Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer
François Hublet, Goran Zuzic, Tommaso d'Orsi, Jingqiu Ding

24 October 2022

Algorithms & Data Structures Exercise sheet 5 HS 22

The solutions for this sheet are submitted at the beginning of the exercise class on 31 October 2022.

Exercises that are marked by * are "challenge exercises". They do not count towards bonus points.

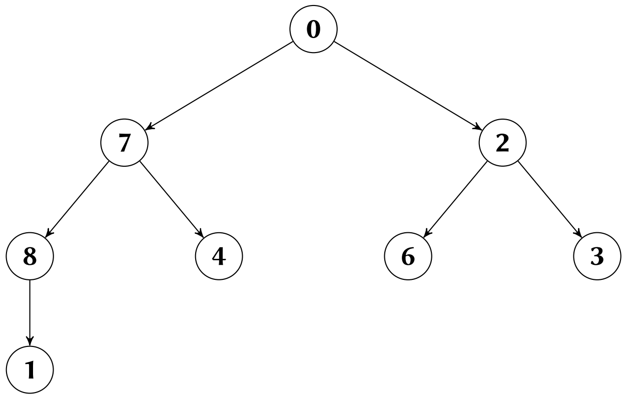
You can use results from previous parts without solving those parts.

Exercise 5.1 *Heapsort* **(1 point)**.

Given the array $[0, 7, 2, 8, 4, 6, 3, 1]$, we want to sort it in ascending order using Heapsort.

(a) Draw the tree interpretation of the array as a heap, before any call of RestoreHeapCondition.

Solution:

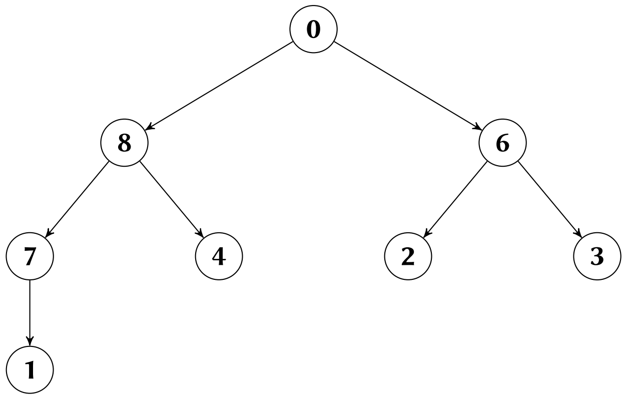


(b) In the lecture you have learned a method to construct a heap from an unsorted array (see also pages 35–36 in the script). Draw the resulting max heap if this method is applied to the above array.

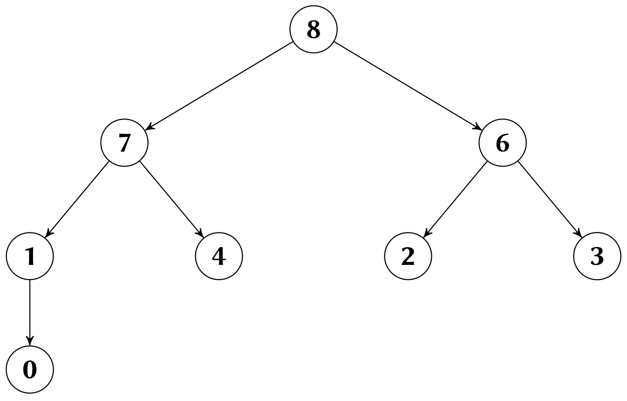
Solution:

We start from the heap drawn above. The root of the heap is at level 0. Heapifying the subtree with root at level 2 leaves the heap unchanged.

Then, heapifying the subtrees with roots at level 1 yields:



Finally, heapifying the subtree at the root node yields

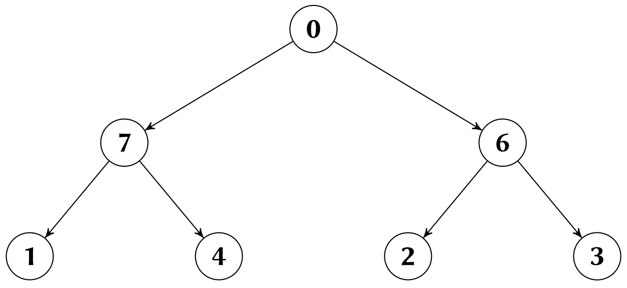


which corresponds to the array $[8, 7, 6, 1, 4, 2, 3, 0]$.

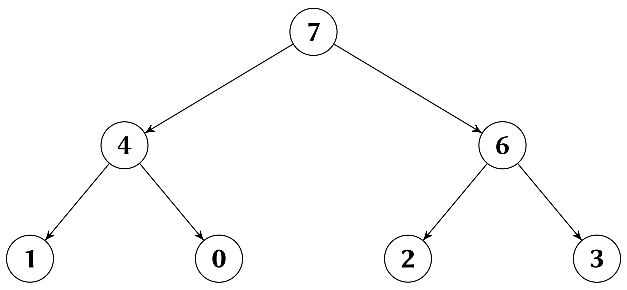
(c) Sort the above array in ascending order with heapsort, beginning with the heap that you obtained in (b). Draw the array after each intermediate step in which a key is moved to its final position.

Solution:

We begin with the max heap $[8, 7, 6, 1, 4, 2, 3, 0]$. We extract the root 8 and put it into the last position in the array, i.e., we swap 8 with the last element 0, removing 8 from the heap, which yields



We then sift 0 downwards until the heap condition is restored:



Now, the array is $[7, 4, 6, 10, 2, 3, 8]$ and contains the one-smaller heap in the front and the sorted entries in the end.

The array after the subsequent steps are as follows. Blue numbers are at their final positions.

1) Swap 7 and 3: $[3, 4, 6, 1, 0, 2, 7, 8]$
   Sift 3 down: $[6, 4, 3, 1, 0, 2, 7, 8]$
2) Swap 6 and 2: $[2, 4, 3, 1, 0, 6, 7, 8]$
   Sift 2 down: $[4, 2, 3, 1, 0, 6, 7, 8]$
3) Swap 4 and 0: $[0, 2, 3, 1, 4, 6, 7, 8]$
   Sift 0 down: $[3, 2, 0, 1, 4, 6, 7, 8]$
4) Swap 3 and 1: $[1, 2, 0, 3, 4, 6, 7, 8]$
   Sift 1 down: $[2, 1, 0, 3, 4, 6, 7, 8]$
5) Swap 2 and 0: $[0, 1, 2, 3, 4, 6, 7, 8]$
   Sift 0 down: $[1, 0, 2, 3, 4, 6, 7, 8]$
6) Swap 0 and 1: $[0, 1, 2, 3, 4, 6, 7, 8]$
   done: $[0, 1, 2, 3, 4, 6, 7, 8]$.

We are done.

Exercise 5.2 *Sorting algorithms.*

Below you see four sequences of snapshots, each obtained in consecutive steps of the execution of one of the following algorithms: `InsertionSort`, `SelectionSort`, `QuickSort`, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

3	6	5	1	2	4	8	7
3	6	5	1	2	4	8	7
3	5	6	1	2	4	8	7

3	6	5	1	2	4	8	7
3	5	1	2	4	6	7	8
3	1	2	4	5	6	7	8

3	6	5	1	2	4	8	7
3	6	1	5	2	4	7	8
1	3	5	6	2	4	7	8

3	6	5	1	2	4	8	7
1	6	5	3	2	4	8	7
1	2	5	3	6	4	8	7

Solution:

InsertionSort – BubbleSort – MergeSort – SelectionSort.

Exercise 5.3 *Counting function calls in recursive functions* **(1 point)**.

For each of the following functions g, h, and k, provide an asymptotic bound in big-O notation on the number of calls to f as a function of n. You can assume that n is a power of two.

Algorithm 1

(a)
```
function $g(n)$
    $i \leftarrow 1$
    while $i < n$ do
        $f()$
        $i \leftarrow i + 2$
    $g(n/2)$
    $g(n/2)$
    $g(n/2)$
```

Solution:

Denoting by $G(n)$ the number of calls to f performed by $g(n)$, we have

$$G(n) = 3G(n/2) + \lfloor n/2 \rfloor \leq 3 \cdot G(n/2) + \frac{1}{2} \cdot n^1.$$

Since $\log_2 3 > 1$, the Master theorem yields $G(n) \leq O(n^{\log_2 3}) = O(n^{1.58\ldots})$.

Algorithm 2

(b)
```
function $h(n)$
    $i \leftarrow 1$
    while $i < n$ do
        $f()$
        $i \leftarrow i + 1$
    $k(n)$
    $k(n)$
function $k(n)$
    $i \leftarrow 2$
    while $i < n$ do
        $f()$
        $i \leftarrow i^2$
    $h(n/2)$
```

Solution:

First, consider the number of calls to f performed in a call of $k(n)$. Variable i takes the values $2, 2^2, 2^4, 2^8 \ldots$, i.e., $(2^{2^j})_{1 \leq j}$. We leave the while loop when $2^{2^j} \geq n$, i.e., when $2^j \geq \log_2 n$ or $j \geq \log_2 \log_2 n$. Hence, the number of iterations is $\lceil \log_2 \log_2(n) \rceil$.

Denoting by $H(n)$ and $K(n)$ respectively the number of calls to f performed by $h(n)$ and $k(n)$, we have

$$\begin{aligned} H(n) &= 2K(n) + n - 1 \\ K(n) &= H(n/2) + \lceil \log_2 \log_2(n-1) \rceil + 1 \end{aligned}$$

Injecting the definition of $K(n)$ into the definition of H, we get

$$H(n) \leq 2H(n/2) + 2\log_2\log_2 n + n \leq 2 \cdot H(n/2) + 3 \cdot n^1$$

and since $\log_2 2 = 1$, the Master theorem yields $H(n) \leq O(n^{\log_2 2} \cdot \log n) = O(n \log n)$. Since $K(n) \leq H(n/2) + O(\log\log n)$, we immediately obtain $K(n) \leq O(n \log n) + O(\log\log n) = O(n \log n)$ too.

Exercise 5.4 *Bubble sort invariant.*

Consider the pseudocode of the bubble sort algorithm on an integer array $A[1, \dots, n]$:

Algorithm 3 BubbleSort(A)

```
for $1 \leq i \leq n$ do
    for $1 \leq j \leq n - i$ do
        if $A[j] > A[j+1]$ then
            $t \leftarrow A[j]$
            $A[j] \leftarrow A[j+1]$
            $A[j+1] \leftarrow t$
return $A$
```

(a) Formulate an invariant INV(i) that holds at the end of the i-th iteration of the outer for-loop.

Solution:

After i iterations of the outer for-loop, the subarray $A[n-i+1, \dots, n]$ is sorted and each element from $A[1, \dots, n-i]$ is not greater than each element from $A[n-i+1, \dots, n]$.

(b) Using the invariant from part (a), prove the correctness of the algorithm. Specifically, prove the following three assertions:

(1) INV(1) holds.

(2) If INV(i) holds, then INV($i+1$) holds (for all $1 \leq i < n$).

(3) INV(n) implies that BubbleSort(A) correctly sorts the array A.

Solution:

(1) INV(1) means that after the first iteration of the outer for-loop, the largest element of A is at position n. Suppose that this largest element was originally at position j for some $1 \leq j \leq n$. If $j = n$, the element will never be swapped by the first inner for-loop, and hence is still at position n at the end, as desired. For $j < n$, this largest element will be swapped to position $j+1$ in the j-th iteration of the inner for-loop, and then swapped to position $j+2$ in the next iteration, and so on until it is swapped to position n. So in both cases it is at position n at the end of the first for-loop.

(2) Let $1 \leq i < n$. Assuming that INV(i) holds, we know that before the $(i+1)$st iteration of the outer for-loop, the i last entries of the array are the i largest entries of the input array A sorted in ascending order. Using a similar reasoning as in (i), we see that during the $(i+1)$st iteration, the largest element among the remaining part of the array (namely $A[1, \dots, n-i]$) will be placed at the last position of this remaining part, so that now the the $i+1$ last entries of the array are the $i+1$ largest entries of the input array in ascending order. Therefore, INV($i+1$) holds.

(3) INV(n) means that the “subarray” $A[1,\dots,n]$ is sorted. But this is actually the full array (since A has length n) returned by BubbleSort(A), which shows that the algorithm correctly sorts the array A.

Exercise 5.5 *Guessing a pair of numbers* **(1 point)**.

Alice and Bob play the following game:

- Alice selects two integers $1 \leq a, b \leq 1000$, which she keeps secret
- Then, Alice and Bob repeat the following:
  - Bob chooses two integers (a', b')
  - If $a = a'$ and $b = b'$, Bob wins
  - If $a > a'$ and $b > b'$, Alice tells Bob ‘high!’
  - If $a < a'$ and $b < b'$, Alice tells Bob ‘low!’
  - Otherwise, Alice does not give any clue to Bob

Bob claims that he has a strategy to win this game in 12 attempts at most.

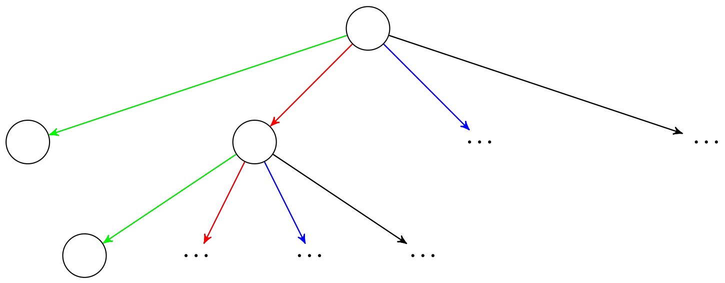
Prove that such a strategy cannot exist.

Hint: *Represent Bob’s strategy as a decision tree. Each edge of the decision tree corresponds to one of Alice’s answers, while each leaf corresponds to a win for Bob.*

Hint: *After defining the decision tree, you can consider the sequence $k_0 = 1$, $k_{n+1} = 3k_n + 1$, and prove that $k_n = \frac{3^{n+1}-1}{2}$. The number of leaves in the decision tree of level n should be related k_n.*

Solution:

Bob’s strategy can be represented as follows, where green arrows correspond to a win, red arrows to ‘high!’, blue arrows to ’low!’, and black arrows to the absence of a clue.



Each node of the corresponding tree has four children, of which one (corresponding to Bob winning the game) has no other child, while the three others can have four children with the same structure as their parent. Denoting by k_n the number of leaves in a tree of level $n + 1$ of the above form, we see that

$$\begin{cases} k_0 = 1 \\ k_{n+1} = 3k_n + 1 \quad \forall n > 0. \end{cases}$$

We will now prove by induction that, for all $n > 0$, we have: $P(n)$: $k_n = \frac{3^{n+1}-1}{2}$.

Base case: $n = 0 \quad k_0 = 1 = \frac{3^{0+1}-1}{2}$, hence $P(0)$.

Inductive step: Let $n > 0$. Assume $P(n)$. Then we have $k_{n+1} = 3k_n + 1 \overset{P(n)}{=} 3 \cdot \frac{3^{n+1}-1}{2} + 1 = \frac{3^{n+2}-3}{2} + 1 = \frac{3^{n+2}-3+2}{2} = \frac{3^{n+2}-1}{2}$, hence $P(n+1)$.

In order for Bob's strategy to allow him to win for any pair of integers chosen by Alice, the tree representing his strategy must have at least $1000 \cdot 1000 = 10^6$ leaves, which is the number of pairs (a, b) that Alice can choose. If Bob's statement is true, we therefore have $k_{11} \geq 10^6$. Now, $k_{11} = \frac{3^{12}-1}{2} < 10^6$, hence Bob cannot win in at most 12 attempts.