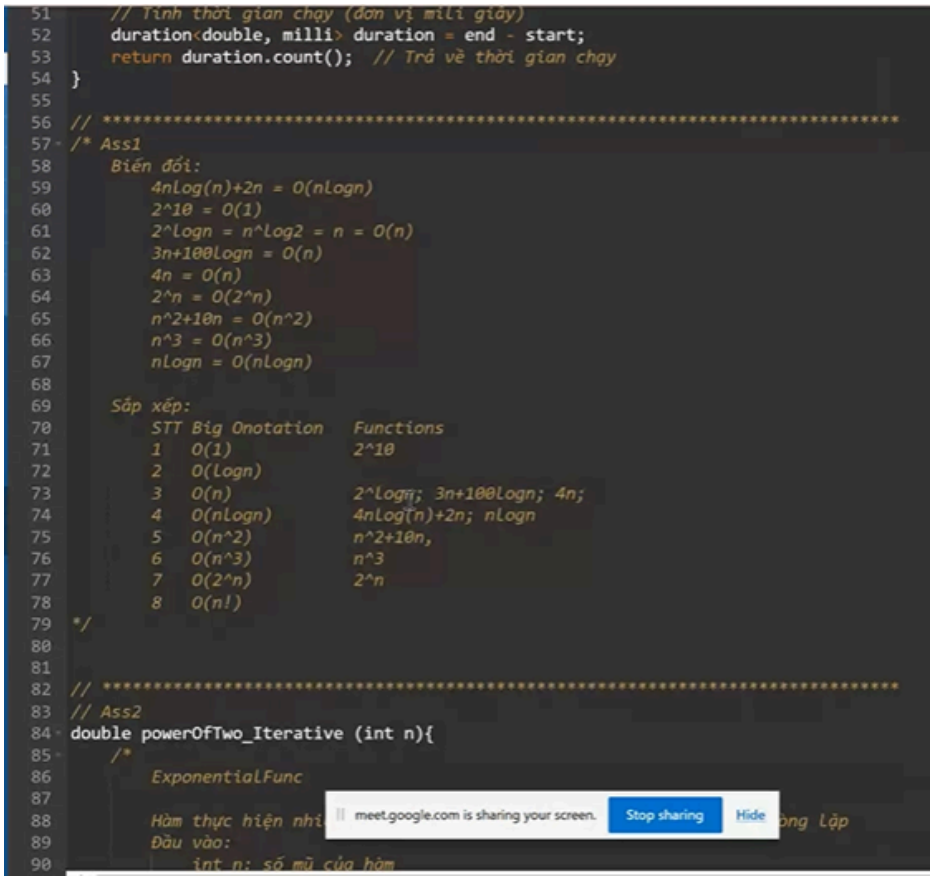
Homework

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

   2^10, 2^logn, 3n + 100.logn, 4n, n.logn, 4n.logn + 2n, n^2 + 10n, n^3, 2^n



```
    // Tính thời gian chạy (đơn vị mili giây)
    duration<double, milli> duration = end - start;
    return duration.count();  // Trả về thời gian chạy
}

// ********************************************************************************
/* Ass1
    Biến đổi:
        4nlog(n)+2n = O(nlogn)
        2^10 = O(1)
        2^logn = n^log2 = n = O(n)
        3n+100logn = O(n)
        4n = O(n)
        2^n = O(2^n)
        n^2+10n = O(n^2)
        n^3 = O(n^3)
        nlogn = O(nlogn)

    Sắp xếp:
        STT Big Onotation   Functions
        1   O(1)           2^10
        2   O(logn)
        3   O(n)           2^logn; 3n+100logn; 4n;
        4   O(nlogn)       4nlog(n)+2n; nlogn
        5   O(n^2)         n^2+10n,
        6   O(n^3)         n^3
        7   O(2^n)         2^n
        8   O(n!)
*/


// ********************************************************************************
// Ass2
double powerOfTwo_Iterative (int n){
    /*
        ExponentialFunc

        Hàm thực hiện nhi                                    ông lặp
        Đầu vào:
            int n: số mũ của hàm
```


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.

```
long long pow_2_n (int n):
    long long res = 1;
    for int i = 0 -> n-1:
        res *= 2;
    return res;
```

T12 = 2 = O(1)

T3 = n - 1 = O(n)

T4 = 1 = O(1)

T5 = 1 = O(1)

T12345 = T12 + T3 * T4 + T5 = O(n)

```cpp
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

- T1 = O(1)
- T(b == 0) = O(1)
- T4 = 1 + T(binPow)
- T5678 = O(1)
- T(binpow) = O(log b)

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

```
1 void deleteFirstEleInArr (int* A, int& n)
2   for (int i = 0; i < n - 1; i ++) {
3       A[i] = A[i + 1];
4   }
5   n --;
6}

class Queue {
   int* values;
   int countValue;
public:

   // Enqueue & dequeue
1   void enqueue(int value) {
2       values[countValue] = value;
3       countValue ++;
4   }
5   void dequeue() {
6       deleteFirstEleInArr(values, countV
7   }
};
```

- deleteFirstEleInArr
  - T1 = 1 = O(1)
  - T2 = n - 2 = O(n)
  - T3 = 3 = O(1)
  - T5 = 1 = O(1)
  - T(deleteFirstEleInArr) = T1 + T2 * T3 + T5 = O(n)
- enqueue
  - T1 = 1 = O(1)
  - T2 = 2 = O(1)
  - T3 = 1 = O(1)
  - T(enqueue) = T1 + T2 + T3 = O(1)
- dequeue
  - T6 = T(deleteFirstEleInArr) = O(n)

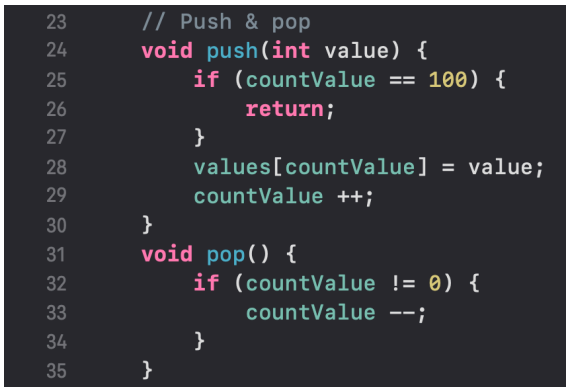
4. Your task is to write operations of queue data structure in pseudo-codes using a linked list, then evaluate the complexities of the operations.

```
33      // Enqueue & dequeue
34      void enqueue(int value) {
35          if (head == NULL) {
36              head = new Node (value, NULL);
37              return;
38          }
39          Node* p = head;
40          while (p->next != NULL) {
41              p = p->next;
42          }
43          p->next = new Node (value, NULL);
44      }
45      void dequeue() {
46          if (head == NULL) {
47              return;
48          }
49          Node* tmp = head;
50          head = head->next;
51          delete tmp;
52      }
53  };
```

- enqueue
  - T34 = 1 = O(1)
  - T35 = 1 = O(1)
  - head == NULL
    - T36 = 3 = O(1)
    - T(head == NULL) = O(1)
  - head ≠ NULL
    - T39 = 1 = O(1)
    - T40 = n - 1 = O(n) (coi n là số lượng phần tử trong linked list)
    - T41 = 1
    - T43 = 3 = O(1)
    - T(head ≠ NULL) = O(n)
  - T(enqueue) = max(T(head == NULL), T(head ≠ NULL)) = O(n)
- dequeue
  - T46 = 1 = O(1)
  - head == NULL
    - T(head == NULL) = 0 = O(1)
  - head ≠ NULL
    - T(49) = 1 = O(1)
    - T(50) = 1 = O(1)
    - T(51) = 1 = O(1)
    - T(head ≠ NULL) = O(1)

- T(dequeue) = max(T(head == NULL), T(head ≠ NULL)) = O(1)

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



```
23      // Push & pop
24      void push(int value) {
25          if (countValue == 100) {
26              return;
27          }
28          values[countValue] = value;
29          countValue ++;
30      }
31      void pop() {
32          if (countValue != 0) {
33              countValue --;
34          }
35      }
```

- push
  - T24 = 1 = O(1)
  - T25 = 1 = O(1)
  - T(countValue == 100) = 1 = O(1)
  - countValue ≠ 100
    - T28 = 2 = O(1)
    - T29 = 1 = O(1)
    - T(countValue ≠ 100) = O(1)
  - T(push) = max(T(countValue ==100); T(countValue ≠ 100)) = O(1)
- pop
  - T32 = 1 = O(1)
  - T(countValue == 0) = 0 = O(1)
  - countValue ≠ 0
    - T33 = 1 = O(1)
    - T(countValue ≠ 0) = 1 = O(1)
  - T(pop) = max(T(countValue == 0), T(countValue ≠ 0)) =

O(1)

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.

```
38      // push & pop
39      void push(int value) {
40          if (head == NULL) {
41              head = new Node(value, NULL);
42              return;
43          }
44          Node* p = head;
45          while (p->next != NULL) {
46              p = p->next;
47          }
48          p->next = new Node(value);
49      }
50      void pop() {
51          if (head == NULL) {
52              return;
53          } else if (head->next == NULL) {
54              delete head->next;
55              head = NULL;
56          }
57          Node* p = head;
58          while (p->next->next != NULL) {
59              p = p->next;
60          }
61          delete p->next;
62          p->next = NULL;
63      }
64  };
```

- push
  - T39 = 1 = O(1)
  - T40 = 1 = O(1)
  - head == NULL
    - T41 = 3 = O(1)
    - T(head == NULL) = O(1)
  - head ≠ NULL
    - T44 = 1 = O(1)
    - T45.46 = n - 1 = O(n) (n ~ là số lượng phần tử trong stack)
    - T48 = 3 = O(1)
    - T(head ≠ NULL) = O(n)
  - T(push) = max(T(head == NULL), T(head ≠ NULL)) = O(n)
- pop
  - T51 = 1 = O(1)
  - T(head == NULL) = 1 = O(1)
  - head ≠ NULL
    - T53 = 1 = O(1)
    - head→next == NULL
      - T54 = 1 = O(1)
      - T55 = 1 = O(1)

        - T(head→next) = O(1)
    - head→next ≠ NULL
        - T57 = 1 = O(1)
        - T58.59 = n - 2 = O(n)
        - T61 = 1 = O(1)
        - T62 = 1 = O(1)
        - T(head→next ≠ NULL) = O(n)
    - T (head ≠ NULL) = max(T(head→next == NULL), T(head→next ≠ NULL)) = O(n)
- T(pop) = max(T(head ≠ NULL), T(head→next ≠ NULL)) = O(n)