Natthapee Sriarunluck, 6480266

Task 1

1) $Let\ f(n)\ =\ n\ and\ g(n)\ =\ n\ log(n)$

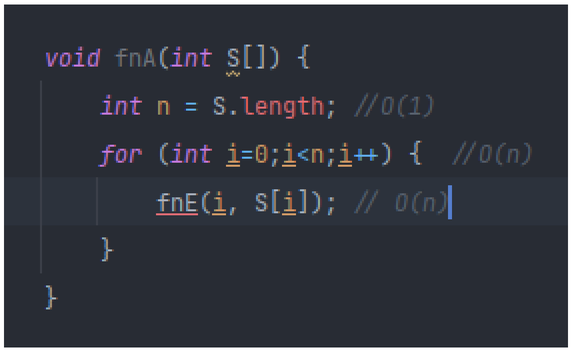
$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{n}{n\ log\ (n)} = \lim_{n\to\infty}\frac{1}{\infty} = 0 < \infty$$

$Therefore,\ f(n)\ is\ O(g(n))\ or\ O(nlog(n))$

2) $\lim_{n\to\infty}\frac{d(n)}{f(n)} < \infty,\ \lim_{n\to\infty}\frac{e(n)}{g(n)} < \infty$

$A\ finite\ constant\ multiplied\ by\ a\ finite\ constant\ cannot\ reach\ an\ infininity\ constant.$
$Therefore,\ the\ preposition\ is\ true.$

3)



```
void fnA(int S[]) {
    int n = S.length; //O(1)
    for (int i=0;i<n;i++) {  //O(n)
        fnE(i, S[i]); // O(n)
    }
}
```

for loop in fnA runs to the length of the array S[ ]. The runtime is O(n). fnE runs in $1000\ \times\ i$ steps where i is also increased to the length of the array S[ ]. Therefore the runtime is $O(n)\ \times\ O(n)\ =\ O(n^2)$

4) $\lim_{n\to\infty}\frac{16n^2+11n^4+0.1n^5}{n^4} = \lim_{n\to\infty}\frac{n^5}{n^4} = n = \infty$

$O(n)\ <\ \infty$. Since $h(n)$ is infinity and not less than, it is not $O(n^4)$

Task 2

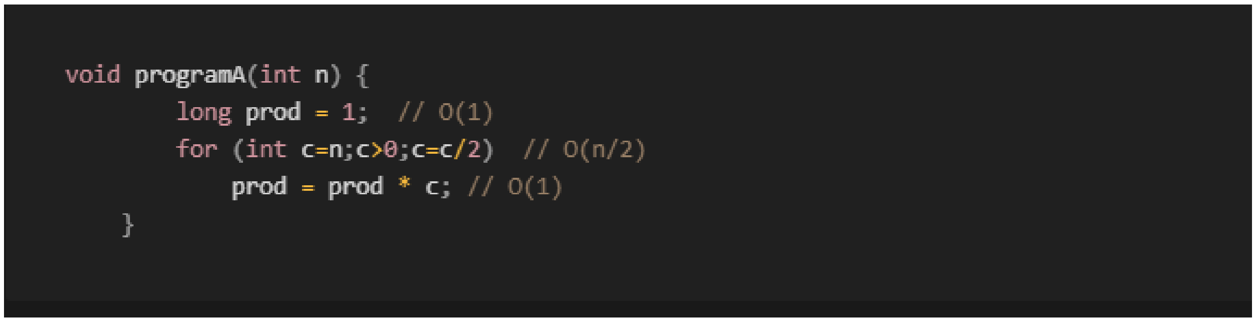
1. Label the wines to 1 to n.
2. Convert each number of wines to binary numbers

3. The largest number n to binary's length is k
4. Label the testers to represent each digit of the binary numbers of length k.
5. Feed the testers according to the binary representations of the wine in which the testers that represents the number "1" are fed.
6. Wait 31 days after feeding all the testers. Read the binary representation of the testers in which the ones that laugh are "1" and normal is "0". That binary representation to decimal is the bottle of wine that was poisoned.

Since the numbers between 1-n can be represented by using $log_2(n + 1)$ bits (binary), and we decide on the number of testers by using binary numbers, the number of testers is $log_2(n + 1)$ Therefore, the runtime is $O(logn)$ since the +1 becomes irrelevant to the behavior as n gets larger (dropping constants).

Task 3

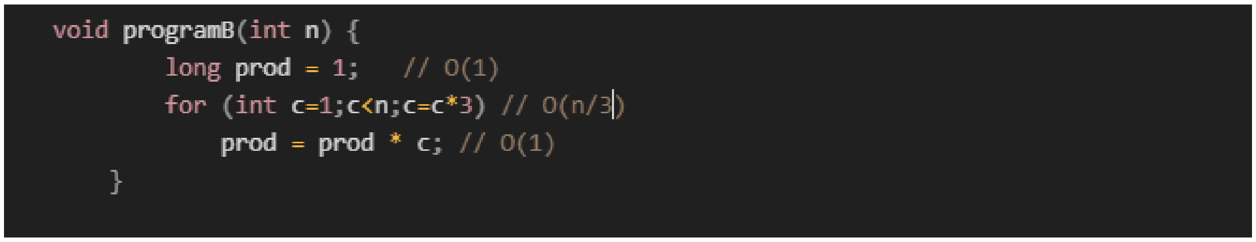
Program A:



```
void programA(int n) {
        long prod = 1;  // O(1)
        for (int c=n;c>0;c=c/2)  // O(n/2)
            prod = prod * c; // O(1)
    }
```

$O(\frac{n}{2}) + O(1) + O(1) = \theta log(n)$

Program B:



```
void programB(int n) {
        long prod = 1;   // O(1)
        for (int c=1;c<n;c=c*3) // O(n/3)
            prod = prod * c; // O(1)
    }
```

$O(\frac{n}{3}) + O(1) + O(1) = \theta log(n)$

Natthapee Sriarunluck, 6480266

Task 4

```
def hsum(X):  # assume len(X) is a power of two
   while len(X) > 1:
      (1) allocate Y as an array of length len(X)/2
      (2) fill in Y so that Y[i] = X[2i] + X[2i+1] for $i$ = 0, 1, …, len($X$)/2 − 1
      (3) X = Y
   return X[0]
```

1) Step 1: $k_1 \times z$

   Step 2: $2k_2 \times \frac{z}{2}$

   Step 3: 1

   Total: $(k_1 + k_2)z + 1$

2)

Step 1	Step 2	Step 3
64	32	1
32	16	1
16	8	1
8	4	1
4	2	1
2	1	1
1	1	1

We can write up sums

$$n(k_1 + k_2) \sum_{i=0}^{log_2(n)} 2^{-i} + \sum_{i=1}^{log_2(n)} 1$$

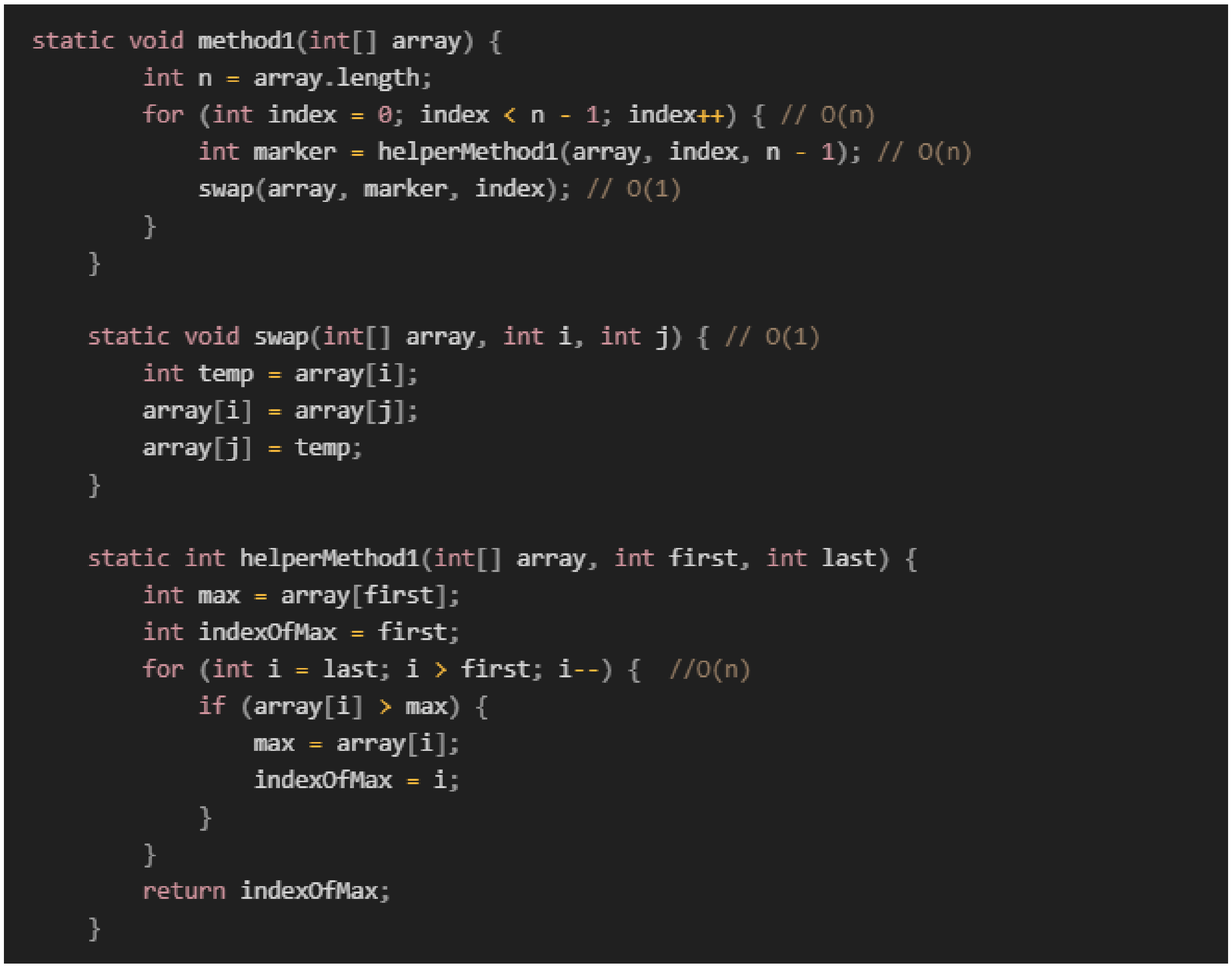
Natthapee Sriarunluck, 6480266

$(k_1 + k_2) \times n \times 2^{-log_2(n)+1} + 1 - 1 + log_2(n)$

$= \frac{1}{2}(k_1 + k_2) + log_2(n)$

The total runtime is = $\frac{1}{2}(k_1 + k_2) + log_2(n)$

Task 5

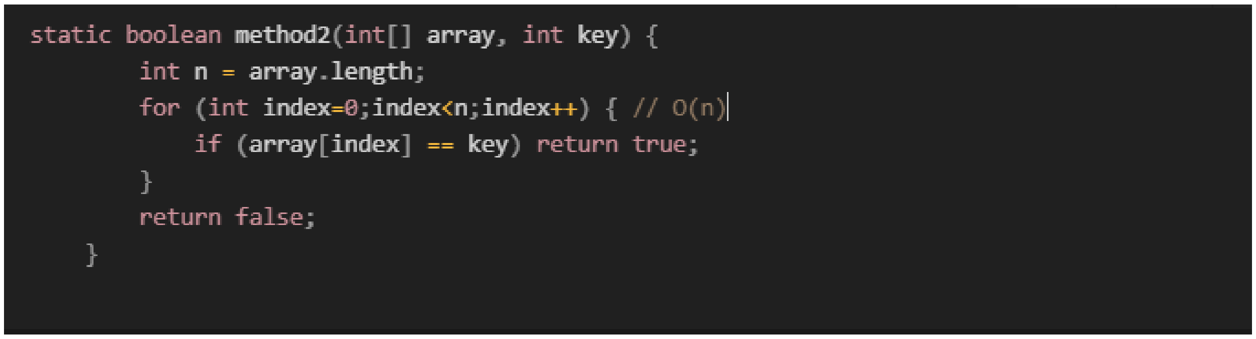


```
static void method1(int[] array) {
        int n = array.length;
        for (int index = 0; index < n - 1; index++) { // O(n)
            int marker = helperMethod1(array, index, n - 1); // O(n)
            swap(array, marker, index); // O(1)
        }
    }

    static void swap(int[] array, int i, int j) { // O(1)
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    static int helperMethod1(int[] array, int first, int last) {
        int max = array[first];
        int indexOfMax = first;
        for (int i = last; i > first; i--) {  //O(n)
            if (array[i] > max) {
                max = array[i];
                indexOfMax = i;
            }
        }
        return indexOfMax;
    }
```

1) Worse Case: $O(n) \times O(n) = O(n^2)$

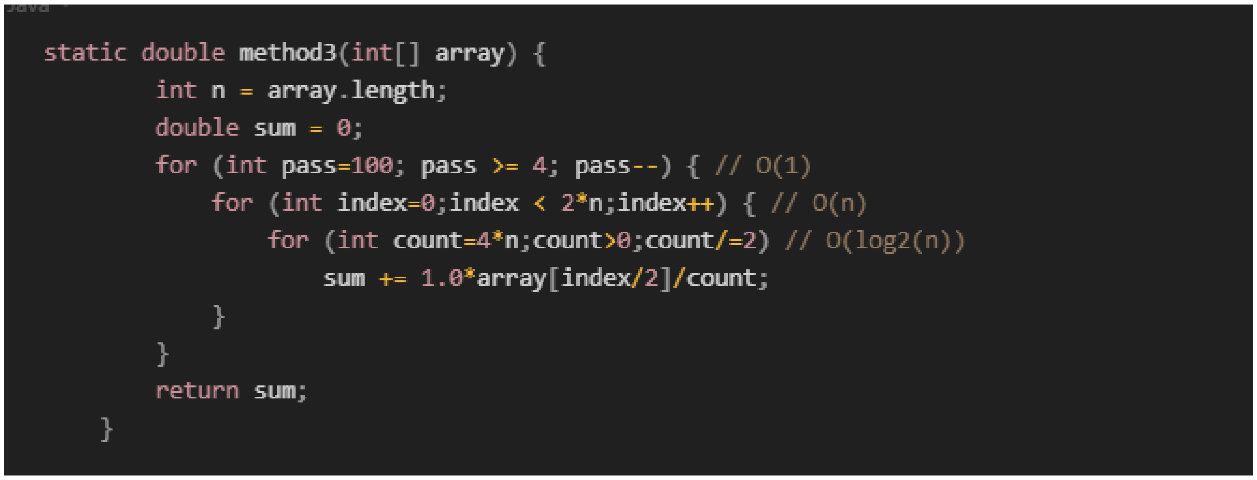
   Best Case: $O(n) \times O(n) = O(n^2)$

Natthapee Sriarunluck, 6480266



```java
static boolean method2(int[] array, int key) {
        int n = array.length;
        for (int index=0;index<n;index++) { // O(n)
            if (array[index] == key) return true;
        }
        return false;
    }
```

2) Worse Case: $O(n)$
   Best Case: $O(1)$

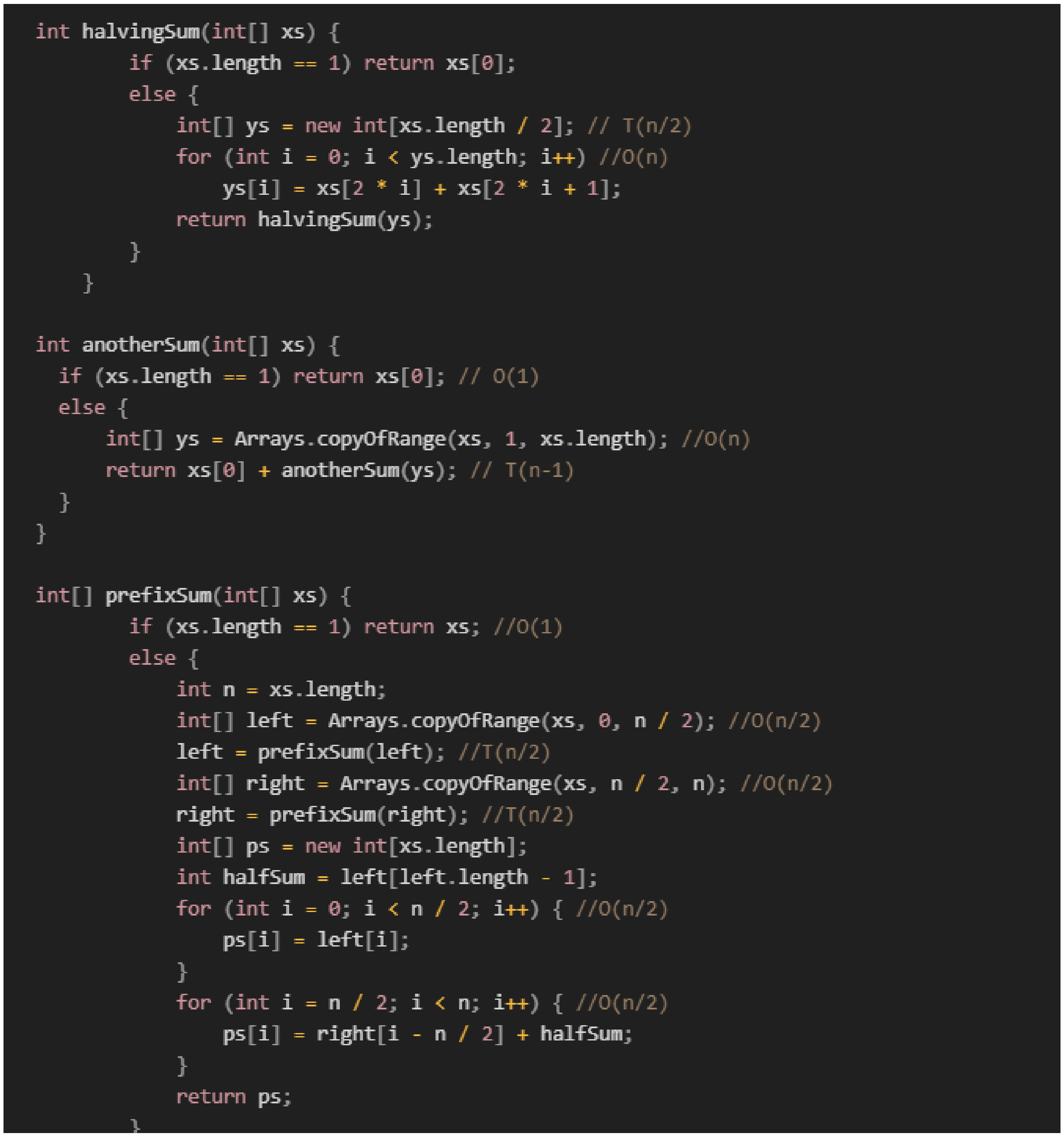


```java
static double method3(int[] array) {
        int n = array.length;
        double sum = 0;
        for (int pass=100; pass >= 4; pass--) { // O(1)
            for (int index=0;index < 2*n;index++) { // O(n)
                for (int count=4*n;count>0;count/=2) // O(log2(n))
                    sum += 1.0*array[index/2]/count;
            }
        }
        return sum;
    }
```

3) Worse Case: $O(n\,log(n))$
   Best Case: $O(n)$

Natthapee Sriarunluck, 6480266

Task 6



```
int halvingSum(int[] xs) {
        if (xs.length == 1) return xs[0];
        else {
            int[] ys = new int[xs.length / 2]; // T(n/2)
            for (int i = 0; i < ys.length; i++) //O(n)
                ys[i] = xs[2 * i] + xs[2 * i + 1];
            return halvingSum(ys);
        }
    }

int anotherSum(int[] xs) {
  if (xs.length == 1) return xs[0]; // O(1)
  else {
      int[] ys = Arrays.copyOfRange(xs, 1, xs.length); //O(n)
      return xs[0] + anotherSum(ys); // T(n-1)
  }
}

int[] prefixSum(int[] xs) {
        if (xs.length == 1) return xs; //O(1)
        else {
            int n = xs.length;
            int[] left = Arrays.copyOfRange(xs, 0, n / 2); //O(n/2)
            left = prefixSum(left); //T(n/2)
            int[] right = Arrays.copyOfRange(xs, n / 2, n); //O(n/2)
            right = prefixSum(right); //T(n/2)
            int[] ps = new int[xs.length];
            int halfSum = left[left.length - 1];
            for (int i = 0; i < n / 2; i++) { //O(n/2)
                ps[i] = left[i];
            }
            for (int i = n / 2; i < n; i++) { //O(n/2)
                ps[i] = right[i - n / 2] + halfSum;
            }
            return ps;
```

6.1) Measure the size of the problem by making n = length of array “xs”
6.2 and 6.3)

halvingSum: $T_n = T(\frac{n}{2}) + O(n)\ solves\ to\ O(n)$

anotherSum: $T_n = T(n - 1) + O(n)\ solves\ to\ O(n^2)$

prefixSum: $2T(\frac{n}{2}) + O(n)\ solves\ to\ O(n\ log(n))$

Natthapee Sriarunluck, 6480266

Task 7

1) Since g(0) = 0 and f(0) = 0, substitute the values and solve for c.
   $g(0) = a \times f(o) + b \times 0 + c$
   $0 = a \times 0 + 0 + c$
   $0 = 0 + c$
   $c = 0$
2) $af(n) + bn + c = 2[af(n - 1) + b(n - 1) + c] + n$
   $af(n) + bn + c = 2af(n - 1) + 2b(n - 1) + 2c + n$
   $af(n) - 2af(n - 1) + bn - 2b(n - 1) - n = 0$
   $a(1) - bn + 2b - n = 0$
   $(- b - 1)n + (a + 2b) = 0$

   $(- b - 1)n = 0$
   $b = - 1$
   Substitue b into $(a + 2b) = 0$
   $a - 2 = 0$
   $a = 2$

3) $g(n) = 2g(n - 1) + n$
   $g(0) = 0$
   $g(1) = 2 \times g(0) + 1 = 2 \times 0 + 1 = 1$
   $g(2) = 2 \times g(1) + 2 = 2 \times 1 + 2 = 4$
   $g(3) = 2 \times g(2) + 3 = 2 \times 4 + 3 = 11$
   $g(4) = 2 \times g(3) + 4 = 2 \times 11 + 4 = 26$
   $g(n) = \sum_{i=0}^{n} 2^{i}(n - i) = 2^{n+1} - n - 2$

Natthapee Sriarunluck, 6480266

4)

Theorem: $g(n) = 2g(n-1) + n$ is equals to $G(n) = 2^{n+1} - n - 2$

Base Case:
LHS: $g(0) = 0 \; by \; definition$
RHS: $G(0) = 2^{0+1} - 0 - 2 = 2 - 0 - 0 = 0$
$LHS = RHS \; so \; Base \; Case \; is \; true$

Inductive Step:
Assume: $g(n) = G(n) \; \forall n <= k$
WTS: $g(n+1) = G(n+1)$

Proof:
$g(n+1) = 2g(n) + n + 1$
$G(n+1) = 2^{n+2} - (n+1) - 2 = 2^{n+2} - n - 3$
$2g(n) + n + 1$ = $2^{n+2} - n - 3$
By using inductive hypothesis
LHS = $2(2^{n+1} - n - 2) + n + 1 = 2^{n+2} - 2n - 4 + n + 1$
$= 2^{n+2} - n - 3$ = RHS

By mathematical induction, g(n) is indeed equals to G(n)