Exercise sheet 2

February 18, 2020

Exercise 1 *(5 points)*
Proof with the aid of complete induction that the following holds for all natural numbers n:

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

Exercise 2 *(5 points)*
Proof with the aid of complete induction that the following holds for all natural numbers $d \in \mathcal{N}$:

$$\sum_{i=1}^{d} (2^{d-i} \cdot i) \leq 2^{d+1} - d - 2$$

Exercise 3 *(10 points)*
Implement the *Heapsort* algorithm as described in the first two lectures. In order to make the algorithm work "in-place" (i.e. the output is in the same array as the input) you have to implement a max heap where you can swap the largest element to the end of the unsorted array part (as seen in the YouTube video from lecture 1).
Write a unit test for each method. Each test should check at least one non-trivial input example. Also test if your algorithm e.g. works with an even or uneven number of elements. If critical boundary cases can be checked easily (e.g. empty input case) please check those too.
Generate a *Heapsort* runtime plot $T(n)$ with different input sizes n as shown in the lecture for *MinSort*. Choose appropriate sizes to get a meaningful plot with enough data points and reasonable runtime (at most 1 min).

Hints:

- In Python, the list data type is the usual way to represent arrays. A list is a mutable sequence of arbitrarily typed objects. Mutable means that you can e.g. change the objects stored in the list after instantiation or change its size. The list itself actually only stores the references to these objects. Arbitrary means that objects of different data types can be stored in the list (e.g. of integer, string, float or more complex data types).
- Python uses call-by-reference when passing parameters to functions. This means that a reference to the object is passed instead of the object itself. If the reference is changed inside the function, the content of the object outside the function will also be changed. Call-by-reference is especially helpful if you are passing huge amounts of data, such that the data itself does not need to be copied, resulting in less memory usage and better runtime. In our example, this could be a potentially very long list which is given to the sorting function. By using call-by-reference we make sure to avoid performance issues in such a case.
- When implementing the *Heapsort* algorithm, you could e.g. use the following functions:
  - heap_sort(array)
  - heapify(array)
  - repair_heap(array, start_index, heap_size)

Commit

Commit your solutions inside a new sub folder **uebungsblatt_02** into your SVN repository. As for exercises 1, 2 and the runtime plot, commit your solutions in PDF format. For exercises 1, 2 you can do it by hand and scan them in or e.g. use LaTeX for the formulas. Likewise commit a text file called **erfahrungen.txt** into the folder. Describe your experience with the exercise sheet in a few sentences: Was it manageable for you? How much time did it take you? Did you have problems with specific parts?