DSA - Assignment 3

Nguyễn Đình Hoàng - 20021361

DSA - Assignment 3

Name: Nguyễn Đình Hoàng - 20021361

1. Sort the following functions in the ascending order of Big O notation :

$4nlogn + 2n \to O(nlogn)$	$2^{10} \to O(1)$	$2^{logn} \to O(2^{logn})$
$3n + 100logn \to O(n)$	$4n \to O(n)$	$2^n \to O(2^n)$
$n^2 + 10n \to O(n^2)$	$n^3 \to O(n^3)$	$nlogn \to O(nlogn)$

Sort: $2^n > n^3 > n^2 > nlog(n) > n > 2^{log(n)} > 1$

2. Given an integer number n, your task is to write two different algorithms in pseudo-codes to calculate 2^n, and evaluate the complexity of the algorithms.

Algorithm 1: Calculate 2^x with brute force

```
Data: n ≥ 0
Result: y = x^n
y ← 1 ;                                           // [+1]
X ← x ;                                           // [+1]
N ← n ;                                           // [+1]
for i ← 1 to N do
  | y ← y × X ;                                   // [+1]
end
/* the comparison i < N and the increment i [+2]  */
/* → P(n) = N + 3 ~ O(N)                          */
```

Algorithm 2: Calculate 2^x

```
Data: n ≥ 0
Result: y = x^n
y ← 1;                                            // [+1]
X ← x;                                            // [+1]
N ← n;                                            // [+1]
while N ≠ 0 do
  | if N is even then
  |   | X ← X × X;
  |   | N ← N/2;
  | else
  |   | if N is odd then
  |   |   | y ← y × X;
  |   |   | N ← N − 1;
  |   | end
  | end
end
/* N ← N/2 → P(N) = log_2 N ~ O(log N)            */
```

3. Your task is to write operations of queue data structure in pseudo-codes using an array, then evaluate the complexities of the operations.

Algorithm 3: Queue Pseudo-code

```
def isEmty() is
    if size = 0 then
        return true;
    end
    return false;
end
def isFull() is
    if size = capacity then
        return true;
    end
    return false;
end
def enQueue(value) is
    if isFull() then
        return false;
    end
    rear ← mod((rear + 1), capacity);
    arr(rear) ← value;
    size ← size + 1;
    return true;
end
/* enQueue: only if-else and assigment → O(1)                    */
def deQueue() is
    if isEmty() then
        return false;
    end
    front ← mod((front + 1), capacity);
    size ← size + 1;
    return true;
end
/* deQueue: only if-else and assignment → O(1)                   */
```

4. Your task is to write operations of queue data structure in pseudo-codes using a linked list, then evaluate the complexities of the operations.

Algorithm 4: Queue using Linked List Pseudo-code

```
def enQueue(value) is
    newNode.value ← value;
    if front = NULL then
        front ← newNode;
        rear ← front;
    else
        rear.next ← newNode;
        rear ← newNode;
    end
    def deQueue() is
        tmp ← front;
        front ← front.next;
        delete tmp;
    end
    /* Time complexity of both operations enQueue() and deQueue() is O(1)
       as we only change few pointers in both operations. There is no
       loop in any of the operations.                                  */
end
```

5. Your task is to write operations of stack data structure in pseudo-codes using an array, then evaluate the complexities of the operations.

Algorithm 5: Stack using Array Pseudo-code

```
top ← −1;
def push(value) is
    if top = size − 1 then
        Stack overflow
        return;
    end
    top ← top + 1;
    arr(top) ← value;
end
def pop() is
    if top + 1 == 0 then
        Stack underflow
        return;
    end
    top ← top − 1;
    return arr(top);
end
/* Time complexity of both operations push() and pop() is O(1), only
   assignment in both operations. There is no loop in any of the
   operations.                                                      */
```

6. Your task is to write operations of stack data structure in pseudo-codes using a linked list, then evaluate the complexities of the operations

Algorithm 6: Stack using Array Pseudo-code

```
top ← NULL;
def push(value) is
    newNode.value ← value;
    newNode.next ← newNode;
    top ← newNode;
end
def pop() is
    if top = NULL then
        Stack Empty
        return;
    end
    tmp ← top;
    top ← top.next;
    delete tmp;
end
/* Time complexity of both operations push() and pop() is O(1), as we
   only change few pointers in both operations. There is no loop in any
   of the operations.                                                  */
```