1 Correctness of Heapsort

We assume the correctness of the procedure *BuildHeap* for this problem. We will prove the correctness of *BuildHeap* in the next section. The textbook already proves the correctness of Heapify.

The following discussion is very detailed. Normally you probably do not have to be so detailed.

The procedure *Heapsort* is:

begin
BUILDHEAP;
for $i = n$ step -1 until 2 do {
  swap $A[1]$ and $A[i]$;
  HEAPIFY(1,$i-1$)
  }

Transforming into a while loop:

begin
BUILDHEAP;
$i = n+1$;
while $(i \geq 3)$ {
    $i = i - 1$;
    swap $A[1]$ and $A[i]$;
    HEAPIFY$(1, i-1)$
}

The invariant I is defined as:

$\forall i \forall j \forall k : [2 \leq i \leq n+1] \wedge [((j < k) \wedge (i \geq k)) \Rightarrow (A[j] \leq A[k])] \wedge [A[1..i]$ is a heap]

Proof:

- Initially $i = n+1$ and the invariant I is true.
- If C is $(i \geq 3)$ and B is the body of the while loop, we will show that $\{I \wedge C\}B\{I\}$. Before the start of B assume that $3 \leq i \leq n+1$ and that I is true. This implies that $A[i \ldots n]$ consists of the largest numbers in a sorted order and that $A[1 \ldots i-1]$ is a heap. When the first statement swaps $A[1]$ and $A[i-1]$, the largest value in the heap $A[1 \ldots i-1]$ now occupies $A[i-1]$ so that $A[i-1 \ldots n]$ are now the largest elements. Since the invariant was true for $i = k$, $A[i-1]$ was less than every element in $A[i \ldots n]$ so that $A[i-1 \ldots n]$ is now sorted. So I is still true.

- When I is true and C is false, then $i = 2$ and we exit the loop. This implies that we have $A[1..n]$ sorted.

The termination function can be defined as $i-3$ where i is the loop variable in the program.

2 Correctness of BuildHeap

In the form of a while loop,

$i = n + 1$;
while $(i \geq 2)$ {
    $i = i - 1$;
    Heapify(i, n);
}

The loop invariant I is defined as
$(n + 1 \geq i \geq 2) \wedge A[i \ldots n]$ is a forest of heaps.
Proof:
The proof goes very similarly as the proof for Heapsort. Very concisely,

- Initially I is true
- The body of the loop joins two heaps into a heap in the forest.
- eventually i=1 and we're done.

The termination function is $(i - 2)$ where i is the loop variable.

3 Problem 3.18

```
procedure Heapify(i,j)
if j is not the root and the father of j contains a larger element than j then
        begin
                interchange A[j] with A[[j/2]]
                Heapify(i,[j/2])
        end

procedure BUILDHEAP
for i=1 to n in steps of 1
    Heapify(1,i);
end
```

BUILDHEAP now takes $O(nlogn)$.