A SOLUTION TO PROBLEM 1 OF HOMEWORK 2,
PREPARED BY GROUP 4.

PART A.

We note that a pointer−based implementation of a heap requires only 4 units of space per key, 1 for a parent pointer, 2 for child pointers, and 1 for the key itself, disregarding the single unit of space required for a pointer to the root. In an array−based implementation of a heap, if K is the maximum ratio of array size to heap size, then the worst case space cost of the implementation will be K units of space per key. Therefore, if K>4 then the array−based implementation will have a worst case space cost greater than the pointer−based implementation, leading one to prefer the pointer implementation unless other considerations dictated otherwise. It is worth noting as well that the pointer−based implementation has a certain advantage over the array−based implementation regarding time cost because the latter incurs time cost due to growing and shrinking, whereas the former does not.

PART B.

The following scheme provides an implementation for a growing and shrinking array−based data structure. *A* is understood to be an array composed of elements of the same type as *element*. The object CAPACITY(A) can be understood as the number of space units constituting A. The object SIZE(A) can be understood as the number of space units in A being "used" at the time SIZE is called. The concept of "use" of a space unit will vary depending on the actual data structure being implemented. In the case of heaps, we view a "heap" as a particular set of space units within the array, rather than the array itself. The formal requirement that we place on the notion "use" is that if "SIZE'(A)" denotes the number of space units in A *not* being "used" at the time SIZE is called, then SIZE(A) + SIZE'(A) = CAPACITY(A).

INSERT(element,A)
1. If (SIZE(A)=CAPACITY(A)) Then AEXPAND(A).
2. INSERT−ELEMENT(element,A).

DELETE(element,A)
1. If (4*SIZE(A)=CAPACITY(A)) Then ACONTRACT(A).
2. DELETE−ELEMENT(element,A).

EXPAND(A)
1. ALLOC(A',2*CAPACITY(A)).
2. For j = 1 to SIZE(A)
3. A'[j]A[j].
4. DEALLOC(A)
5. return A'.

CONTRACT(A)
1. ALLOC(A',(1/2)*CAPACITY(A)).

2. for j = 1 to SIZE(A)
3. A’[j]A[j].
4. DEALLOC(A).
5. return A’.

The operations INSERT−ELEMENT and DELETE−ELEMENT will be dependent on the type of data structure being used. For the case of heaps, these operations can be those described in CLR, pp. 149−150. The operation of “EXTRACT−MAX” corresponds to the deletion of a particular element of the array that must have the property of being the maximum element in the array, whereas other types of DELETE−ELEMENT operations may allow for the deletion of elements with properties other than this property.

We will demonstrate that the amortized time cost incurred by growing and shrinking is $\theta(1)$ per INSERT or DELETE. The symbol “T” will be used to denote time cost. Our first observation is that T(growing and shrinking) = θ(T(copying during growing and shrinking)), because we assume that T(SIZE)=T(CAPACITY)=T(=)=T()=T(ALLOC)=T(*)=T(DEALLOC)=T(return)= $\theta(1)$.

We will add 2 time cost units of credit to the data structure at the completion of each INSERT and DELETE operation. We will subtract 1 time cost unit of credit for each act of copying, specifically, each execution of line 3 of either EXPAND or CONTRACT. By demonstrating that the total credit in the data structure is always nonnegative, we will have shown that the amortized time cost of growing and shrinking is $\theta(1)$ per INSERT or DELETE.

Let N={1,2,3,…}. Let A be an array such that SIZE(A)=0 and CAPACITY(A)=1. Let CREDIT denote the credit at a given point in the data structure embodied in A by means of our above scheme.

CLAIM: for all possible sequences of INSERT and DELETE operations $\{s_j\}_{j\in N}$, for all $j\in N$, CREDIT ≥ 0 at all points during s_j.

PROOF: Let $\{s_j\}_{j\in N}$ be a sequence of INSERT and DELETE operations. Let $j\in N$. Suppose that for all $p\in N$ such that $p<j$, CREDIT ≥ 0 at all points during s_p. This is an Induction hypothesis. We will show that for all points in s_j, CREDIT ≥ 0, thereby establishing our claim by Induction.

> CASE 1: j = 1. Therefore, CREDIT = 0 at the beginning of s_j. Since SIZE(A) = 0 for all points before line 2 of s_j, and credit CREDIT can only be decreased in line 1, there is no possibility of a decrease in CREDIT during s_j. Therefore, for all points in s_j CREDIT ≥ 0. End of CASE 1.
>
> CASE 2: $j\in N\backslash\{1\}$. Let k be the smallest element of N such that $k \geq j$ and an act of copying occurs during s_k. Let h be the greatest element of N such that $h < j$ and an act of copying occurs during s_h. Let m = SIZE(A) at the end of line 1 of s_h. Let $CREDIT_h$ = CREDIT at the end of line 1 of s_h. By assumption (i.e. induction hypothesis), $CREDIT_h \geq 0$. If $j \neq k$, then no acts of copying occurred

during s_j and, by choice of h, CREDIT at all points of $s_j \geq$ CREDIT at all points of s_h . . Therefore, CREDIT at all points of $s_j \geq 0$ if $j \neq k$. In what follows we will show that CREDIT at all points of $s_k \geq 0$, so if j = k it will also be true that CREDIT at all points of $s_j \geq 0$.

> SUBCASE 1. s_h is an instance of INSERT. Therefore, at the beginning of line 1 of s_h , CAPACITY(A) = SIZE(A) = m. Therefore, at the end of line 1 of s_h , CAPACITY(A)=2m. End of SUBCASE 1.
>
> SUBCASE 2. s_h is an instance of DELETE. Therefore, at the beginning of line 1 of s_h , CAPACITY(A) = 4*SIZE(A) = 4m. Therefore, at the end of line 1 of s_h , CAPACITY(A) = (1/2)*4m = 2m. End of SUBCASE 2.

Therefore, at the end of line 1 of s_h , CAPACITY(A)=2m. By choice of h and k,

therefore, CAPACITY(A) = 2m at the beginning of s_k.

> SUBCASE 1. s_k is an instance of INSERT. Since s_k involves an act of
>
> copying, SIZE(A) = CAPACITY(A) = 2m at the beginning of s_k. Therefore, (SIZE(A) at the beginning of s_k) – (SIZE(A) at the end of step 1 of s_h) = 2m – m = m. Since there are k−h opportunities to increase SIZE(A) by at most 1 between these two points, m ≤ k–h. CREDIT at the beginning of s_k = $CREDIT_h$ + 2(k−h), because k−h INSERT or DELETE operations occur between between the end of line 1 of s_h and the beginning of s_k and each such operation results in the addition of 2 to CREDIT. During s_k, CREDIT is reduced by exactly 2m. Therefore, CREDIT at all points of $s_k \geq CREDIT_h + 2(k-h) - 2m \geq CREDIT_h \geq 0$. End of SUBCASE 1.
>
> SUBCASE 2. s_k is an instance of DELETE. Since s_k involves an act of copying, 4*SIZE(A) = CAPACITY(A) = 2m at the beginning of s_k, so SIZE(A) = m/2 at this point. Therefore, (SIZE(A) at the end of step 1 of s_h) – (SIZE(A) at the beginning of s_k) = m – m/2 = m/2. Since there are k−h opportunities to decrease SIZE(A) by at most 1 between these two points, m/2 ≤ k−h. CREDIT at the beginning of s_k = $CREDIT_h$ + 2(k−h), because k−h INSERT or DELETE operations occur between between the end of line 1 of s_h and the beginning of s_k and each such operation results in the addition of 2 to CREDIT. During s_k, CREDIT is reduced by exactly m/2. Therefore, CREDIT at all points of $s_k \geq CREDIT_h + 2(k-h) - m/2 \geq CREDIT_h \geq 0$. End of SUBCASE 2.

Therefore, CREDIT at all points of $s_k \geq 0$. As discussed in the beginning of CASE 2, we can conclude that CREDIT at all points of $s_j \geq 0$. End of CASE 2.

By Induction, we conclude that for all $j \in N$, CREDIT ≥ 0 at all points during s_j.
Recalling that $\{s_j\}_{j \in N}$ was an arbitrary sequence of operations, the proof is complete.