

I. Fill in the asymptotic analysis of each method. Provide the worst case for every method, and the amortized cost only if it is different

e indicates an Element, p indicates a Position, and i indicates an Index.

Sequence	Worst Case		Amortized	
	Linked	Array	Linked	Array
atIndex(i)	$O(n)$	$O(1)$		
indexOf(p)	$O(n)$	$O(1)$		
set(i, e)	$O(n)$	$O(1)$		
set(p, e)	$O(1)$	$O(1)$		
addBefore(p, e)	$O(1)$	$O(n)$		
addLast(e)	$O(1)$	$O(n)$		$O(1)$
add(i, e)	$O(n)$	$O(n)$		
remove(p)	$O(1)$	$O(n)$		
remove(i)	$O(n)$	$O(n)$		

II. Read the following algorithm, and answer the questions about it.

Algorithm:Sequence-Thinner(S)

Input : S - Sequence of n Comparable “Things”

```

1 for  $i \leftarrow 0$  to  $S.size()$  do
2   | PQ.insert( $S.get(i), i$ )
3 end

4 while ( $S.size() > (n/2)$ ) do
5   | Position  $mx \leftarrow S.atIndex(0)$ 
6   | for  $i \leftarrow 0$  to  $S.size()$  do
7     | if  $S.get(i) \geq mx.element()$  then
8       | |  $mx \leftarrow S.atIndex(i)$ 
9       | end
10  | end
11  |  $S.remove(mx)$ 
12  | Entry  $min \leftarrow PQ.removeMin()$ 
13  |  $S.remove(min.getValue())$ 
14 end

15 return  $PQ.min().getKey()$ 

```

1) What is the worst-case running time of Sequence-Thinner(S), if PQ is an Ordered Sequence implementation and S is an Array Sequence implementation?

$O(n^2)$

- The first for loop runs n times
- $PQ.insert()$ is an $O(n)$ operation, and $S.get()$ is $O(1)$. Since they are done for each loop, $n*O(n) + n*O(1) = O(n^2)$
- The while loop runs $n/4$ times (where S ranges from size n to size $n/2$, removing 2 each time.)
- The for loop inside the while loop runs a different number of times in each iteration. The sum of these is $(n + n/2) * (n/4) = n^2/4 + n^2/8$ total times inside this loop.
- Inside the inner for loop is $S.atIndex()$, an $O(1)$ operation - since it is done in each loop - $(n^2/4 + n^2/8)*O(1) = O(n^2)$
- $S.remove(p)$ is $O(n)$, $PQ.removeMin()$ is $O(1)$, and $S.remove(i)$ is $O(n)$. Since they are in the while loop, $n/4*O(n) + n/4*O(1) + n/4*O(n) = O(n^2)$

2) What if PQ is an Ordered Sequence implementation and S is a Linked Sequence implementation?

$O(n^3)$

- $PQ.insert()$ is an $O(n)$ operation, and $S.get()$ is $O(n)$. Since they are done for each loop, $n*O(n) + n*O(n) = O(n^2)$
- Inside the inner for loop is $S.atIndex()$, an $O(n)$ operation - since it is done in each loop - $(n^2/4 + n^2/8)*O(n) = O(n^3)$
- $S.remove(p)$ is $O(1)$, $PQ.removeMin()$ is $O(1)$, and $S.remove(i)$ is $O(n)$. Since they are in the while loop, $n/4*O(1) + n/4*O(1) + n/4*O(n) = O(n^2)$

3) What if PQ is an Unordered Sequence implementation and S is an Array Sequence implementation?

$O(n^2)$

- $PQ.insert()$ is an $O(1)$ operation, and $S.get()$ is $O(1)$. Since they are done for each loop, $n*O(1) + n*O(1) = O(n)$
- The inner for loop is exactly the same as in question 1 - $O(n^2)$
- $S.remove(p)$ is $O(n)$, $PQ.removeMin()$ is $O(n)$, and $S.remove(i)$ is $O(n)$. Since they are in the while loop, $n/4*O(n) + n/4*O(n) + n/4*O(n) = O(n^2)$

4) Assuming we use an Array Sequence, which of the two versions of the *Priority Queue* is more efficient for PQ (consider total time, not just Asymptotic)?

The Unordred version of the PriorityQueue is better, because every item is added to the list, but only $n/4$ items are removed.

5) If the values 32 to 17 were in the original sequence (ordered highest to lowest), what would be returned?

21

The while loop removes the largest and smallest values from the sequence in each iteration. Since there are 16 items, that would remove the highest 4 (29-32) and the lowest 4 (17-20). Then the loop would terminate ($S.size() = 8$, $n = 16$) The next lowest Key in the PriorityQueue is 21, therefore, 21 would be returned.