Practice problems: Amortized analysis

1. In this problem we consider a *monotone priority queue* with operations Init, Delete, and DeleteMin. Consider the following implementation using a boolean array A:

```
Init(n)
   for i=1 to n do
     A[i]=true
   end
end

Delete(i)
   A[i]=false
end

DeleteMin()
   i=1
   While A[i]=false do
       i=i+1
   end
   if i=<|A| then
       Delete(i)
       return i
   else
       return 0
   end
end
```

   (a) Analyze the running time of each of the procedures.

   (b) Describe a simple modification to DeleteMin such that it has amortized running time $O(1)$ (while maintaining the running times of Init and Delete). Explicitly give the potential function used in your analysis.

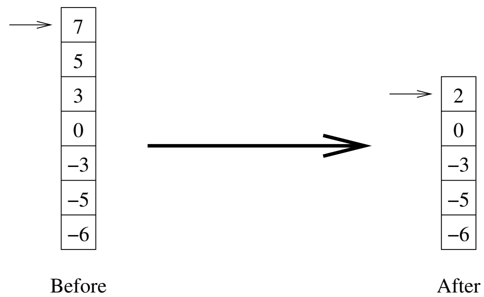
   (c) Describe a different implementation such that both Delete and DeleteMin have worst-case running time $O(1)$.

2. **(CPS130 final spring 2001)** An *ordered stack* $\mathcal{S}$ is a stack where the elements appear in increasing order. It supports the following operations:

   - INIT($\mathcal{S}$): Create an empty ordered stack.

- Pop($\mathcal{S}$): Delete and return the top element from the ordered stack.
- Push($\mathcal{S}$, x): Insert x at top of the ordered stack and reestablish the increasing order by repeatedly removing the element immediately below x until x is the largest element on the stack.
- Destroy($\mathcal{S}$): Delete all elements on the ordered stack.

Example: The following shows an example of an ordered stack and the same stack after performing a Push($\mathcal{S}$,2) operation (the order is reestablished by removing 7, 5, and 3)



Like a normal stack we implement an ordered stack as a double linked list (maintaining a pointer to the top element).

(a) What is the worst-case running time of each of the operations Init, Pop, Push, and Destroy?

(b) Argue that the amortized running time of all operations is $O(1)$.

3. We have previously seen that n increment operations on an initially ze ro k-bit counter can be performed in $O(n)$ time, that is, the amortized time for one increment operation is O(1). In this problem we will consider both incrementing and decrementing a binary counter.

(a) Describe a scenario where n increment/decrement operations performed on an initially zero k-bit counter take $O(nk)$ time.

In order to deal with decrement operations more efficiently we modify the counter representation such that each 'bit' can take the values 0,1 and -1 (instead of just 0 and 1). We store the counter in an array $A[0..k-1]$ ($A[i] \in \{-1, 0, 1\}$) and assume m to be the leftmost non-zero 'bit':

$$m = \max_{0 \le i < k} \{i | A[i] \neq 0\}.$$

We define $m = -1$ if all the 'bits' of the counter are zero. The value stored in the counter is

$$val(A, m) = \sum_{i=0}^{m} A[i] 2^i.$$

Note that $val(A, m) = 0$ iff $m = -1$.

(b) Give an example showing that the representation of a number other than 0 is not unique.

Consider the following procedures for incrementing and decrementing the counter. For simplicity we assume that the counter has infinite size $k = \infty$, that is we assume that we always have enough 'bits':

```
INCREMENT(A,m)
   if m =-1 then
          A[0] = 1
          m = 0
   else
          i = 0
          while A[i] = 1 do
                A[i] = 0
                i = i + 1
          A[i] = A[i] + 1
          if A[i] = 0 and m = i then
                m = -1
          else
                m = max{m,i}
   end

DECREMENT(A,m)
   if m=-1 then
          A[0] = -1
          m = 0
   else
          i = 0
          while A[i]  = -1 do
                A[i] = 0
                i = i + 1
          A[i] = A[i] - 1
          if A[i] = 0 and m = i then
                m = -1
          else
                m = max{m,i}
   end
```

(c) Assume that n increment and decrement operation are performed on an initially zero counter. Show that the amortized cost of an operation is $O(1)$.