ANSWER - 1 :

1st iteration:
1st step: 7 2 13 2 11 4 -> 2 7 13 2 11 4
2nd step: 2 7 13 2 11 4 -> 2 7 13 2 11 4
3rd step: 2 7 13 2 11 4 -> 2 7 2 13 11 4
4th step: 2 7 2 13 11 4 -> 2 7 2 11 13 4
5th step: 2 7 2 11 13 4 -> 2 7 2 11 4 13

2st iteration:
1st step: 2 7 2 11 4 13 -> 2 7 2 11 4 13
2nd step: 2 7 2 11 4 13 -> 2 2 7 11 4 13
3rd step: 2 2 7 11 4 13 -> 2 2 7 11 4 13
4th step: 2 2 7 11 4 13 -> 2 2 7 4 11 13

3rd iteration:
1st step: 2 2 7 4 11 13 -> 2 2 7 4 11 13
2nd step: 2 2 7 4 11 13 -> 2 2 7 4 11 13
3rd step: 2 2 7 4 11 13 -> 2 2 4 7 11 13

The final sorted array will be 2, 2, 4, 7, 11, 13

ANSWER - 2 :

1. **Memory allocation:** In C++, arrays are statically allocated, meaning that the size of the array must be specified at the time of declaration and cannot be changed later on. On the other hand, vectors are dynamically allocated, which means that their size can be changed during runtime.

2. **Operations:** Arrays in C++ do not have many built-in operations such as push_back(), pop_back(), insert(), and erase() which are available in vectors, These operations allow you to add and remove elements from the end or specific position of the vector.

ANSWER - 3 :

The time complexity of this code segment is O(n^2).

The outer for loop iterates n times and the inner for loop also iterates n times, so the total number of iterations is n * n = n^2.

The builtin_popcount(i) function is used to count the number of set bits in the binary representation of the variable i. The time complexity of this function is considered to be O(1) or constant time, as it takes the same amount of time to count the set bits regardless of the value of i. Therefore, it does not affect the overall time complexity of the code.

The statement inside the outer loop is executed only when the number of set bits in the binary representation of i is equal to 2, which is a constant number and it is not dependent on the value of i.

Therefore, the overall time complexity of the code is O(n^2) which is determined by the two nested loops.

ANSWER - 4 :

There are a few flaws in this code:

1. The first flaw is in the for loop that is used to count the number of distinct elements. The loop iterates from i=0 to i=n. However, the range of the vector a is from 0 to n-1, so the loop should iterate from i=0 to i=n-1.

2. The second flaw is in the if statement inside the for loop. It is checking if a[i] is not equal to a[i-1], but this will not work for the first element of the array (i=0) because the a[-1] is not exists and it will throw an error.

   To fix this, we can initialize ans = 1 before the loop, and change the if condition to check a[i] != a[i+1]

ANSWER - 5 :

The time complexity of this code segment is O(n^2) and the space complexity is O(n).

The outer for loop iterates n times, and for each iteration, the inner for loop iterates n/i times. The total number of iterations of the inner for loop is the summation of n/i for i = 1 to n. This summation can be simplified as n*(1 + 1/2 + 1/3 + ... + 1/n) which is approximately equal to n*ln(n) and in the worst case scenario it will be n.

Therefore, the overall time complexity of this code is O(n^2) which is determined by the two nested loops.

The code uses a vector of vectors, d, to store the divisors of the numbers from 1 to n. The space complexity of this data structure is O(n) because it requires space to store n vectors, each with a maximum size of n/i.

ANSWER - 6 :

Name	Accessibility from own class	Accessibility from derived class	Accessibility from world
Public	YES	YES	YES
Private	YES	NO	NO
Protected	YES	YES	NO

ANSWER - 7 :

'new' and 'delete' are keywords used for dynamic memory allocation and deallocation.

'new' is used to dynamically allocate memory for an object or an array. It returns a pointer to the memory location that has been allocated.

'delete' is used to deallocate the memory that has been previously allocated by 'new'. It releases the memory that was allocated by 'new' and returns it to the system for reuse.

It is important to note that if memory is allocated dynamically using 'new' keyword, it should also be deallocated using 'delete' keyword to avoid memory leaks. Also, It's not allowed to use 'delete' on memory that was not allocated dynamically.

ANSWER - 8 :

I agree with Bob that the algorithm will take a long time to finish in the worst case, assuming n is at its maximum of 10^6. The O(n^3) notation describes the worst-case time complexity of the algorithm, which means that the running time of the algorithm will increase rapidly as the input size (n) increases.

In this case, we can estimate that the number of operations is around (10^6)^3 = 10^18 and the instructions per operation is around 10^9.

So, time = (10^18) / (10^9) = 10^9 seconds.

This is equivalent to approximately 31.7 years.

ANSWER - 9 :

1. **Time complexity:** Binary search has a time complexity of O(log n) while linear search has a time complexity of O(n). This means that, as the size of the input data increases, binary search will take significantly less time to complete than linear search.

2. **Data structure:** Binary search can only be applied to sorted data, whereas linear search can be applied to both sorted and unsorted data.

ANSWER - 10 :

The function has a memory leak, as the memory allocated by the "new" operator is not properly deallocated. The "new" operator dynamically allocates memory on the heap and the memory remains allocated until it is explicitly deallocated by the "delete" operator. Since the function returns without deallocating the memory, the program will eventually run out of memory

A way to fix this :

```
void func(){
        int* p = new int;
        delete p;
        return;
}
```

This way, the memory will be automatically deallocated when the function exits and no memory leak will occur.