**ACE Tutorial 4**

**Question 1: Stacks and Queues**

Consider an array-based queue, where the underlying array of size *N* is used in a circular fashion. We keep track of two variables: *f* referring to the index of the front element and *sz* referring to the number of stored elements. When the queue has *fewer than* *N* elements, the array index *r = (f + sz) mod N* is the first empty slot past the rear of the queue.

Consider a queue that has an underlying array *A* of size 5. Fill in the following *f*, *sz* and *r* values, and show the state of the array *A* after each operation.

* Initial State of *A*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Dequeue

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 16

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Dequeue

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Dequeue

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 19

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 22

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

* Enqueue 25

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Element |  |  |  |  |  |

|  |  |
| --- | --- |
|  | value |
| *f* |  |
| *sz* |  |
| *r* |  |

What happens here? Is *r* referring to an empty cell? Can we add more elements to the array?

**Question 2: Lists**

Consider a growable array-based array list. Let *push(o)* be the operation that adds an element *o* at the end of the list. For the pseudocode of the *push(o)* algorithm, see Slide 13 in Lists.pdf. When the array is full, we replace the array with a larger one. There are two commonly used strategies which determine the size of the new array.

***Incremental strategy***: when an array of size *n* is full, we replace it with a new array of size *(n+c)*, where *c* is a constant.

***Doubling strategy:*** when an array of size *n* is full, we replace it with a new array of size *2n*.

Assume that when the array is not full, adding an element into it takes a constant time 1. Fill in the two tables below, which illustrate the process of performing a series of *push(o)* operations over an initial array which is empty and of size 1, using the incremental strategy and the doubling strategy, respectively. For the incremental strategy, we set *c=3*.

***Incremental strategy***, *c=3*

|  |  |  |  |
| --- | --- | --- | --- |
| Array size | Push *i-*th element | Time for adding elements | Time for copying elements |
|  | 1 |  |  |
|  | 2 |  |  |
|  | 3 |  |  |
|  | 4 |  |  |
|  | 5 |  |  |
|  | 6 |  |  |
|  | 7 |  |  |
|  | 8 |  |  |
|  | 9 |  |  |
|  | 10 |  |  |
|  | 11 |  |  |
|  | 12 |  |  |

Let *m* denote the total number of push operations in the series, *k* denote the number of times of increasing the array size. Can you express the relationship between *m* and *k* using *c*?

Let *T(m)* denote the total time for performing these *m* push operations. How to express *T(m)* using *m, k* and *c*? Which big-Oh class does *T(m)* belong to? Which big-Oh class does *T(m)/m* belong to?

***Doubling strategy***

|  |  |  |  |
| --- | --- | --- | --- |
| Array size | Push *i-*th element | Time for adding elements | Time for copying elements |
| 1 | 1 | 1 | 0 |
|  | 2 |  |  |
|  | 3 |  |  |
|  | 4 |  |  |
|  | 5 |  |  |
|  | 6 |  |  |
|  | 7 |  |  |
|  | 8 |  |  |
|  | 9 |  |  |
|  | 10 |  |  |
|  | 11 |  |  |
|  | 12 |  |  |
|  | 13 |  |  |
|  | 14 |  |  |
|  | 15 |  |  |
|  | 16 |  |  |

Let *m* denote the total number of push operations in the series, *k* denote the number of times of increasing the array size. Can you express the relationship between *m* and *k*?

Let *T(m)* denote the total time for performing these *m* push operations. How to express *T(m)* using *m* and *k*? Which big-Oh class does *T(m)* belong to? Which big-Oh class does *T(m)/m* belong to?