(c) Why do you think hash functions produce a fixed-size output? Explain your answer in one sentence. (2 marks)

Hash functions use a fn. such that it lies within the range of size of the hash table and thus produce fixed-size output. Ex- $f(x) = n \bmod 19$ producing a fixed size output

(d) Explain the worst case complexity for search in Hashing with chaining. Explain your answer in a sentence. (2 marks)

Worst-case complexity in Hashing with chaining is $O(n)$ as it may happen that all the values are chained at one index only.

Q2. [15 Marks] Provide an implementation of the Queue data structure using a dynamic array. Use dynamic memory allocation of the array for the storage of elements in the Queue. The memory allocated for the array should grow in size as the size of the queue grows and it should shrink in size if the queue size becomes small. Your implementation should run in O(1) amortized time for enqueue and dequeue operations.

(a) Write your class definitions and the pseudocode for the enqueue and dequeue functions.

```
class Queue {
    int* arr
    int f, b, capacity, size
    Queue() {
        arr = new int[16];
        capacity = 16 // A minm arbitrary initial size
        size = 0;
        f = -1 ; // Front element (Initially not present)
        b = 0 ; } // back element
    Enqueue ⇆ (int x) {
        b = (b+1) mod capacity ;
        if(b == f) { // queue full
            capacity = 2 * capacity
(Reallocation ← int* temp = new int[capacity]
when
size = capacity)  ~~for i in range~~ int c = f ;
                  while(c != b) { temp[c-f] = arr[c]
                                  c = (c+1) % cap size
```

```
delete arr
arr = temp
f = 0 , b = size-1 ;
arr [++b] = x ; }
size++ ; }
Dequeue(){ f = (f+1) %. capacity ; size--;
if (size < capacity/4) { int* temp = new arr [capacity/2];
// copy elements from arr to temp
int c = f
while (c != b) {
temp [c-f] = arr [f]
delete arr ;
arr = temp f = 0
b = size-1 ; }
else { arr [b] = x } // Reallocation not required
```

(To prevent Hysterisis)

(b) Prove that the amortized time complexity of your enqueue and dequeue operations is O(1)

- Let us suppose that we perform n enqueue and dequeue operations, then at max $\frac{n}{c}$ of them requires nk time involving copying while other requires some (n-c) constant a time for enqueue and dequeue

Amortized Total time for n enqueue and dequ operation

$$= \frac{a \times (n-c) + c * nk}{n}$$

$$= a(1-c) + ck$$

$$= O(1) \quad (\because a, c, k \text{ are constant})$$

Thus Amortized time = O(1)