NetID: hxie13 QuizID: 86757 Score: 4 / 5 Answer Source: PrairieLearn

1. In implementing Stack ADT, using which of the following data structure gives worst asymptotic runtime for `push`? (Assume we require to `push` at the end of list or array)

A. [Correct Answer] [Your Answer] Singly linked list with head pointer only.
B. All options provide the same runtime.
C. Singly linked list with head and tail pointer.
D. Array (size of array larger than possible elements in stack).
E. Doubly linked list with head and tail pointer.

2. What is the result of executing the following code snippet?

Assume all required libraries are included and no compile-time/runtime errors occur.

```
int main() {
   list<int> myList;
   for (int i=1; i<6; i++)
      myList.push_back(i);

   for (list<int>::iterator it = myList.begin(); it != myList.end(); it++ )
      *it = *it * 3;

   for (list<int>::iterator it = myList.begin(); it != myList.end(); it++ )
      cout << *it << " ";

   return 0;
}
```

A. `3 6 9 12`
B. [Correct Answer] [Your Answer] `3 6 9 12 15`
C. `1 2 3 4 5`
D. None of the other options is correct.
E. `1 2 3 4`

3. We have implemented the Stack ADT as an array. Every time the array is full, you resize the array creating a new array that can hold 3 elements more than the previous array and copy values over from the old array. What is the total running time for n pushes to the stack.

A. $O(n)$.
B. $1/3 * O(n)$.
C. [Correct Answer] $O(n^2)$.
D. [Your Answer] $O(\log n)$.
E. $O(1)$.

4. Suppose we have implemented the Queue ADT as a singly-linked-list with `head` and `tail` pointers and no sentinels. Which of the following best describe the tightest running times for the functions `enqueue` and `dequeue`, assuming there are $O(n)$ items in the list, and that the front of the queue is at the `head` of the list?

A. $O(1)$ for `enqueue` and $O(n)$ for `dequeue`.
B. None of the options is correct
C. $O(n)$ for `enqueue` and $O(1)$ for `dequeue`.
D. [Correct Answer] [Your Answer] $O(1)$ for both.
E. $O(n)$ for both.

5. Suppose `queue<int> q` contains 6 elements `1, 2, 3, 4, 5, 6` (enqueued in that order). What is the result of executing the following code snippet? (Assume member function `front()` returns the value found at the front of the queue without removing it.)

```
for(int i = 1; i<7; i++){
    if(i%2==0) {
        q.enqueue(q.front());
        q.dequeue();
    }
}
```

A. `q` remains the same.
B. The elements `q` are reversed.
C. The even numbers in `q` are reversed.
D. [Correct Answer] [Your Answer] The front half of the original `q` is now at the back half.
E. The odd numbers in `q` are reversed.