CS4311 Design and Analysis of Algorithms

Homework 2 (Solution Sketch)

1. In proving the inductive case, John has assumed that $1+2+\cdots+k = O(n)$, and his target is to show $1+2+\cdots+k+(k+1) = O(n)$. By definition, this means that John has assumed $1+2+\cdots+k \leq cn$ for some chosen c, and his target is to show $1+2+\cdots+k+(k+1) \leq cn$ for *the same* c.

   Unfortunately, when John wrote:

   $$1 + 2 + \cdots + k + (k + 1) = O(n) + (k + 1) = O(n) + O(n) = O(2n) = O(n),$$

   this cannot achieve his target, because when we apply the assumption, we only get:

   $$1 + 2 + \cdots + k + (k + 1) \leq cn + (k + 1),$$

   which is not at most cn. In other words, the inductive case cannot be shown, so that John's proof is wrong.

2. One way is to show the following (very long) statement by induction:

   After each swap, we simultaneously have

   (a) Only node x may violate the heap property (with $parent(x)$ having a larger value);

   (b) The depth of x is decreased by 1;

   (c) The value in the parent of x is smaller than the values in the children of x (if exists);

   By (a), we can see that whenever $parent(x)$ has a smaller value, we can stop, knowing that all nodes satisfy the heap property. By (b), we can see that the swapping will eventually stop. By (c), we can guarantee that after each swap, only node x may violate the heap property (why?).

3. There are a couple of algorithms for this problem.

   Method 1: By merging: We partition the array into blocks of contiguous d entries. Then we sort each block independently. Afterwards, we merge the ith block with the $(i+1)$th block, for $i = 1, 2, \ldots, n/d - 1$.

   Method 2: By heap: We insert the first d entries. Then, we alternately call `Extract-Min` to output the smallest item, and `Insert` to insert the next entry from the array.

   Method 3: By heap: We observe that the 1st, $(2d+1)$th, $(4d+1)$th, $(6d+1)$th, and so on entries are increasing. Similarly, the jth, $(2d+j)$th, $(4d+j)$th, and so on entries are also increasing. We thus can naturally partition the input array into $\Theta(d)$ increasing arrays. Then, we can marge these $\Theta(d)$ arrays by using a heap (how?).

   Method 4: By sorting: We partition the array into blocks of contiguous d entries. Then at the ith step, we sort the ith and the $(i+1)$th blocks together, for $i = 1, 2, \ldots, n/d - 1$.