



# Algorithms and Datastructures

## Sample Solution Exercise Sheet 13

### Exercise 1: (Binary) Heaps and Heapsort (12 Bonus Points)

- (a) Implement a *binary heap* using the *array implementation* from the lecture. The heap should support the following functions: `create`, `insert` (eines key-value pairs), `get_min` and `delete_min`. You may use the template `heap.py`. (5 Points)

*Hint:* To implement `delete_min` efficiently one overwrites the root with the last element of the heap and then deletes the last element. Afterwards one has to repair the min-heap property.

- (b) Implement the heapsort algorithm by using your implementation from the previous task.<sup>1</sup> Explain the  $O(n \log n)$  runtime of heapsort.

Argue why there can't be a heap implementation where `insert`, `get_min` and `delete_min` have all constant runtime. (3 Points)

- (c) In this task we consider *ternary heaps*. They are similar to binary heaps with the difference that each parent node may have 3 children. We also have that the underlying tree is filled up with nodes from 'top to bottom' and 'left to right'.

Give the minimal and maximal number of nodes of a ternary heap of depth  $d$ . (1 Point)

Assume we use an array implementation for ternary heaps<sup>2</sup>, starting with index 1 (not 0). Let  $i$  be the index of a node  $v$  that is neither the root nor a leaf. What are the indices of  $v$ 's parent and its three children? (3 Points)

### Sample Solution

- (a) See `heap.py`

- (b) Heapsort inserts  $n$  elements into the heap. In a second loop, alternating `get_min` and `delete_min` empties the heap and retrieving the smallest element at any time. The runtimes of `insert` and `delete_min` are logarithmic in the number of elements while `get_min` is constant:

$$\sum_{i=1}^n \underbrace{O(\log n)}_{\text{insert}} + \sum_{i=1}^n \left( \underbrace{O(1)}_{\text{get}} + \underbrace{O(\log n)}_{\text{delete}} \right) = O(n \log n)$$

Why these 3 operations can not be constant: Assume all of them would be constant, then heapsort would sort an array of  $n$  arbitrary numbers in time  $O(n)$ . This is a contradiction to the fact that any comparison-based sorting algorithm takes at least  $\Omega(n \log n)$  time.

<sup>1</sup>If you did not solve the previous task, you may use `heapq`. In `heapq`, `heappush` equals the `insert` and `heappop` the `delete_min` operation from the lecture. `heappush` and `heappop` can be applied on Python-lists (for more detail see [here](#)).

<sup>2</sup>Similar to the array implementation of binary heaps on slide 26 in lecture 9.

(c) **Min:** Tee is complete up depth  $d - 1$  but contains just 1 node in depth  $d$ :

$$1 + \sum_{i=0}^{d-1} 3^i = 1 + \frac{3^d - 1}{3 - 1} = \frac{3^d + 1}{2}$$

**Max:** Tee is complete up depth  $d$ :

$$\sum_{i=0}^d 3^i = \frac{3^{d+1} - 1}{3 - 1} = \frac{3^{d+1} - 1}{2}$$

**Index left child:**  $3 \cdot i - 1$

**Index middle child:**  $3 \cdot i$

**Index right child:**  $3 \cdot i + 1$

**Index parent:**  $\lfloor \frac{i+1}{3} \rfloor$

## Exercise 2: Hashing

(8 Bonus Points)

- (a) Let  $h(s, j) := h_1(s) - 2j \bmod m$  and  $h_1(x) := x + 2 \bmod m$ . Insert the keys 51, 13, 21, 30, 23, 72 (in the given order) into a hash table of size  $m = 7$  by using the hash function  $h$  and *linear probing* for collision resolution. (The following table should show the final state after inserting all keys.) (1 Point)

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

- (b) Assume we would like to insert the sequence of numbers from part a) in a table of size  $m = 7$  by using *quadratic probing*. Which of the following hash functions would be the better choice? Explain your answer.

- $h_1(x, i) := x + 6i + 2i^2 \bmod m$
- $h_2(x, i) := x + i + 4i^2 \bmod m$

Insert the keys by using the better hash function into the following table.

(2 Points)

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

- (c) Let  $h(s, j) := h_1(s) + j \cdot h_2(s) \bmod m$  with  $h_1(x) = x \bmod m$  and  $h_2(x) = 1 + (x \bmod (m - 1))$ . Insert the keys 28, 59, 47, 13, 39, 69, 12 in a hash table of size  $m = 11$  by using *double-hashing* for collision resolution. (2 Points)

|   |   |   |   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|---|---|---|----|
|   |   |   |   |   |   |   |   |   |   |    |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

- (d) Given the hash functions  $h_1(x) := x + 2 \bmod m$  and  $h_2(x) := 3x \bmod m$  with  $m = 7$ , find three pairwise distinct keys  $u, v, w \in \mathbb{N}$  such that  $h_1(u) = h_1(v) = h_1(w) \neq h_2(u) = h_2(v) = h_2(w)$ . Insert  $u$  and  $v$  into the following table by using *Cuckoo Hashing*.

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

If we also insert  $w$ , we obtain a cycle. To avoid this, we apply a rehash by increasing the table's size to  $m' = 11$  and use two new hash functions  $h'_1$  and  $h'_2$ . Give two distinct functions  $h'_1$  and  $h'_2$  of the form  $(ax \bmod m')$  with  $a \not\equiv 0$  such that  $u, v$  and  $w$  can be inserted into the new table (i.e., that no cycle is created). (3 Points)

## Sample Solution

- (a)

|    |    |    |    |    |    |   |
|----|----|----|----|----|----|---|
| 30 | 13 | 21 | 72 | 51 | 23 |   |
| 0  | 1  | 2  | 3  | 4  | 5  | 6 |

(b)  $h_2$  ist not suitable, since there is no free array position for 23:

$$h_2(51, 0) = 2$$

$$h_2(13, 0) = 6$$

$$h_2(21, 0) = 0$$

$$h_2(30, 6) = 5$$

$$h_2(23, 0) = 2$$

$$h_2(23, 1) = 0$$

$$h_2(23, 2) = 6$$

$$h_2(23, 3) = 6$$

$$h_2(23, 4) = 0$$

$$h_2(23, 5) = 2$$

$$h_2(23, 6) = 5$$

Table filled with  $h_2$ :

|    |    |    |    |   |    |    |
|----|----|----|----|---|----|----|
| 21 | 23 | 51 | 30 |   | 72 | 13 |
| 0  | 1  | 2  | 3  | 4 | 5  | 6  |

(c)

|   |    |    |    |    |    |    |    |   |   |    |
|---|----|----|----|----|----|----|----|---|---|----|
|   | 69 | 13 | 47 | 59 | 39 | 28 | 12 |   |   |    |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8 | 9 | 10 |

(d) We choose  $u := 2$ ,  $v := 9$  and  $w := 16$ . Thus, we have  $h_1(u) = h_1(v) = h_1(w) = 4 \neq 6 = h_2(w) = h_2(v) = h_2(u)$ .

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   | v |   | u |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

As new functions we can choose for instance  $h'_1(x) = x \bmod 11$  and  $h'_2(x) = 2x \bmod 11$ .