1️⃣ Function: `alpha(int m)`

📌 **Question:**
Write a function `alpha(int m)` that prints all numbers from `1` to `m` in a single line, separated by spaces.
✅ **Example Input & Output:**

makefile
CopyEdit
`Input: 5`
`Output: 1 2 3 4 5`

2️⃣ Function: `beta(int x)`

📌 **Question:**
Write a function `beta(int x)` that returns a string where the first row contains `x` copies of `'a'`, the second row has `x-1` copies of `'b'`, and so on, until a single character remains.
✅ **Example Input & Output:**

vbnet
CopyEdit
`Input: 3`
`Output: "aaabbc"`

💡 **Explanation:**

- `'a'` is repeated `3` times
- `'b'` is repeated `2` times
- `'c'` is repeated `1` time
- Final string: `"aaabbc"`

3️⃣ Function: `gamma(int x)`

📌 **Question:**
Write a function `gamma(int x)` that returns an array of 3 values:

1. The square of `x`

2. The sum of digits of `x`
3. `6` if `x` is **prime**, otherwise `9`

✅ **Example Input & Output:**

makefile
CopyEdit
```
Input: 13
Output: [169, 4, 6]
```

💡 **Explanation:**

- 132=16913^2 = 169132=169
- Sum of digits: 1+3=41 + 3 = 41+3=4
- `13` is **prime**, so output is `6`

4️⃣ Function: `delta(int[] arr)` (Simulating Call by Reference)

📌 **Question:**
Write a function `delta(int[] arr)` that takes an array of two integers.

- It replaces `arr[0]` with `arr[1] + 7`
- It replaces `arr[1]` with `arr[0] + 5` (original value of `arr[0]`)
- Print the updated values of both numbers.

✅ **Example Input & Output:**

makefile
CopyEdit
```
Input: 3 8
Output: 15 8
```

💡 **Explanation:**

- `arr[0] = arr[1] + 7 = 8 + 7 = 15`
- `arr[1] = original arr[0] + 5 = 3 + 5 = 8`

Java code :

```
import java.util.*;
import java.lang.*;
import java.io.*;

class Codechef
{
    public static void alpha(int m){
        for(int i =1; i<=m; i++){
            System.out.print(i+" ");
        }
    }

    public static String beta(int x){
        String total_string = "";
        char m = 'a';
        while(x != 0){
            for(int i = 0; i < x; i++){
                total_string += m;
            }
            m = (char)(m + 1);
            x--;
        }
        return total_string;
    }

    public static int[] gamma(int x){
        int[] charan = new int[3];
        charan[0] = (int)(Math.pow(x, 2));

        int sum = 0;
        int t = x;
        while(t != 0){
            sum = sum + (t % 10);
            t = t / 10;
        }
        charan[1] = sum;

        int flag = 0;
        for(int i = 2; i <= x / 2; i++){
            if((x % i) == 0){
                flag = 1;
                break;
            }
```

```
        }

        if(flag == 0) charan[2] = 6;
        else charan[2] = 9;

        return charan;
    }

    // Simulating Call by Reference using an array
    public static void delta(int[] arr) {
        int temp = arr[0];
        arr[0] = arr[1] + 7;  // 15
        arr[1] = temp + 5;  // 8
    }

    public static void main (String[] args) throws java.lang.Exception
    {
        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        alpha(x);
        System.out.println();

        String m = beta(x);
        System.out.println(m);

        int[] t = gamma(x);
        System.out.println(Arrays.toString(t));

        for(int i = 0; i < 3; i++){
            System.out.print(t[i] + " ");
        }
        System.out.println();

        int a = sc.nextInt();
        int b = sc.nextInt();

        // Creating an array to pass by reference
        int[] arr = {a, b};
        delta(arr);

        // Printing updated values of a and b
        System.out.println(arr[0] + " " + arr[1]);
    }
}
```

CPP CODE :

```
#include <bits/stdc++.h>
using namespace std;

void alpha(int m) {
    for (int i = 1; i <= m; i++) {
        cout << i << " ";
    }
    cout << endl;
}

string beta(int x) {
    string total_string = "";
    char m = 'a';
    while (x != 0) {
        for (int i = 0; i < x; i++) {
            total_string += m;
        }
        m = m + 1;
        x--;
    }
    return total_string;
}

vector<int> gamma(int x) {
    vector<int> charan(3);
    charan[0] = pow(x, 2);

    int sum = 0, t = x;
    while (t != 0) {
        sum += (t % 10);
        t /= 10;
    }
    charan[1] = sum;

    int flag = 0;
    for (int i = 2; i <= x / 2; i++) {
        if (x % i == 0) {
            flag = 1;
            break;
        }
```

```
    }
    charan[2] = (flag == 0) ? 6 : 9;
    return charan;
}

// Call by Reference in C++
void delta(int &a, int &b) {
    int temp = a;
    a = b + 7;  // 15
    b = temp + 5;  // 8
}

int main() {
    int x;
    cin >> x;
    alpha(x);

    string m = beta(x);
    cout << m << endl;

    vector<int> t = gamma(x);
    for (int i = 0; i < 3; i++) {
        cout << t[i] << " ";
    }
    cout << endl;

    int a, b;
    cin >> a >> b;
    delta(a, b);

    cout << a << " " << b << endl;
    return 0;
}
```

Call by REF :

Modified Code using Pointers

If you want to pass addresses, you need to **use pointers (`int*`) instead of references (`int&`)**:

cpp

CopyEdit

```
#include <bits/stdc++.h>
using namespace std;

void delta(int *a, int *b) {
    int temp = *a;
    *a = *b + 7;  // Modifying the value at address of a
    *b = temp + 5;  // Modifying the value at address of b
}

int main() {
    int a, b;
    cin >> a >> b;

    // Calling delta function with addresses
    delta(&a, &b);

    cout << a << " " << b << endl;
    return 0;
}
```

Key Changes

✅ Used **pointers (`int *`)** instead of references
✅ Used **dereferencing (`*a, *b`)** to modify values at the passed memory locations
✅ Used **`&a, &b` in the function call** to send the address of variables

1️⃣ Function: `alpha(int m)`

📌 Observations:

✅ **Simple Iteration** – The function prints numbers from `1` to `m` in a single line.
✅ **Time Complexity**: **O(m)** – It iterates from `1` to `m`, printing each number.
✅ **Edge Cases:**

- `m = 1` → Only prints `1`

- `m = 0` → Prints nothing
- **Negative values are not handled** → Function should check if `m < 1`

🛠 Possible Enhancements:

- Use `StringBuilder` instead of `System.out.print()` for better efficiency in large inputs.
- Handle negative `m` by printing an error message.

2️⃣ Function: `beta(int x)`

📌 Observations:

✅ **Pattern Generation** – The function builds a decreasing sequence of characters (`a`, `b`, `c`, ...).
✅ **String Concatenation Issue**:

- Using `total_string += m;` inside a loop creates multiple string objects (inefficient).
- Instead, use `StringBuilder` to improve performance.
  ✅ **Time Complexity**: **O(x²)** –
- First iteration runs `x` times,
- Second iteration runs `x-1` times, and so on.
- Total operations = `x + (x-1) + (x-2) + ... + 1 = O(x²)`.
  ✅ **Edge Cases:**
- `x = 1` → Only `a` is printed
- `x = 0` → Returns an empty string

🛠 Possible Enhancements:

- **Optimize string concatenation** with `StringBuilder` (`O(x²) → O(x)`).
- **Add validation for `x`** (avoid negative values).

3️⃣ Function: `gamma(int x)`

📌 Observations:

✅ **Three Key Computations:**

- **Square Calculation:** `Math.pow(x,2)`, stored as an integer.
- **Digit Sum Calculation:** Extracting and summing digits using a loop.
- **Prime Check:** Determines if `x` is prime using `O(√x)` method.

  ✅ **Prime Check Improvement:**
- Checking divisibility up to `x/2` is **inefficient**.
- Optimal approach: **Check divisibility up to √x** (reduces complexity from `O(x)` to `O(√x)`).

  ✅ **Time Complexity: O(√x) for prime check**, **O(log x) for sum of digits**, **O(1) for square**.

  ✅ **Edge Cases:**
- `x = 1` → Special case (not prime)
- `x = 0` → Might cause an issue in prime checking

🛠 Possible Enhancements:

- **Optimize Prime Check:** Replace `for (int i = 2; i <= x / 2; i++)` with `for (int i = 2; i * i <= x; i++)`.
- **Handle edge cases** like `x = 0` and `x = 1`.

4️⃣ Function: `delta(int[] arr)`

📌 Observations:

✅ Pass-by-Reference Simulation

- Java **does not support true pass-by-reference**, so an **array** is used as a workaround.

  ✅ **Swaps Values with Computation:**
- `arr[0] = arr[1] + 7`
- `arr[1] = original arr[0] + 5`

  ✅ **Side Effects:**
- The function **modifies the original array**, affecting values in `main()`.

  ✅ **Time Complexity: O(1)** – Only a few arithmetic operations.

  ✅ **Edge Cases:**
- `arr[0] = 0`, `arr[1] = 0` → Should not cause errors.
- Large values of `a` and `b` should not cause overflow.

🛠 Possible Enhancements:

- **Use separate variables** to avoid modifying input directly.

- **Explain the logic clearly** (since the function updates values in an unusual way).

🔥 Final Key Takeaways

1. **Optimize String Operations** → Use `StringBuilder` for efficiency in `beta()`.
2. **Efficient Prime Check** → Use `O(√x)` instead of `O(x/2)` in `gamma()`.
3. **Handle Edge Cases** → Check negative values, zero inputs, and special cases like `1` in `gamma()`.
4. **Understand Java's Memory Model** → Java uses **pass-by-value** for primitives and **pass-by-reference-like behavior** for objects/arrays.