Prob. 1	Prob. 2	Prob. 3	Prob. 4

Problem 1.

This is a paragraph

Problem 2.

Problem 3.

Let's consider this problem in a high and theoritical point of view. We know that we have a priority queue that supports insert and delete-min in $O(log(n))$ worst-case execution time. This means that :

- $\exists\alpha$ such that the real execution time of insert is always less than $\alpha log(n)$
- $\exists\beta$ such that the real execution time of delete-min is always less than $\beta log(n)$

The main idea is that if we begin with an initially empty tree, there will always be more insert than delete-min. Therefore we can design a potential function that makes the amortized cost of insert a little bit higher (but in the same asymptotic class) and makes the amortized cost of delete-min constant.

Let's call n the total number of nodes in the tree. We are now able to define our potential function $\Phi(n)$: for $n > 0, \Phi(n) = \beta nlog(n)$; $\Phi(0) = 0$ (by continuity)

Let's compute the sum of the real cost and the potential variation of the two operations insert and delete-min. First for delete-min, the tree has initially n nodes and we insert a new one.

$realcost + \Delta\Phi \leqslant \alpha log(n) + \beta((n+1)log(n+1) - nlog(n))$

Let's compute separately $\Delta\Phi/\beta$:

$\Delta\Phi/\beta = nlog(n+1) - nlog(n) + log(n+1)$

$= nlog(\frac{n+1}{n}) + log(n+1)$

$= nlog(1+1/n) + log(n+1)$

$= n[1/n + o(1/n)] + log(n+1)$

$= log(n+1) + 1 + o(1)$

$= O(log(n))$

Therefore, $realcost + \Delta\Phi \leqslant O(log(n))$, the amortized cost of insert is less than $log(n)$. Let's do the same for the delete-min operation :

$realcost + \Delta\Phi \leqslant \beta log(n) + \beta(nlog(n) - (n+1)log(n+1))$

$realcost + \Delta\Phi \leqslant \beta log(n) + \beta(nlog(n) - (n+1)log(n))$

$realcost + \Delta\Phi \leqslant \beta log(n) - \beta(log(n))$

$realcost + \Delta\Phi \leqslant 0$

Thus, the amortized time of delete-min operation is $O(1)$.

Problem 4.

1.

After 15 pushes followed by 7 pops the stack will be in the following state :

S_0	8
S_1	7
S_2	6 5
S_3	4 3 2 1

2.

The worst case scenario for pop is for any given integer k when S_0 contains one element and for i from 1 to k S_i is half full. In that state a pop will empty S_0 which will then take elements from S_1 emptying it in the process. S_1 will then take elements from S_2 emptying it in the process. This so empties every S_i until the last one resulting in a number of operation equal to the number of elements in the stack. We can also realize that the resulting state is the worst case scenario for a push operation.

The worst case for push is for any given integer k when for i from 0 to k S_i is completely full. In that state pushing an element in the stack will overload S_0 which will then push its elements on S_1 overloading it in the process. S_1 will then push its elements on S_2 overloading it in the process. This so overload every S_i until the last one resulting in a number of operation equal to the number of elements in the stack. We can also realize that the resulting state is the worst case scenario for a pop operation.

The worst case scenario for pop and push are both in linear $O(n)$ time.

3.

Let k be any integer and $n = 2^k - 1$.

Initial state : Empty stack

Operation sequence :

1. n pushes
2. n times a push followed by a pop
3. n pops

The second part of the proposed sequence of operations consist of n worst case pushes and n worst case pops. That makes $O(n^2)$ basic operations. We can then conclude that the two operations can't be achieved in constant amortized time.