* 1. A binary heap with minimum key in the root contains exactly n^3 elements, where n is positive integer. We have to remove $n^2 \cdot \lg(n)$ smallest elements from this heap. What is asymptotic complexity of this process? Will the complexity be different if we consider a binomial heap instead?

A. Removing an element from a binary heap containing N elements takes $O(\log(N))$ time, in this case $N = n^3$, the time is therefore $O(\log(n^3)) = O(\log(n))$. Repeating the process $n^2 \cdot \lg(n)$ times results in complexity
$O(n^2 \cdot \lg(n) \cdot \log(n)) = O(n^2 \cdot \log^2(n))$.
B. The same reasoning holds for for binomial heap.
C. Discussion: Can the process be sped up by some "global" approach like e.g. "cut away" some larger upper section of the heap tree", which would remove more elements in one (possibly fast) operation? All k smallest elements in the heap can be found in $O(k \cdot \log(k))$ time, using e.g. BFS which prioritizes the expansion of the search tree into the nodes with smallest keys. Or (equivalently in this case) using Prim algorithm started in the heap root and using the children's key as the weight of the edge parent-child. Both approaches need a separate queue to complete the task. So, the complexity of discovering $n^2 \cdot \lg(n)$ smallest keys would be
$O(n^2 \cdot \lg(n) \cdot \log(n^2 \cdot \lg(n)) = O(n^2 \cdot \lg(n) \cdot \log(n))$ which gives us no gain compared to the previous method.

* 2. The depth of a d-ary heap is h and it is also the depth of all its leaves. Therefore, the number of leaves is d^h. What is the minimum and maximum possible number of keys comparisons which are perfomed in this heap during operation DeleteMin?

Minimum: Let $c_1, ..., c_d$ be the immediate children of the root. Root is removed and substituted by value V from the bottom of the heap. Let $C_k = \min(c_1, ..., c_d)$. In slight abuse of notation, cj denotes both node in the heap and also the key stored in that node. Heap reapir swaps V with C_k. Now it is possible that all children of V (originally children of C_k) are bigger than V, and therefore the repair operation stops here. The number of comparisons in the whole process is $2d$. (Finding minimum among the children of an element takes $d-1$ comparisons and then the element is compared once against the child with the minimum value.)
Maximum:
The value of V is so big that in the process of the repair heap operation it eventually reaches the lowest level (a leaf) in the heap. In all levels, except for the last which is the leaves level, exactly d comparisons are made. There are $h+1$ levels (including the root in 0-th level) in the heap, the maximum number of key comparisons is thus dh.

* 3. What is the maximum possible degree (= number of children) of a key in a binomial heap with N elements?

It is the degree of the root of the biggest binomial tree in the heap. Denote this degree by D. It holds, $2^D \leq N < 2^{D+1}$.

* 4. In general, the degree of a node in a binomial heap is not bounded from above, it can be arbitrarily big (providing the heap itself is sufficiently big). The children of the node are the roots of other binomial subtrees and the node stores the references to those roots in some structure S. There are two possibilities:
1. The references in S are stored in sorted order according to the order of the trees they refer to.
2. The references in S are stored in random order.
Consider how the speed of operations Insert and DeleteMin is affected by implementing either possibility1 or 2. Which one would you recommend and why?

It is better to keep the roots of a node subtrees (= node children) sorted by their order (or the degree, which is the same as the order). During the merge operation, the trees of the same order in two heaps have to be merged. When the merge operation is a part of ExtractMin operation the children of the node with the minimum value has to be merged with other binomial trees in the original heap. There are O(log n) children of the node with the minimum value. If these children are stored in random order the search for the node with the smallest degree would take O(log n) time. The search for the node with the next smallest degree would also take O(log n) time etc., in total it would take $O(\log^2(n))$ time (or $O(\log(n) \cdot \log(\log(n)))$ time if the nodes are sorted just before merging) . Sorted children guarantee one pass through all roots in both merged heaps which results in O(log n) time of the merge operation.

* 5. A Fibonacci heap is originally empty. Next, we insert $2^n + 5$ different keys ($n > 2$). Then we perform operation DeleteMin and consolidate the heap. How many binomial trees which have their roots in the root list of the heap will be there in the heap after the consolidation?

After inserting $2^n + 5$ different keys into the fibonacci heap, the heap will contain $2^n + 5$ trees, each of them being a single (root) node. Operation DeleteMin reduces the number of trees (and also elements, in this case) to $2^n + 4$

and runs the consolidate operation. The result of the consolidate operation will be only binomial trees, as there are no "reduced" trees in the heap from which some nodes were removed previously. There will be only two binomial trees, one with 4 elements and onothe one with 2^n elements.

6. Choose some particular data structure(s) which is, in your opinion, suitable for representing a binomial heap in computer memory. Show that your implementation preserves asymptotic complexity of operation Insert, AccessMin and DeleteMin. You can use more types of structures if you want to.

In Merge, ExtractMin, Insert operation, whole subtrees change their location in the heap. That is, O(n) nodes change their location in the heap in a single operation. Therefore, it seems that an implementation should prefer some kind of structure based mainly on pointer manipulation, not manipulation of the nodes themselves, as it usual in e.g. binary heap. A node of order k might be equipped with a list of k references(pointers) to its children. Accessing the children of node with degree d then takes O(d) time. If the heap supports DecreaseKey operation each node has to be equipped with a reference to its parent. Also, each root node should be equipped with the reference to the next node in the list of binomial trees in the heap. As each node in the heap may become eventually a root of some binomial tree on the highest level in the heap, it may be a good idea to equip each node with the same reference despite the fact that it is has to be set set to null when the node is not a tree root.
The list of children of a node has to be extendible in O(1) time, because a node may acquire a new child during Merge operation. A Min reference should be impleemnted pointing to the node with the minimum key in the heap. In Insert operation it can be updated in O(1) time, in Extract min in can be updated in O(log(n)) time.
The list of all binomial tree roots may be bi-directional (as in Fibonacci heap), it speeds up ExtractMin operation, a node can be removed from such list in O(1) time.

7. There are n ($n \geq 2$) different integer keys and an empty binomial heap. We insert all keys into the heap in random order. What is the asymptotic complexity of this process? Is it possible that for some particular ordering of keys the complexity of inserting them all into the heap would be smaller?

Amortized complexity of a sequence of Insert operations in a binomial heap is O(1), therefore the process will take O(n) time. The complexity does not depend on the values of the inserted elements.

8. A binomial heap keeps minimum values in the roots of its trees. The task is to find a maximum value in this heap and delete it from the heap. How long is the time in which the task is finished?

Any maximum value can be stored only in a leaf of some binomial tree. There are O(n) leaves in a binomial heap with n elements. As there is no additional structure among the leaves, all leaves has to be searched to find the maximum. The asymptotic complexity of the search is thus O(n).

9. Let us suppose that binomial heap H is not completely filled, i.e. some of its binomial trees are empty. The number of unempty trees is k. How many leaves are there in the whole heap? Suppose H contains exactly n keys. What is the relation between n and the maximum possible value of k?

A.
The number of leaves in a binomial tree of order $p > 0$ is exactly half of the number of all its nodes.
The proof by induction is trivial, there is one leaf in the binomial tree of order one and each binomial tree of order $p > 1$ is a composition of two binomial trees of order $p-1$.
So, the number of the leaves in a binomial heap does not depend on a presence of particular binomial trees of order $p > 0$. If a binomial heap contains the binomial tree of order 0 then the number of the leaves in the heap is equal to $(n+1)/2$, otherwise it is equal to $n/2$.
B.
The number of binomial trees in a binomial heap with n elements is equal to the number of 1's in the binary expression of n. It holds, $2^k - 1 \leq n$, as the number of 1's can be at most $\lceil \log_2(n) \rceil$.

10. Binary heap is stored in 1D array of keys. There is a simple formula describing how to calculate the indices of the left and right children of a given key when we know the index of the key. Write down this formula. Write a similar formula for a ternary heap and for a general d-ary heap.

A. binary.
iC = index of **C**urrent node; similarly, iL, iM = indices of Left and Right child, respectively.

0-based indices: iL = 2*iC+1, iR = 2*iC+2
1-based indices: iL = 2*iC, iR = 2*iC+1

B. ternary. (**L**eft-, **M**iddle-, Right-, child)
0-based indices: iL = 3*iC+1, iM = 2*iC+2, iR = 2*iC+3
1-based indices: iL = 3*iC−1, iM = 3*iC, iR = 3*iC+1

C. General *d*-ary tree. (iCh_j - index of *j*-th child, presuming there is 0-th, 1-st, ..., (*d*−1)-th child of a node)
0-based indices: $iCh_j = d*iC+j+1$
1-based indices: $iCh_j = d*(iC-1)+j+2$