```c
#ifndef __QUEUE_H__
#define __QUEUE_H__

#include <limits.h>
#include <stdlib.h>

typedef struct {
    int front, rear, size;
    unsigned capacity;
    void** job;
} Queue;

Queue* createQueue(unsigned capacity)
{
    Queue* queue = (Queue*)malloc(
        sizeof(Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;

    // This is important, see the enqueue
    queue->rear = capacity - 1;
    queue->job = (void*)malloc(
        queue->capacity * sizeof(int));
    return queue;
}

int isFull(Queue* queue)
{
    return (queue->size == queue->capacity);
}

// Queue is empty when size is 0
int isEmpty(Queue* queue)
{
    return (queue->size == 0);
}

void enqueue(Queue* queue, void* item)
{
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1)
                  % queue->capacity;
    queue->job[queue->rear] = item;
    queue->size = queue->size + 1;
}

void* dequeue(Queue* queue)
{
    if (isEmpty(queue))
        return NULL;
    void* item = queue->job[queue->front];
    queue->front = (queue->front + 1)
                   % queue->capacity;
    queue->size = queue->size - 1;
    return item;
}

void* random_dequeue(Queue *queue)
{
    if (isEmpty(queue))
        return NULL;
    else if (queue->size == 1)
        return dequeue(queue);

    int lower_limit = 0;
    int upper_limit = queue->size - 1;

    int random_index = (rand() % (upper_limit - lower_limit) + 1) + lower_limit;

    /* swap the random index with the one at front and then call dequeue */

    /* get the pointer at random index, and make a copy of it*/
    void *temp = queue->job[random_index];

    /* the pointer at random index points to same place as front pointer*/
    queue->job[random_index] = queue->job[0];

    /* front pointer now points where the old random index pointed to */
    queue->job[0] = temp;

    /* return normal dequeue - random pointer will be returned */
    return dequeue(queue);
}

#endif
```