Dalhousie University
CSCI 6057/4117 — Advanced Data Structures
Winter 2022
Assignment 1

Distributed Wednesday, January 12 2022.
Due at 23:59 Wednesday, January 26 2022.

Guidelines:

1. All assignments must be done individually. The solutions that you hand in must be your own work.
2. Submit a PDF file with your assignment solutions via Brightspace. We encourage you to typeset your solutions using LaTeX. However, you are free to use other software or submit scanned handwritten assignments as long as **they are legible**.
3. Working on the assignments of this course is a learning process and some questions are expected to be challenging. Start working on assignments early.
4. Whenever you are asked to prove something, give sufficient details in your proof. When it is straightforward to prove a lemma/property/observation, simply say so, but do not claim something to be straightforward when it is not.
5. We have the following late policy for course work: `http://web.cs.dal.ca/~mhe/csci6057/assignments.htm`

Questions:

1. [10 marks] A *sorted stack* is a stack in which elements are sorted in increasing order from bottom to top. It supports the following operations:

   `pop(S)`, which removes and returns the top element from the sorted stack S.

   `push(S, x)`, which first pushes item x onto the top of the sorted stack S and then maintains the increasing order by repeatedly removing the item immediately below x until x becomes the largest item in the stack.

   Note that here the `push` operation is different from that of a standard stack. For example, suppose that the items in the sorted stack S, from bottom to top, are currently $-9, -7, 0, 1, 4, 11, 12, 20$. Then, after calling `push(S, 5)`, the content of the stack, from bottom to top, becomes $-9, -7, 0, 1, 4, 5$.

   We implement the stack as a linked list.

   Show that the amortized costs of both operations are $O(1)$.

2. [10 marks] In the vector solution to the problem of maintaining resizable arrays shown in class, we allocate a new array with twice the size as the old one when inserting into a full array.

   In this question, we study the following alternative solution: Initially the array has a block of memory that can store at most c_0 entries, where c_0 is a positive constant. Each time we insert into a full array, we allocate a new memory block whose size (i.e., the maximum number of entries that it could store) is the size of the current memory block plus a fixed constant value c. We then copy all the elements from the current memory block to the newly allocated memory block, deallocate the old memory block and insert the new element into the new block. The new memory block will be used to store the content of the array afterwards, until we attempt to insert into a full array again.

   Prove that, under this scheme, performing a series of n insertions into an initially empty resizable array takes $\Omega(n^2)$ time, no matter what constant value we assign to c.

3. [15 marks] In class we saw the example of incrementing an initially 0 binary counter. Now, suppose that the counter is expressed as a base-3 number. That is, it has digits from $\{0, 1, 2\}$.

   (i) [5 marks] Write down the pseudocode for `INCREMENT` for this counter. The running time should be proportional to the number of digits that this algorithm changes.

   (ii) [5 marks] Define the actual cost of `INCREMENT` to be the exact number of digits changed during the execution of this algorithm. Let $C(n)$ denote the total cost of calling `INCREMENT` n times over an initially 0 base-3 counter. Use amortized analysis to show that $C(n) \leq (3/2)n$ for any positive integer n. To simplify your work, assume that the counter will not overflow during this process.

   Note: I will give the potential function as a hint after the problem statement, though you will learn more if you try to come up with your own potential function.

   (iii) [5 marks] Prove that for any c such that $C(n) \leq cn$ for any positive integer n, the inequality $c \geq 3/2$ holds. Again, assume that the counter will not overflow during this process.

   Hint: Let D_i be the counter after the i-th `INCREMENT`, and Let $|D_i|_d$ be the number of digit d in D_i for $d \in \{0, 1, 2\}$. Define $\Phi(D_i) = |D_i|_1/2 + |D_i|_2$.

4. [15 marks] In this problem, we consider two stacks, X and Y. n and m denote the sizes of X and Y (here the size of a stack is the number of elements currently in the stack), respectively. We maintain these two stacks to support the following operations:

   `PushX(a)`: Push element a onto stack X;

   `PushY(a)`: Push element a onto stack Y;

`MultiPopX(k)`: pop $\min(k, n)$ elements out of stack X;

`MultiPopY(k)`: pop $\min(k, m)$ elements out of stack Y;

`Move(k)`: pop $\min(k, n)$ elements out of stack X, and each time an element is popped, it is pushed onto stack Y.

We represent each stack using a doubly-linked list, so that `PushX`, `PushY`, and a single pop operation on either stack can be performed in $O(1)$ worst-case time.

(i) [6 marks] What is the worst-case running time of `MultiPopX(k)`, `MultiPopY(k)` and `Move(k)`?

(ii) [9 marks] Perform amortized analysis to show that each of these five operations uses $O(1)$ amortized time.

Hint: Define a potential function that makes use of both n and m. In one possible solution, under the potential function of this method, some operations may have negative amortized cost. This is however fine. Why? Think about what conclusion can be drawn when an operation has negative amortized cost with respect to a particular potential function that starts out at 0 and always remains nonnegative.

DALHOUSIE UNIVERSITY
CSCI 6057 - ADVANCED DATA STRUCTURES
WINTER 2022 - ASSIGNMENT 1

1. SOLUTION:

① pop(S):

Given: pop(S) → removes and returns the top element from the sorted stack S.

Using potential method:

Let
$\bar{c}_i$ = amortized cost
c_i = actual cost
x = no. of elements in the stack before any operation.

Φ_i = potential energy AFTER the i^{th} operation
Φ_{i-1} = potential energy BEFORE the i^{th} operation

Now, after pop(S), there will be $(x-1)$ elements remaining on the stack, i.e. the potential energy after i^{th} operation

actual cost $(c_i) = 1$

$$\therefore \bar{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

$$= 1 + [x-1] - x \quad = 0$$

$$\therefore \bar{c}_i = O(1) \qquad - ①$$

Hence proved that, since only head of the stack is removed, it takes constant time irrespective of the size of the stack i.e. O(1).

② push (S, x):

Aggregate analysis:

Suppose we have 'k' items on stack, and item 'x' that is pushed is greater than other items, we will have to compare each item and remove all too, therefore it will take O(k) time.

But each item can be removed at most once. If we charge '3' for every push operation, the first '2' are charged in the case when we compare head of the stack with x and x is greater in value and add it to the stack.

The last '1' credit is used when x being in the stack is removed and compared further.

During the comparision, two of the situations happen as follows:

① Item < x [i.e. charge 2 is deducted/charged]
② Item > x

For ② case, we use the last '1' credit to remove it from the list. Hence, no longer need of comparision.

So, for every push, the total time for 'n'

pushes is at most $3n \cong O(n)$

$\therefore$ Amortized cost $= O(n)/n = O(1)$ — (2)

From equation (1) and (2), it is proved that pop(S) and push(S,x) take constant time, thus their amortized cost is O(1).

2. SOLUTION:

Given the scheme, let the ~~amortized~~ cost (a_0) be as follows:

$$a_0 = \begin{cases} 1 & \text{when the element is inserted WITHOUT resizing} \\ (2N + N) = 3N & \text{when array is resized.} \end{cases}$$

size of the array alloted

no. of elements copied from previous array

Note: → ($\star$ $n = N$)

for worst case for N elements, as we do N insertions we get

$$= N * 3N$$

no. of elements

cost of insertion in resized array

$$= 3N^2$$

$\therefore \Omega(N^2)$, despite the constant value c.

3. SOLUTION:

(i) pseudocode for INCREMENT in base-3 counter:

For a K-bit base-3 counter:

```
INCREMENT( A[0.... K-1])

        i ← 0
        while i<K and A[i] = 2
            do A[i] ← 0
               i ← i + 1
        if i<K
          then A[i] ← A[i] + 1
```

(ii) To prove, $C(n) \leq \frac{3}{2}n$:

Where C(n) = cost of intially 0 bit 3-base counter for n-increments

n => positive integer

Note: Keeping in mind that counter is not overflowing, the conditions will be defined accordingly.

Using aggregate method:

Let's look at the change in bits at each position.

0 - 000
1 - 001
2 - 002
3 - 010
4 - 011
5 - 012
6 - 020
7 - 021
8 - 022
9 - 100
....

Here, bits at 0^{th} position change after every increment i.e. 3^0.

Bits at 1^{st} position change after every 3^{rd} increment i.e. 3^1.

Bits at 2^{nd} position change after every 9^{th} increment i.e. 3^2. and so on.

∴ Total no. of changes = summation of the times everytime bit changes.

Suppose, $n < 3^{K+1}$ for some K, thus total no. of changes in 'n' increments will be no more than or equal to :

$$\therefore C(n) \leq \frac{n}{3^0} + \frac{n}{3^1} + \frac{n}{3^2} + \frac{n}{3^3} + \cdots + \frac{n}{3^K}$$

$$\therefore C(n) \leq n + \frac{n}{3} + \frac{n}{9} + \frac{n}{27} + \cdots + \frac{n}{3^K}$$

$$\therefore C(n) \leq n + n\left(\frac{1}{3} + \frac{1}{9} + \frac{1}{27} + \cdots + \frac{1}{3^K}\right)$$

$$\therefore C(n) \leq n + n\left(\sum_{i=1}^{K} \frac{1}{3^K}\right)$$

$$\therefore C(n) \leq n + n\left(\frac{1/3}{1 - 1/3}\right)$$

$$\therefore C(n) \leq n + \frac{n}{2}$$

$$\therefore C(n) \leq \frac{3n}{2}$$

Hence, $C(n) \leq \frac{3}{2} n$

(iii)

To show $C(n) \leq \frac{3}{2}n$ i.e. cn

where $c \geq \frac{3}{2}$ and n is a positive integer.

Let us assume $c = \frac{3}{2}$ and a value less than $\frac{3}{2}$ i.e.

$c = \left(\frac{3}{2} - \alpha\right)$ where $\alpha > 0$

① $c = \frac{3}{2}$

Using potential method, we prove that when $c = \frac{3}{2}$, $C(n) \leq cn$ holds true.

Let $D_i \in \{0, 1, 2\}$ → counter's state at i^{th} operation

$|D_i|_d$ = no. of digits (d) in D_i

$\therefore \phi_{D_i} = |D_i|_1 / 2 + |D_i|_2$ — ①

$\therefore \phi_{D_{i-1}}$ = potential before i^{th} operation — ②

Now, b_i = no. of digits as 2 at the end from D_{i-1}

Case 1: $D_{i-1} = 0/1/2 \cdots 0/1/2\,0\underbrace{2 \cdots 2}_{b_i}$

Here, we have 0 before the last sequence of digits in b_i.

∴ After 1 increment we have:

$$D_{i-1} = 0/1/2 \cdots 0/1/2\,1\underbrace{0 \cdots 0}$$

Using (1):

$$\therefore \phi(D_i) = \phi_{D_{i-1}} - b_i + \frac{1}{2} \quad \text{--- (3)}$$

Case 2: $D_{i-1} = 0/1/2 \cdots 0/1/2\,1\underbrace{2 \cdots 2}_{b_i}$

Here, we have 1 before the last sequence of digits in b_i.

∴ After 1 increment we have:

$$D_{i-1} = 0/1/2 \cdots 0/1/2\,2\underbrace{0 \cdots 0}$$

Using (1):

$$\phi_{D_i} = \phi_{D_{i-1}} - b_i + 1 - \frac{1}{2}$$

$$\therefore \phi_{D_i} = \phi_{D_{i-1}} - b_i + \frac{1}{2} \quad \text{--- (4)}$$

Here ϕ_{D_i} are equal for both cases.

Actual cost for both cases : $t_i = b_i + 1$

∴ amortized costs : $a_i = t_i + \phi_{Di} - \phi_{Di-1}$

$= b_i + 1 + \phi_{Di-1} - b_i + \frac{1}{2} - \phi_{Di-1}$

$$\boxed{a_i = \frac{3}{2}} \quad - (5)$$

Since $a_i = t_i + \phi_{Di} - \phi_{Di-1}$

we have :

$$\sum_{l=0}^{n} a_l = \sum_{l=0}^{n} t_l + \sum_{l=0}^{n} [\phi_{Dl} - \phi_{Dl-1}]$$

$$= \sum_{l=0}^{n} t_l + \phi_{Dl} - \phi_{D0}$$

$\phi_{Dl} \geq 0$, $\phi_{D0} = 0$

∴ $cn = \sum_{l=0}^{n} a_l \geq \sum_{l=0}^{n} t_l = C(n)$

∴ For $c = \frac{3}{2}$, $C(n) \leq cn$.

(2) $C = \left(\frac{3}{2} - \alpha\right), \alpha > 0$

Let $\delta = \lceil -\log_3 \alpha \rceil$

$\Rightarrow \delta \geq -\log_3 \alpha$

$\Rightarrow -\delta \leq \log_3 \alpha$

$\Rightarrow 3^{-\delta} \leq \alpha$ - (1)

Suppose $n = 3^\delta$

$\Rightarrow Dn = 10\underbrace{\ldots}0$

As previously concluded, for n bits,

1st bit changes → 3^0 time(s)
2nd bit changes → 3^1 times
3rd bit changes → 3^2 times
⋮
n^{th} bit changes → 3^δ times.

$\therefore C(n) = \sum_{K=0}^{\delta} 3^K = 3^{\delta+1} - 1$

Now, from (1) i.e. $3^{-\delta} \leq \alpha$ we get that

$(C - \alpha)n \leq (C - 3^{-\delta})n$

$$\therefore (c-\alpha)n \leq \left(\frac{3}{2} - 3^{-\delta}\right) 3^{\delta}$$

$$\leq \frac{3}{2} \cdot 3^{\delta} - 1$$

$$\leq \frac{3^{\delta+1}}{2} - 1$$

And,

$$\frac{3^{\delta+1}}{2} - 1 < 3^{\delta+1} - 1 = C(n)$$

$\therefore$ From $c = \frac{3}{2}$ and $c = \left(\frac{3}{2} - \alpha\right)$, α is > 0.

We prove that for any c such that $C(n) \leq cn$ for any positive integer n, the equality $c \geq \frac{3}{2}$ holds.

4. SOLUTION:

(i) When MultiPopX(K) pops entire stack, the running time of each operation is equal to the size of the stack (n) = O(n). Similarly, for MultiPopY(K) of size m has running time = O(m).

Move(K), pops element from X and pushes them onto Y. Considering the worst case, we transfer entire stack on X, we use 'n' pops and 'n' pushes for a worst case running time of O(n).

(ii) Given that elements from X stack is popped and pushed onto Y, the potential function can be defined as:

$$\phi(n,m) = 3n + m$$

Intially the potential is zero and for non-empty stacks, the potential > 0

∴ The amortized costs of all the operations:

(1) PushX(a):

$$\bar{c_i} = c_i + \phi_{D_{i+1}} - \phi_{D_i}$$

$$= 1 + [3(n+1) + m] - (3n + m)$$

$$\boxed{\bar{c_i} = 4}$$

(2) PushY(a):

$$\bar{c}_i = c_i + \phi_{D_{i+1}} - \phi_{D_i}$$

$$= 1 + [3n + (m+1)] - (3n+m)$$

$$\boxed{\bar{c}_i = 2}$$

(3) MultiPopX(k):

$$\bar{c}_i = c_i + \phi_{D_{i+1}} - \phi_{D_i}$$

$$= K + [3(n-K) + m] - [3n+m]$$

$$\boxed{\bar{c}_i = -2K}$$

(4) MultiPopY(K):

$$\bar{c}_i = c_i + \phi_{D_{i+1}} - \phi_{D_i}$$

$$= K + [3n + (m-K)] - (3n+m)$$

$$\boxed{\bar{c}_i = 0}$$

(5) Move(K):

$$\bar{c_i} = c_i + \phi_{D_{i+1}} - \phi_{D_i}$$

$$= 2K + [3(n-k) + (m+K)] - (3n+m)$$

$$= 2K + [3n - 3K + m - K] - 3n - m$$

$$\boxed{\bar{c_i} = 0}$$

∴ from above calculations, we deduce all the five operations have $O(1)$ amortized time.

But only MultiPopX(a) had negative cost as even when PushX(a) has paid for a possible transfer, MultiPopX(a) made an unneccessary transfer.