Fall 2005: CSI2114 – ASSIGNMENT 1

Prof. WonSook LEE

Ex 1 (20 points)

(10 points) (a) Fill a table showing a series of following queue operations and their effects on an initially empty queue *Q* of integer objects. Here *Q* is implemented with an Array of size 7.

Operation	Output	Q
enqueue (4)		f→ 4 ← R
dequeue ()	4	f→ ← R
dequeue ()	EmptyStakeException	
enqueue (44)		f→ 44← R
enqueue (7)		f→ 44,7← R
enqueue (6)		f→ 44,7,6← R
dequeue ()	44	f→ 7,6← R
isEmpty()	False	f→ 7,6← R
enqueue (3)		f→ 7,6,3← R
enqueue(5)		f→ 7,6,3,5← R
dequeue ()	7	f→ 6,3,5← R
dequeue ()	6	f→ 3,5← R
dequeue ()	3	f→ 5← R
dequeue ()	5	f→ ← R
enqueue(32)		f→ 32← R
enqueue(39)		f→ 32,39← R
enqueue(9)		f→ 32,39,9← R
size()	3	f→ 32,39,9← R
enqueue (32)		f→ 32,39,9,32← R
size()	4	f→ 32,39,9,32← R
dequeue ()	32	f→ 39,9,32 ← R
enqueue (6)		f→ 39,9,32,6← R
enqueue (5)		f→ 39,9,32,6,5← R
dequeue ()	39	f→ 9,32,6,5← R
front()	9	f→ 9,32,6,5← R
size()	4	f→ 9,32,6,5← R
enqueue (9)		f→ 9,32,6,5,9← R

(10 points) (b) Write the elements of the resulted array

	9	32	6	5	9	

Ex 2 (30 points)

(10 points) (a) Describe how to implement the queue ADT using two stacks. That is: write pseudo code algorithms which implement the *enqueue()* and *dequeue()* methods of the queue using the methods of the stack.

Assume that we have two stacks named ***S1*** and ***S2***.

S1 will have store the queue elements assuming the top element is the

stack is the front of the queue.
S2 will be used as temporary storage when enqueuing new element.

Explanation:
dequeue
When we want to dequeue an element, we can just perform a pop on ***S1***.
enqueue
While when we want to enqueue an element ***o***, we should pop all the elements from ***S1*** and push them in ***S2***, then push the element ***o*** in ***S1*** and then pop all the elements from ***S2*** and push them back in ***Q1***.

The code follows:

```
Algorithm dequeue()
If S1.isEmpty()  then
        ERROR
Return S1.pop()


Algorithm enqueue(object o)
While  not S1.isEmpty
      S2.push(S1.pop())

S1.push(o)

While not S2.isEmpty
      S1.push(S2.pop())
```

(5 points) (b) What are the running times of your *dequeue()* and *enqueue()* algorithms?
dequeue will be O(1)
enqueue will be O(n)

(10 points) (c) Describe how to implement the stack ADT using two queues. That is: write pseudocode algorithms which implement the *pop()* and *push()* methods of the stack using the methods of the queue.
3 marks for the assumption of the top of the stack with respect to the queue and other assumptions.
3 marks for push (1 to check if the queue is empty first)
4 marks for pop (1.5 mark for each loop, 1 mark for enqueuing o)

Assume that we have two queues named ***Q1*** and ***Q2***.

Q1 will have store the stack elements assuming the front element is the

	top of the stack. ***Q2*** will be used as temporary storage when pushing new element. **Explanation** **pop** When we want to pop an element, we can just perform dequeue on ***Q1*** (Thus we are taking the top of the stack). **push** When we want to push an element ***o***, we should dequeue all the elements from ***Q1*** and enqueue them in ***Q2***, then enqueue the element ***o*** in ***Q1*** and then dequeue all the elements from ***Q2*** and enqueue them in ***Q1***. The code follows: ***Algorithm Pop()*** ***If Q1.isEmpty() then*** ***ERROR*** ***Return Q1.dequeue()*** ***Algorithm push(object o)*** ***While not Q1.isEmpty*** ***Q2.enqueue(Q1.dequeue())*** ***Q1.enqueue(o)*** ***While not Q2.isEmpty*** ***Q1.enqueue(Q2.dequeue())*** **(5 points)** (d) What are the running times of your *pop()* and *push()* algorithms? Pop will be O(1) Push will be O(n)
Ex 3 (15 points)	Give an example of a positive function *f(n)* such that *f(n)* is neither $O(n^2)$ nor $\Omega(n^2)$. Explain both assertions. *Let* $f(n) = n^2(\sin(n)+1)$ Notice that: • *f(n)* is positive function. • For n=(x+2rΠ) where 0≤x≤180 and r=1,2,3, …. f(n) is bounded from below by n^2 curve • For n=(x+2rΠ) where 180≤x≤360 and r=1,2,3, ….

	f(n) is bounded from above by n^2 curve More formal explanation: Thus there is no c and n_o for which $f(n) \leq c.\ n^2$ for all $n \geq n_o$ i.e. $n^2(\sin(n)+1) \leq c.\ n^2$ ➔ $\sin(n) \leq c-1$ (impossible to hold any c. e.g. c=1) Then f is not $O(n^2)$ Thus there is no c and n_o for which $f(n) \geq c.\ n^2$ for all $n \geq n_o$ i.e. $n^2(\sin(n)+1) \geq c.\ n^2$ ➔ $\sin(n) \geq c-1$ (impossible for any c. e.g c=3) Then f is not $\Omega(n^2)$
Ex 4 (20 points)	Give a big-Oh characterization, in terms of n, of the running time of the following method. Show your analysis! *public void Ex(int n)* *int a = 1;* *for (int i = 0 ; i < n*n ; i++)* *for (int j = 0; j <= i; j++)* *if(a <= j)* *a = i;* *}* • Initializing variable a at the beginning takes $O(1)$ time. • There are two nested loops, which are controlled by counters i and j. The body of the outer loop, controlled by counter i, is executed m times, $m = (n * n)$, for $i = 0,1,\ldots m$. This implies that the increment and testing of counter i, contribute a number of primitive operations proportional to m, that is $O(m)$ time. • The body of the inner loop, which is controlled by counter j, is executed $i+1$ times, depending upon the current value of the outer loop counter i. This the statement, (if condition check) is executed 1+2+3….+(m+1) times. We know that 1+2….+(m+1) = (m+1) (m+2)/2, which implies that the statement in the inner loop contributes $O(m^2)$ time. The running time of the given code (in terms of m) is given by the sum of three terms, $O(1)$, $O(m)$ and $O(m^2)$ which implies the running time is $O(m^2)$. Now replacing m by n, the running time of the algorithm becomes

	$O(n^4)$.
Ex 5 (15 points)	Prove that $5\,n^3 + n^2 + 5000\,n + 2^{n+1}$ is $O(2^n)$ using the formal definition of O. By the definition of big-Oh, we need to find a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $5\,n^3 + n^2 + 5000\,n + 2^{n+1} \leq c.2^n$ for $n \geq n_0$. *It is easy to see that* $5\,n^3 \leq 2^{n+1}$ *for large n, say* $n \geq 20$ *And* $n^2 \leq 2^{n+1}$ *for any* $n > 1$ *And* $5000\,n \leq 2^{n+1}$ *for also* $n \geq 20$ *Thus* $5\,n^3 + n^2 + 5000\,n + 2^{n+1} \leq 2^{n+1} + 2^{n+1} + 2^{n+1} + 2^{n+1}$ *Implies* $5\,n^3 + n^2 + 5000\,n + 2^{n+1} \leq 4.\ 2^{n+1} = 8.\ 2^n$ One possible solution is choosing $c = 8$ and $n_0 = 20$ at which $5\,n^3 + n^2 + 5000\,n + 2^{n+1} \leq c.2^n$