University of Bristol
COMS21103: Data Structures and Algorithms
Problem Set 7

Problem 1: A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. Use aggregate analysis to determine the amortised per operation.

Solution: Let c_i denote the cost of the ith operation. Summing the cost of the n operations gives us that

$$\sum_{i=0}^{n} c_i \leq n + \sum_{i=0}^{\lfloor \lg n \rfloor} 2^j = n + 2^{\lfloor \lg n \rfloor + 1} - 1 \leq n + 2n - 1 < 3n.$$

Therefore the amortised cost of each operation is less than $3n/n = O(1)$. □

Problem 2: A sequence of stack operations is performed on a stack whose size never exceeds k. After every k operations, a copy of the entire stack is made for backup purposes. Show that the cost of n stack operations, including copying the stack, is $O(n)$ by assigning suitable amortised costs to the various stack operations.

Solution: Notice that the stack size is at most k. We only need to assign an extra credit to the each stack operation. Then we get extra k credits after every k operations, and these k credits can be used to pay for copying. □

Problem 3: Execute the following operations on an initially empty Fibonacci heap:

Insert(18), Insert(14), Insert(17), Insert(28), Insert(32), Insert(37),
Insert(25), Insert(36), Insert(53), Insert(40), ExtractMin(), DecreaseKey(40, 30),
Delete(36), ExtractMin().

For all intermediate steps, illustrate the resulting Fibonacci heap. New elements should always be inserted to the right of the current minimum. The consolidation operation after ExtractMin starts with the next element on the right hand side of the deleted minimum.

Solution: See Page 3 and Page 4.

Problem 4: Professor Pinocchio claims that the height of an n-node Fibonacci heap is $O(\lg n)$. Show that the professor is mistaken by exhibiting, for any positive integer n, a sequence of Fibonacci-heap operations that creates a Fibonacci heap consisting of just one tree that is a linear chain of n nodes.

Solution: We first obtain a single tree with two nodes: Such tree can be constructed easily by adding three nodes first and applying an ExtractMin operation once. With this we have a chain of 2 nodes.

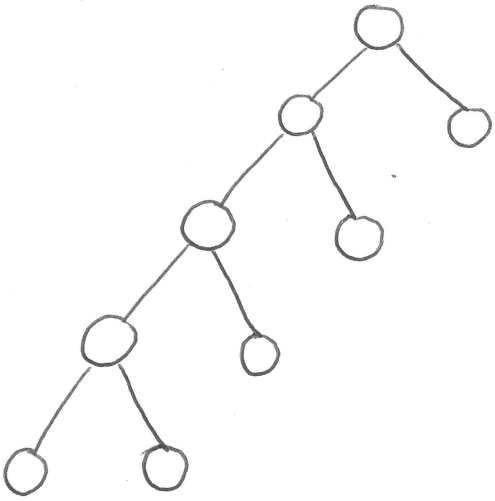
Assume that we construct a chain of length ℓ, and we can extend the length of the chain by 1 through the following operations: Assume that k is the key of the root of the current tree (chain), and a, b, c be three new keys such that $a < b < c < k$. We perform the following operations:

Insert(a); Insert(b); Insert(c); ExtractMin; DecreaseKey(c, a); ExtractMin.

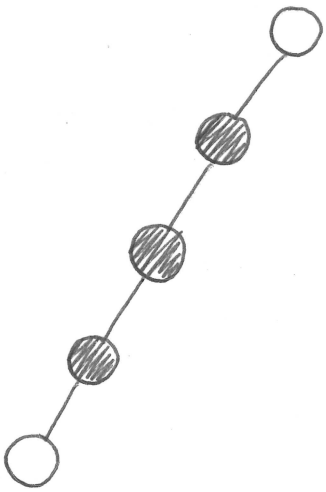
It is easy to see that these operations together extend the length of the chain by 1. Performing this set of operations $n - 1$ times leads to a chain of n nodes. □

Problem 5: What is the *actual* (not amortised) worst-case running time of a single DecreaseKey operation on a Fibonacci heap with n elements (in O-notation)? Justify your answer.

Solution: For any Fibonacci heap with n elements, the actual worst-case runtime of a single DecreaseKey is $O(n)$. We first use the ideas similar with Problem 4 to construct a tree like this:



After that, we delete every node's right child so that every node, except the root and the leaf, is marked. Notice that at the moment we obtain a chain like this:



Finally, we apply a DecreaseKey operation on the leaf so that the heap-order is violated. In this case, the cascading-cut operation needs to be recursively invoked $O(n)$ times. Hence, the worst-case runtime of a single DecreaseKey operation is $O(n)$. □

Problem 6: Suppose that we generalise the cascading-cut rule to cut a node x from its parents as soon as it loses its kth child, for some integer k. Prove that, as long as $k = O(1)$, specific choices of k do not change the asymptotic amortised costs of the operations discussed in class.

Solution: Assume that we allow a node to lose k children before we move it to the root list. It is easy to see that Insert operation is not affected. Now we analyse the amortised costs of ExtractMin and DecreaseKey.

We define the potential function Φ be twice the number of marked nodes plus the number of trees in the root list, where a node is marked if it has lost $k-1$ children. In particular, a node is unmarked if it is in the root list. With an analysis very similar to the one for usual Fibonacci heaps, we obtain that the minimum size u_d of a tree with the root node of degree d satisfies

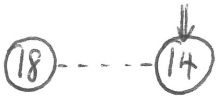
$$u_d \geq k + 1 + u_1 + \cdots + u_{d-k+1},$$

and so u_d is bounded below by a Fibonacci number F_{d-k}. Hence, as long as $k = O(1)$, u_d is exponential in d and the proof of the usual Fibonacci heaps carries over. □

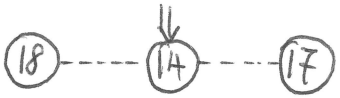
Solution of Problem 3

Insert(18)

Insert(14)



Insert(17)

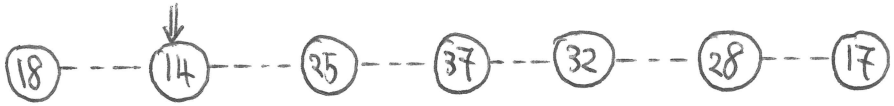


Insert(28)

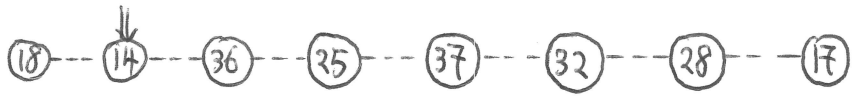
Insert(32)

Insert(37)

Insert(25)

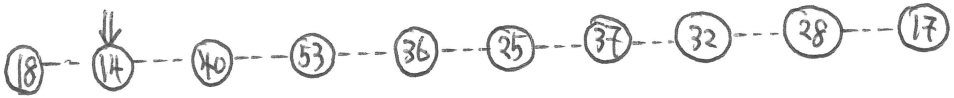


Insert(36)



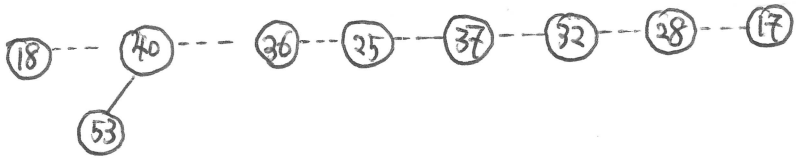
Insert(53)

Insert(40)

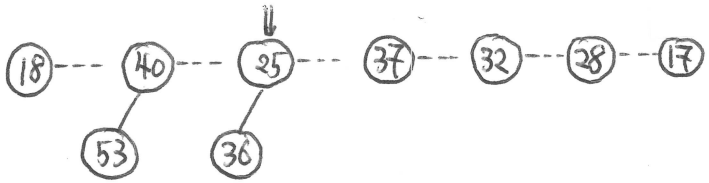


Extract Min

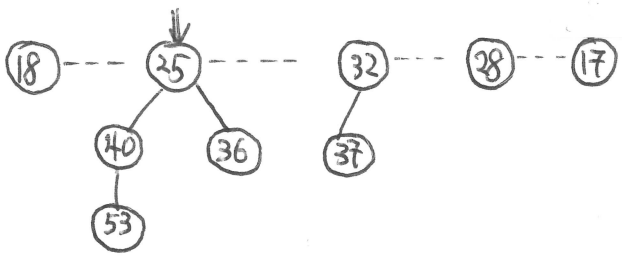
i)



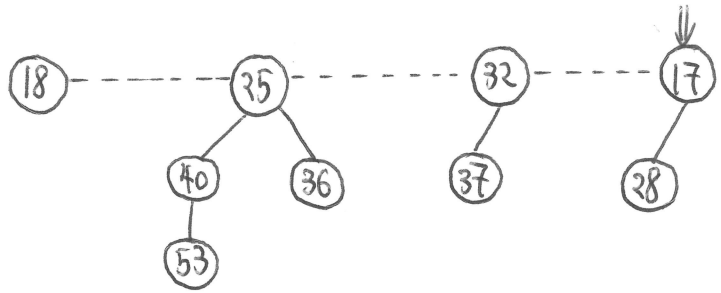
ii)



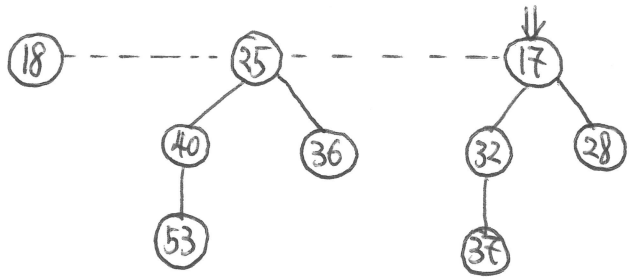
iii)



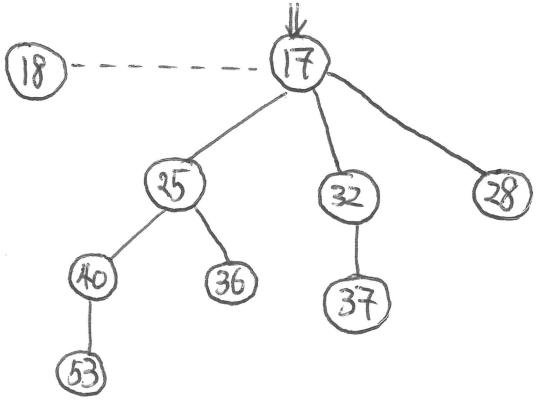
(iv)



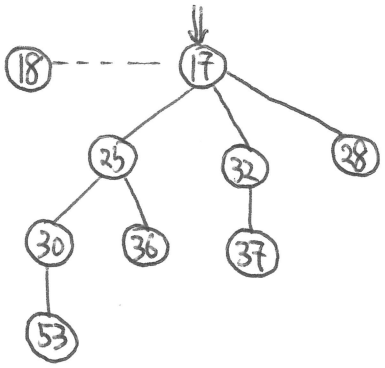
(v)



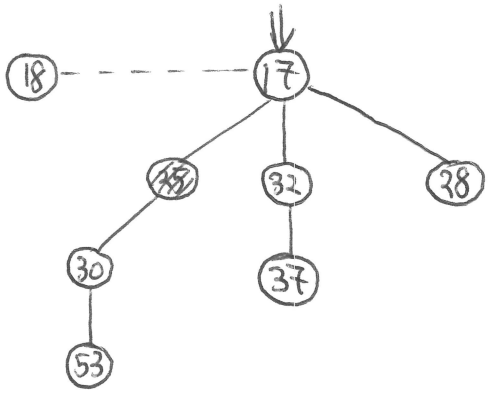
(vi)



DecreaseKey(40, 30)

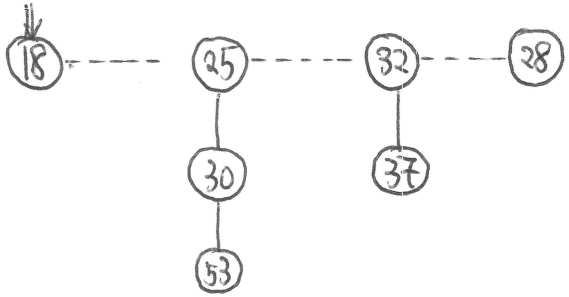


Delete(36)

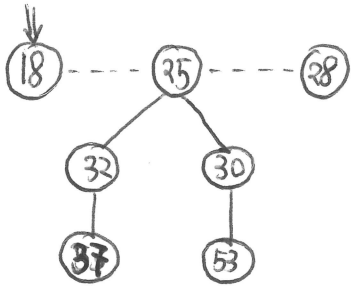


ExtractMin

i)



ii)



iii)