

Fundamental Algorithms

Note: exercises 1 and 4 are slightly different to those given on the German version of the worksheet.

Exercise 1

Compute the number of comparisons that will be performed by MERGESORT in the best case.

Solution:

Note: we only count comparisons between array elements!

In MERGESORT, comparisons are only performed during the MERGE-step. In the best case, the first element of one partition will be compared to (and found to be larger than) all $\frac{n}{2}$ elements of the other partition. After copying these $\frac{n}{2}$ elements of the other partition into the array, the elements of the first partitions will be copied to the array without performing any comparison. Thus, in the best case, MERGE will require $\frac{n}{2}$ comparisons.

If we assume $n = 2^k$, we get the following recurrence for the number $C(n)$ of comparisons in the best case:

$$C(n) = C\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right) + \frac{n}{2} = 2C\left(\frac{n}{2}\right) + \frac{n}{2}$$

Of course, $C(1) = 0$.

We try to solve this recurrence by substitution, and guess

$$C(n) := an \log_2 n + b$$

as the solution.

For $n = 1$, we get:

$$C(1) = a \log_2 1 + b = b = 0 \quad \Leftrightarrow \quad b = 0$$

And for $n > 1$:

$$\begin{aligned}
 an \log_2 n = C(n) &= 2C\left(\frac{n}{2}\right) + \frac{n}{2} \\
 \Leftrightarrow an \log_2 n &= 2\left(a\frac{n}{2} \log_2 \frac{n}{2}\right) + \frac{n}{2} \\
 \Leftrightarrow an \log_2 n &= 2\left(a\frac{n}{2}(\log_2 n - 1)\right) + \frac{n}{2} \\
 \Leftrightarrow an \log_2 n &= an \log_2 n - an + \frac{n}{2} \\
 \Leftrightarrow 0 &= -an + \frac{n}{2} \\
 \Leftrightarrow a &= \frac{1}{2}
 \end{aligned}$$

Hence, in the best case, MERGESORT will require $\frac{n}{2} \log_2 n$ comparisons between array elements (for $n = 2^k, k \in \mathbb{N}$).

Exercise 2

State the minimal and maximal number of elements that can be stored in a heap of depth d .

Solution:

The number of elements per level of a heap doubles from each level to the next (except the lowest level which may contain less elements). Hence, the number of elements in a heap of depth d is given by the sum

$$\sum_{k=0}^{d-1} 2^k + n_d = \frac{1 - 2^d}{1 - 2} + n_d = 2^d - 1 + n_d,$$

where $1 \leq n_d \leq 2^d$ is the number of elements on the lowest level. The number of elements in a heap of depth d is therefore at least 2^d , and at most $2^{d+1} - 1$.

Exercise 3

Prove: for each subtree of a heap, the maximal number of this subtree will be stored in its root.

Hint: use the heap property for your proof.

Solution:

We prove this statement by induction over the height h of the heap.

case $h = 1$: A heap of height $h = 1$ only has one single node, which is also the maximal node.

step $h \rightarrow h + 1$: Our induction assumption is that, in each tree of height $\leq h$, the maximal element will be stored in the root. A heap of height $h + 1$ consist of the root and two subtrees of height h . Due to the induction assumption, the maximal elements of the subtrees will be stored in their roots. The heap property requires that these two roots are also smaller than their parent element, which is the root of the entire tree. Therefore, the root of the entire tree is larger than all elements in the subtrees, and, hence, is the maximal element in the entire heap.

Exercise 4

Prove the correctness of the algorithm BUILDHEAP:

```
BUILDHEAP( A: Array[1..n]) {  
    heapsize := n;  
    for i from n downto 1 do {  
        HEAPIFY(A,n,i)  
    }  
}
```

Solution:

We assume that HEAPIFY is correct in the sense that HEAPIFY(A,n,i) will restore the heap property of the heap starting from element $A[i]$ provided that the heap property is satisfied by all heaps starting from element $A[j], j > i$.

To prove the correctness of BUILDHEAP, we use the following **loop invariant**:

Before each iteration of the for-loop, the heap property will be satisfied in all heaps starting from an element $A[j]$, where $j > i$.

Due to the correctness of HEAPIFY, the heap property will, after the iterating the loop for i , also be satisfied for the heap starting from element $A[i]$. Hence, the loop invariant is also satisfied for the next iteration of the loop.

At the end of the loop, the loop invariant states, that the heap property is satisfied for all heaps starting from any element $A[j], j > 0$. Therefore, the heap property is satisfied for the entire heap, and the algorithm BUILDHEAP is correct.