                     if (y < a[i]) {
                              y = a[i];
                     }
                     System.out.println("x: " + x + ", y: " + y);
            }
            int[] result = new int[] { x, y };
            return result;
   }

}
```

c) 2n

d)

```
var x , y : int; a : array 1..n of int;
input a;
for i = 0 to n−2 do
        if a [ i ] > a [ i + 1 ] then
            if a [ i + 1 ] < x then
                    x := a [ i + 1];
            fi
            if a [ i ] > y then
                    y := a [ i ];
            fi
        else
            if a [ i ] < x then
                    x := a [ i ];
            fi
            if a [ i + 1 ] > y then
                    y := a [ i + 1 ];
            fi
        fi
        i = i + 2;
od
output : x , y
```

e) Nein, da in n auf jeder Position der gesuchte Wert sein kann. Daher müssen alle Werte verglichen werden.

Aufgabe 2

```
import java.util.ArrayList;

public class Einbrecher {

   public class Item {
            private int weight = 0;
            private int value = 0;

            public Item(int weight, int value) {
                     this.weight = weight;
                     this.value = value;
            }

            public int getWeight() {
                     return this.weight;
            }

            public int getValue() {
                     return this.value;
            }
   }

   ArrayList<Item> items = new ArrayList();
   int maxWeight;
   String maxBitmask = new String();
   int bestValue = 0;
   String bestBitmask = "";
   long runTime = 0;

   public Einbrecher(String[] args) {
            this.maxWeight = Integer.parseInt(args[0]);

            for (int i = 1; i < args.length; i += 2) {
                     this.items.add(new Item(Integer.parseInt(args[i]), Integer.parseInt(args[i + 1])));
            }

            for (int i = 0; i < this.items.size(); i++) {
                     this.maxBitmask += "1";
            }
   }

   private void calculateBestPermutation() {
            long startTime = System.currentTimeMillis();
            int maxRange = (int) (Math.pow(2, this.items.size()) - 1);
            for (int i = 0; i < maxRange; i++) {
                     String currentBitmask = Integer.toBinaryString(i);
                     while (currentBitmask.length() < this.maxBitmask.length()) {
```

```
                        currentBitmask = "0" + currentBitmask;
                }
                int currentValue = 0;
                int currentWeight = 0;
                int index = 0;
                while (index < this.items.size()) {
                        if (currentBitmask.charAt(index) == '1') {
                                currentWeight += this.items.get(index).getWeight();
                                currentValue += this.items.get(index).getValue();
                        }
                        index++;
                }
                if (currentWeight <= this.maxWeight && currentValue > this.bestValue) {
                        this.bestBitmask = currentBitmask;
                        this.bestValue = currentValue;
                }
        }
        long endTime = System.currentTimeMillis();
        this.runTime = endTime - startTime;
    }

    @Override
    public String toString() {
        return this.bestBitmask + "\nValue " + this.bestValue + "\nTime needed: " + this.runTime + " ms";
    }

    public static void main(String[] args) {
        if (args.length % 2 != 1) {
                System.out.println("Wrong usage, please input:");
                System.out.println("[max weight] [weight item1] [value item1] [weight item2] [value item2] ...");
                return;
        }

        Einbrecher hans = new Einbrecher(args);
        hans.calculateBestPermutation();
        System.out.println(hans.toString());
    }

}
```

Aufgabe 3

a) x^n

b) Im Worstcase werden immer 2 Multiplikationen durchgeführt (alle Zahlen ungerade). Betrifft Fälle mit $n = 2^a-1$ ($a \in \mathbb{N}$)

n	**Anzahl Multiplikationen**
3	2
7	4
15	6
31	8
63	10

-> worstcase Komplexität = $(\log_2(n+1)*2)-2$

Aufgabe 4

```
int binSearch(int[] array, int item, int min, int max)
{
  if (min > max)
    return -1;
  else
    {
      int pivot = Math.round((min+max)/2);
      if (array[pivot] > item)
        return binSearch(array, item, min, pivot-1);
      else if (array[pivot] < item)
        return binSearch(array, item, pivot+1, max);
      else
        return pivot;
    }
}
```

Aufgabe 5

Bubble Sort und Insertion Sort mit einer Komplexität von O(n). Grund: beide führen pro Durchlauf nur einen Vergleich durch.

Aufgabe 6

Merge Sort und Quick Sort, da diese für diesen Fall die einzigen Algorithmen mit Komplexität O(n*log(n)) sind. Die anderen haben jeweils eine Komplexität von $O(n^2)$.

Aufgabe 7

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

public class QuickSort {

   public static void main(String[] args) {
          final int MIN = 0;
          final int MAX = 1000;

          int[] rndArray = new int[MAX];

          for (int i = 0; i < MAX; i++) {
                rndArray[i] = (int) (Math.random() * MAX) + 1;
          }
          HashMap<Long, Integer> map = new HashMap<>();
          for (int M = 0; M < MAX; M++) {
                int[] tempArray = rndArray.clone();
                long startTime = System.nanoTime();
                quickSort(tempArray, M, MIN, MAX - 1);
                long stopTime = System.nanoTime();

                sortCheck(tempArray);
                System.out.println("M = " + M + ": " + (stopTime - startTime)
                            + " ns");

                map.put(stopTime - startTime, M);
          }

          Iterator<Long> iter = map.keySet().iterator();
          Long min = Long.MAX_VALUE, tmp;
          while (iter.hasNext()) {
                tmp = iter.next();
                if (min > tmp)
                      min = tmp;
          }

          System.out.println("MINIMUM: " + map.get(min) + ", " + min + " ns");

   }

   private static void quickSort(int[] rndArray, int M, int left, int right) {

          if ((left + right) / 2 <= M) {
                insertionSort(rndArray, left, right + 1);
                return;
          }
```

```
		int i = left, j = right;
		int pivot = rndArray[left + (right - left) / 2];

		while (i <= j) {
			while (rndArray[i] < pivot) {
				i++;
			}
			while (rndArray[j] > pivot) {
				j--;
			}

			if (i <= j) {
				exchange(rndArray, i, j);
				i++;
				j--;
			}
		}
		if (left < j)
			quickSort(rndArray, M, left, j);
		if (i < right)
			quickSort(rndArray, M, i, right);

	}

	private static void exchange(int[] numbers, int i, int j) {
		int temp = numbers[i];
		numbers[i] = numbers[j];
		numbers[j] = temp;
	}

	private static void insertionSort(int[] rndArray, int left, int right) {
		int temp;
		for (int i = left; i < right; i++) {
			temp = rndArray[i];
			int j = i;
			while (j > 0 && rndArray[j - 1] > temp) {
				rndArray[j] = rndArray[j - 1];
				j--;
			}
			rndArray[j] = temp;
		}

	}

	private static void sortCheck(int[] rndArray) {
		List<Integer> intList = new ArrayList<Integer>();
		for (int index = 0; index < rndArray.length; index++) {
			intList.add(rndArray[index]);
		}
```

```
		System.out.println("sorted: " + isSorted(intList));
	}

	public static <T extends Comparable> boolean isSorted(List<T> listOfT) {
		T previous = null;
		for (T t : listOfT) {
			if (previous != null && t.compareTo(previous) < 0)
				return false;
			previous = t;
		}
		return true;
	}

}
// END OF CODE
```

Mit 1000 Datensätzen ist kein empirisch signifikanter Wert für M zu finden.

Aufgabe 8

```
import java.util.Arrays;

public class MergeSortHilfsfeld {

   public int[] sort(int[] array, int left, int right) {
            if (left < right) {
                     int mid = (left + right) / 2;

                     array = sort(array, left, mid);
                     array = sort(array, mid + 1, right);
                     array = merge(array, left, mid, right);
            }
            return array;
   }

   private int[] merge(int[] array, int left, int mid, int right) {
            int[] hilfsArray = new int[array.length];
            int i, j;
            for (i = left; i <= mid; i++) {
                     hilfsArray[i] = array[i];
            }
            for (j = mid + 1; j <= right; j++) {
                     hilfsArray[right + mid + 1 - j] = array[j];
            }
            i = left;
            j = right;
            for (int k = left; k <= right; k++) {
                     if (hilfsArray[i] <= hilfsArray[j]) {
                              array[k] = hilfsArray[i];
                              i++;
                     } else {
                              array[k] = hilfsArray[j];
                              j--;
                     }
            }
            return array;
   }

   public static void main(String[] args) {
            int[] randomArray = new int[10];
            for (int i = 0; i < randomArray.length; i++) {
                     randomArray[i] = (int) (Math.random() * randomArray.length + 1);
            }
            MergeSortHilfsfeld ms = new MergeSortHilfsfeld();
            int mid = randomArray.length / 2;
            int[] randArray1 = Arrays.copyOfRange(randomArray, 0, mid);
            int[] randArray2 = Arrays.copyOfRange(randomArray, mid,
                              randomArray.length);
            int[] arr1 = ms.sort(randArray1, 0, randArray1.length - 1);
```

```
		int[] arr2 = ms.sort(randArray2, 0, randArray2.length - 1);
		int[] arr = finalSort(arr1, arr2);
		for (int i = 0; i < arr.length; i++) {
			System.out.println(i + 1 + ": " + arr[i]);
		}
	}

	private static int[] finalSort(int[] arr1, int[] arr2) {
		int[] result = new int[arr1.length + arr2.length];
		int i = 0, j = 0;
		while (i < arr1.length || j < arr2.length) {
			if (i >= arr1.length) {
				result[i + j] = arr2[j++];
			}
			else if (j >= arr2.length){
				result[i + j] = arr1[i++];
			}
			else {
				if (arr1[i] <= arr2[j]) {
					result[i + j] = arr1[i++];
				} else {
					result[i + j] = arr2[j++];
				}
			}
		}
		return result;
	}
}
```

Aufgabe 9
n*log(n)