Pouria Moradpour - 4024023040

Question 1

(A)

I first define an integer value `pushed_counter`, and increment it with each push to the max-heap.
By storing another value in each node (for example by using an array), `priority` which would store `pushed_counter` when it is being pushed, we now have the `priority` which is actually the order of pushing (Adding nodes).
The created max heap works based on the numbers stored in `priority` and so the last added node is the biggest and so is now in the head, which is what we need to pop in order to achieve stack properties, essentially implementing the LIFO principle.

$StackPush$: Inserting the element into the max-heap (which is $O(log(n))$ time as discussed in the lectures.)

$StackPop$: removing and returning the head of the max-heap (also takes $O(log(n))$ due to *heapify* process, as discussed in the lectures)

(B)

I first define an integer value `pushed_counter`, and increment it with each enqueue to the max-heap.
By storing another value in each node (for example by using an array), `-priority` such that `priority` would store `pushed_counter` when it is being pushed, we now have the `priority` which is actually the order of pushing (Adding nodes). Therefore `-priority` is the negative value of `pushed_counter` in each enqueue. ``The created max heap works based on the numbers stored in``-priority` and so the first added node is the biggest and so is now in the head, which is what we need to $dequeue$ in order to achieve queue properties, essentially implementing the FIFO principle.

$EnQueue$: Inserting the element into the max-heap (which is $O(log(n))$ time as discussed in the lectures.)

$DeQueue$: removing and returning the head of the max-heap (also takes $O(log(n))$ due to *heapify* process, as discussed in the lectures)

Question 2

$$MH = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]$$

Visual Representation of the Max-Heap:

```
              15
           /     \
          /       \
         /         \
       13           9
     /   \        /   \
    5     12     8     7
   / \   / \    /
  4   0 6   2  1
```

Perform Delete-Max Operation

Remove the Root and Replace with the Last Element

1. Remove 15 (root).
2. Replace 15 with 1, the last element.

New heap:

```
            1
         /     \
       13        9
     /   \     /   \
    5     12  8     7
   / \   / \
  4   0 6   2
```

Perform Heapify Down

Start heapifying down from the root (1).

1. Compare 1 with its children 13 and 9. The largest child is 13.

- Swap 1 and 13.

```
          13
       /      \
     1         9
   /   \     /   \
  5    12   8     7
 / \   / \
4   0 6   2
```

2. Compare 1 with its new children 5 and 12. The largest child is 12.

- Swap 1 and 12.

```
          13
       /      \
     12        9
   /   \     /   \
  5     1   8     7
 / \   / \
4   0 6   2
```

3. Compare 1 with its new children 6 and 2. The largest child is 6.

- Swap 1 and 6.

```
          13
       /      \
     12        9
   /   \     /   \
  5     6   8     7
 / \   / \
4   0 1   2
```

Question 3

Array:

$$[23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$$

The corresponding complete binary tree for this array:

```
          23
        /     \
      17        14
    /   \      /   \
   6     13   10    1
  / \   /
 5   7  12
```

in level k=3, the right child of 6, is 7 and is bigger. This indicates that the given array is **not** a correct max-heap.

Question 4

This question is solved using $Double\ Hashing$ whose definition is:

$$h(k,i) = (h_1(k) + ih_2(k))\ mod\ n$$

by substituting the functions $h_1(k)$ and $h_2(k)$ we have the desired hash function:

$$h(k,i) = (kmod13 + i(1 + kmod11))\ mod\ 13$$

now, for $k = 14$ we calculate $h(14,1)$ and increment 1 until we find a free place for $k = 14$:

$$h(14,1) = (14mod13 + 1(1 + 14mod11))\ mod\ 13\ =\ 5$$

5 is already occupied by 78, so we calculate $h(14,2)$:

$$h(14,2) = (14mod13 + 2(1 + 14mod11))\ mod\ 13\ =\ 9$$

9 is already occupied by 80, so we calculate $h(14,3)$:

$$h(14,3) = (14mod13 + 3(1 + 14mod11))\ mod\ 13\ =\ 0$$

Since 0 is empty in the hash table, we have found the desired place for 14 :)

Question 5

$$h(k) = kmod(2^p - 1)$$

We know that:

$$(a + b + \ldots)modm = ((amodm) + (bmodm) + \ldots)modm$$

I am using string 1 and string 2 to represent an permutations of two **arbitrary** characters

$$String_1 = AB$$

$$String_2 = BA$$

For String1 = "AB":

$$hash = (A \times 2^p + B) mod (2^p - 1)$$

$$= ((A \times 2^p) mod (2^p - 1) + B mod (2^p - 1)) mod (2^p - 1)$$

$$= (A + B) mod (2^p - 1)$$

(as we know $x \times 2^p \; mod(2^p - 1) = x$)

5. For String2 = "BA":

$$hash = (B \times 2^p + A) mod (2^p - 1)$$

$$= (B + A) mod (2^p - 1)$$

and so their $h(k)$ value is identical. I have not shown it in n character form as it would be way too long, but it is clear that we can generalize this to a n-character string's permutations as well, essentially showing that the defined hash functions causes conflict between permutation of a string.

Question 6

By writing the process of linear probation for all 4 options, I have concluded that option 2 is the one that cannot be correct even for a single chosen sequence of insertion.

Linear Probation Process

We first write the linear probation process that must have happened for each key for it current place to make sense.

- **C**: hash 3, position 0
  By linear probation, 0 must have been the first free place in the table.
  3, 4 ,5, and 6 must have been taken .
- **E**: hash 5, position 1
  By linear probation, 1 must have been the first free place in the table.
  5, 6, and 0 must have been taken.

- **B**: hash 5, position 2
  By linear probation, 2 must have been the first free place in the table.
  5, 6, 0, and 1 must have been taken.
- **G**: hash 3, position 3
  Position 3 was not taken at the time of hashing G.
- **F**: hash 4, position 6
  By linear probation, 6 must have been the first free place in the table.
  4 and 5 must have been taken.
- **D**: hash 5, position 4
  By linear probation, 4 must have been the first free place in the table.
  5, 6, 0, 1, 2, and 3 must have been taken.
- **A**: hash 3, position 6
  By linear probation, 6 must have been the first free place in the table.
  3, 4, and 5 must have been taken.

Conclusion

As mentioned in the assignment document, the process has started when the table was empty. So, the first insertion must not have needed linear probation and should have been placed with $i = 1$. The only key satisfying this is G, clear from the linear probation process I have written above. Now we are certain that the first insertion was the insertion of G.
By following the same logic, the second element should have had **at most** $i = 2$ (as there were only one key inserted beforehand). By looking at the the linear probation process I have written above, we fail to find such key. Therefore, the table in option 2 can't be correct.