TECHNICAL UNIVERSITY OF APPLIED SCIENCES ULM

# Assignment #2

Student names and Group: *Petar Stoyanov & Eetu Hyvärinen - Group 08*

Course: *Algorithms and Data Structures* – Professor: *Prof. Schied*
Due date: *November 14th, 2019*

**Exercise 2.1**

(*a*) Which of the following assertions are a loop-invariant of the while loop in lines 7 to 10 (with brief justification):

(*1*) k < 0

(*2*) k $\geq$ 0

(*3*) res $\geq$ 0

(*4*) res = x $\cdot(n - k)$

(*b*) Use a suitable loop invariant to justify that multiply(n, x) calculates n $\cdot x$ for values n $\geq$ 0.

(*c*) Provide a suitable termination function for the loop that can be used to show that the program terminates for all values n $\geq$ 0.

**Answer.**

(*a*) (*1*) not a loop invariant - the loop stops at k = 0

(*2*) loop invariant - the loop stops at k = 0

(*3*) not a loop invariant - x can be a negative number

(*4*) loop invariant - tested with x begin a positive, a negative and a decimal number - was correct for 4 iterations

(*b*) -

(*c*) k $\leq$ 0

## Exercise 2.3

Which of the following statements apply? Justify the answers briefly.
(*a*) $4n^3 + 999n^2 = \Omega(n^4)$

(*b*) $5n^3 + 321n^2 = \Theta(n^2)$

(*c*) $\log_{10}(3n) = \Theta(\log_2(n))$

(*d*) $3^{n+1} = \Theta(3^n)$

(*e*) $77n^2 + 3 \cdot \log_5(n) + \log_2(6n) = O(n^2 + \log_{10}(n))$

**Answer.**

(*a*) false, $n^4$ is lower bound and $n^3$ grows slower than it

(*b*) false, $n^3$ on the left side grows faster than $n^2$ on the right

(*c*) true,

$$log_{10}(3n) = log_2(3n)/log_2(10)$$

remove $log_2(10)$, it's a constant

$$log_2(3n) = \Theta log_2(n), \text{ 3 is also a constant}$$

$$log_2(n) = \Theta log_2(n) \text{which is correct}$$

(*d*) true, $3 \times 3^n = \Theta(3^n)$

(*e*) false,

$$n \cdot log_{10}(n)/log_{10}(5) == O(n \cdot log(n))$$

$$log_{10}(6n)/log_{10}(2) == O(log(n))$$

In other words: $n^2 + n \cdot log(n) + log(n) = O(n^2 + log(n))$

Left side grows slightly faster than the right

## Exercise 2.5

Determine the runtime complexity for the following pieces of code, depending on variable n as problem size.

**Answer.** Runtime complexity of the code is calculated as
**Outer loop Complexity * Inner loop Complexity**

(*a*) $O(n) * O(n) = O(n^2)$

(*b*) $O(\log(n)) * O(n) = O(n \cdot \log(n))$

(*c*) $O(n) * O(3n) = O(n+3n) = O(n)$

(*d*) $O(\log(n)) * O(n) = O(n \cdot \log(n))$

### Exercise 2.7

Analyze the runtime complexity of the following method **processArray** depending on the length n of the array **arr**. You can assume that the length of the array is a power of two, i. e. $n = 2^m$ .

**Answer.**

$$\text{line 5 for loop is} \frac{n}{2} \text{ iterations} = O(n)$$

$$\text{line 6 for loop is the same} \frac{n}{2} \text{ iterations} = O(n)$$

$$\text{line 7 general calculations} = O(1)$$

$$\text{line 11 for loop is n iterations} = O(n)$$

$$\text{line 13 while loop is log n iterations} = O(log(n))$$

$$\text{line 14 and 15 general calculations} = O(1)$$

$$(line5 * line6 * line7) + (line11 * line13 * [line12 + line15] * line14)$$

$$(O(n) * O(n) * O(1)) + (O(n) * O(log(n)) * O(1))$$

$$O(n^2) + O(n \cdot log(n))$$

$$\text{Final runtime complexity: } O(n^2),$$

$$\text{because it has worse complexity than } n \cdot log(n)$$

### Exercise 2.9

(*a*) Write an efficient method in class **TaskDemo** that rearrange all tasks in the list in such a way, that all tasks with priority **HIGH** are at the beginning of the list, all tasks with priority **MEDIUM** in the middle and all tasks with priority **LOW** at the end of the list. Use the methods **getPriority(i)** and **swap(i, j)** to rearrange the taks. Class **TaskList** must not be changed! You can check with method **isOrdered()** whether the tasks are correctly arranged in the order **HIGH – MEDIUM – LOW**.

(*b*) Measure the runtime for reordering task lists of sizes n = 100, n=1 000,..., n= 10 000 000. A measurement method **runtime(int n)** is available in class **TaskDemo** (see Moodle). Which runtime behavior can be recognized? Does your implementation succeed to reorder a randomly generated list of 10 000 000 tasks in less than 5 seconds (executed with option -Xint)?

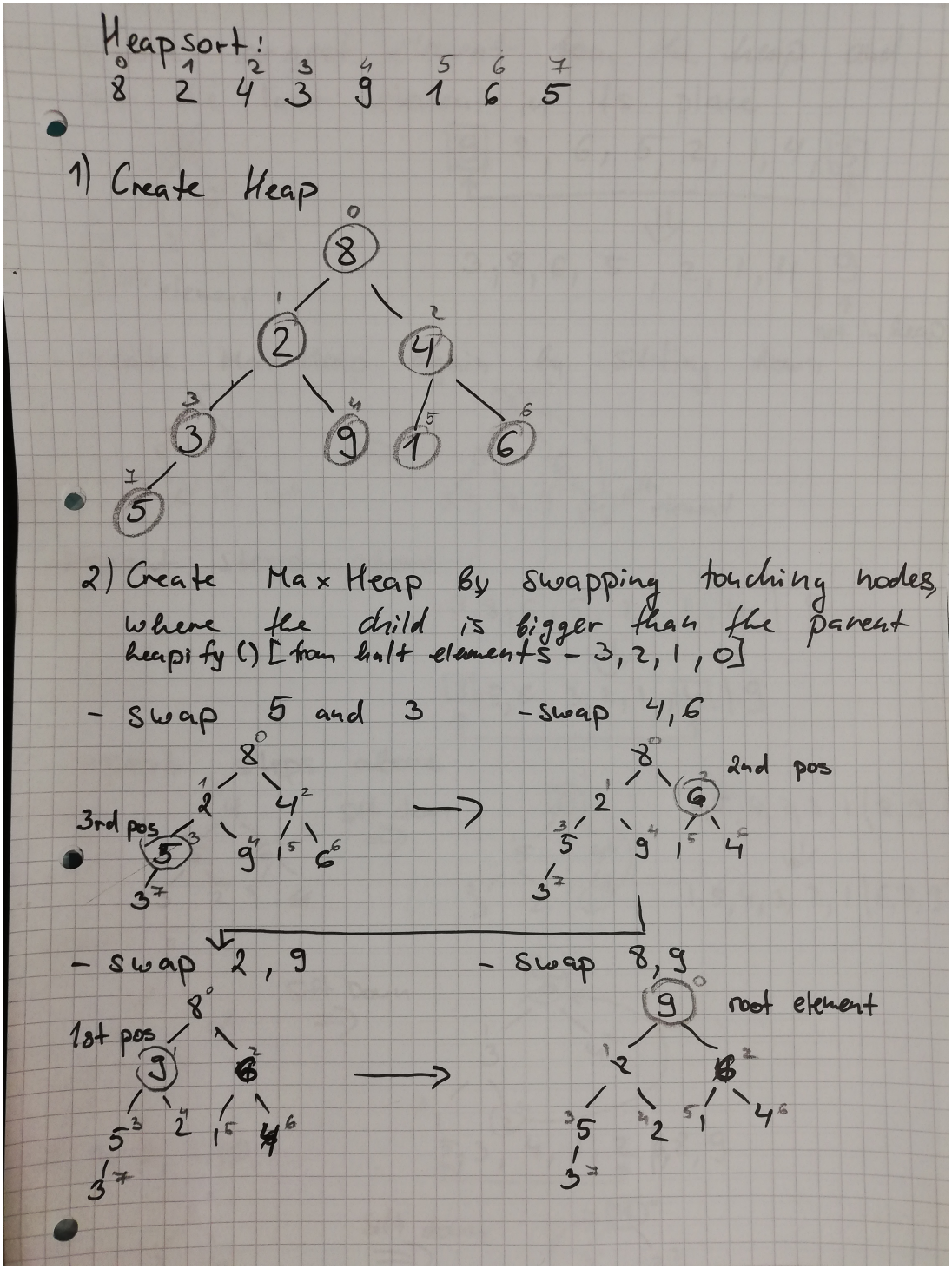
**Answer. Code:**

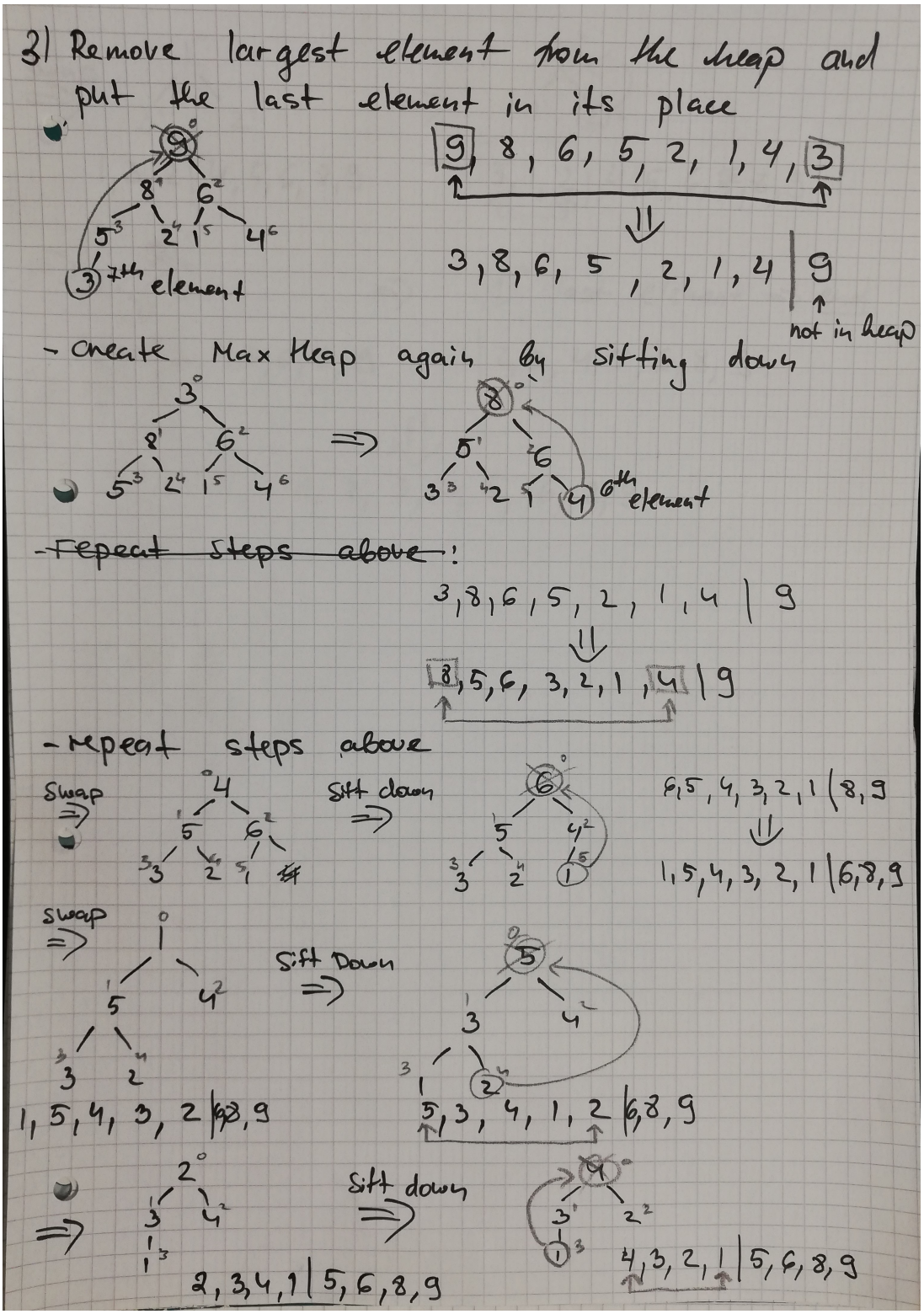
```
//Code...
//Code...
//Code...
//Code...
```

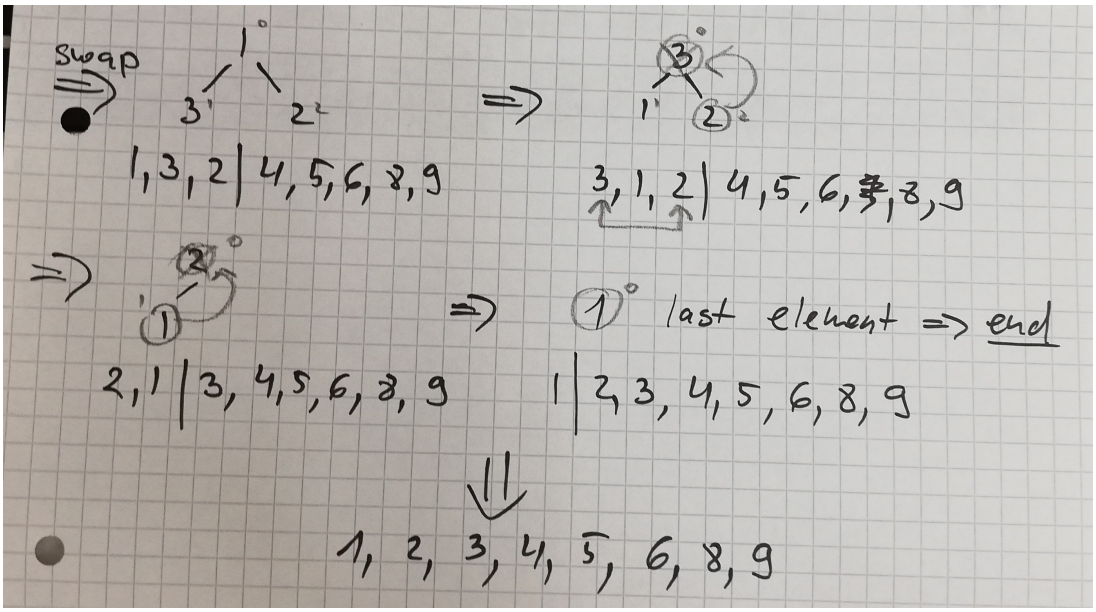
**Exercise 2.11**

| Use the following sequence of numbers as an example to explain how **Heapsort** works:<br>8, 2, 4, 3, 9, 1, 6, 5<br>Indicate the essential intermediate steps |
|---|

**Answer.**




3) Remove largest element from the heap and put the last element in its place
9, 8, 6, 5, 2, 1, 4, 3
3, 8, 6, 5, 2, 1, 4 | 9
3th element
not in heap
- create Max Heap again by sifting down
- Repeat steps above!
3, 8, 6, 5, 2, 1, 4 | 9
8, 5, 6, 3, 2, 1, 4 | 9
- repeat steps above
Swap
Sift down
6, 5, 4, 3, 2, 1 | 8, 9
1, 5, 4, 3, 2, 1 | 6, 8, 9
Swap
Sift Down
1, 5, 4, 3, 2 | 6, 8, 9
5, 3, 4, 1, 2 | 6, 8, 9
Sift down
2, 3, 4, 1 | 5, 6, 8, 9
4, 3, 2, 1 | 5, 6, 8, 9


swap
1,3,2 | 4,5,6,8,9
3,1,2 | 4,5,6,8,9
2,1 | 3,4,5,6,8,9
1 | 2,3,4,5,6,8,9
last element => end
1, 2, 3, 4, 5, 6, 8, 9