1. (a) Design and analyse an algorithm for building a $k$-ary heap in the following two different ways.

- One starts at the last internal node and works backwards along the array till the root running heapify at each intermediate node, assuming that the subtrees rooted at its children are already heaps.
- The other grows the size of the heap one at a time, initially assuming only element one is present, at the first array position. When it increases to include one extra element, it checks if it is correctly placed with respect to its parent. At this stage it is assumed that the array of elements preceding it constitute a heap. The algorithm then moves the new node up the tree using exchanges until a correct position has been found.

Obtain precise expressions for the parent and children of a node in a $k$-ary heap indicating how you computed it. Assume that the root is at $A[1]$. Also derive exact asymptotic bounds on the running times of the two different BUILD-HEAP procedures, in terms of $n$ and $k$.

(b) Design a routine to restore the heap property for key-increase/key-decrease. That is, you have a heap, and one of the keys (given by an index) has its value changed (increased/decreased). Analyse the running time precisely.

(c) If a node is to be deleted, from an existing heap (the node index is given as input) write a routine to efficiently restore the heap property among the remaining nodes. Analyse the running time.
(d) Write a routine for inserting a new node into an existing heap. Analyse the complexity assuming the original heap has \( n \) elements.

2. Modify randomised quicksort to pick a set of three elements uniformly at random and then partition around the median element. We assume that the number \( n \) of elements is at least 3, and that the array contains all distinct elements.

(a) Compute the probability of picking an element of rank \( i \) as pivot by this method. Derive an exact value.

(b) Calculate the probability of a split at least as balanced as \( \alpha, 1 - \alpha \).

(c) Prove that for the normal randomised quicksort, the expected running time is \( \Omega(n \log n) \). Note, in the lecture, we only saw an \( O(n \log n) \) expected running time. This asks a proof of the lower bound.