Consider the problem of searching an elemet x in an array 'arr[]' of size n. The problem can be solved in O(Logn) time if.

1) Array is sorted
2) 2) Array is sorted and rotated by k. k is given to you and k <= n
3) 3) Array is sorted and rotated by k. k is NOT given to you and k <= n
4) 4) Array is not sorted

1 Only

1 & 2 only

1, 2 and 3 only

1, 2, 3 and 4

What does the following function do?

```
int fun(int x, int y)
{
    if (y == 0)   return 0;
    return (x + fun(x, y-1));
}
```

A $x + y$

B $x + x*y$

C $x*y$

D x^y

What does the following function do?

```
int fun(int x, int y)
{
    if (y == 0)   return 0;
    return (x + fun(x, y-1));
}

int fun2(int a, int b)
{
    if (b == 0) return 1;
    return fun(a, fun2(a, b-1));
}
```

- You have to sort 1 GB of data with only 100 MB of available main memory. Which sorting technique will be most appropriate?

- What is the worst case time complexity of insertion sort where position of the data to be inserted is calculated using binary search?

Consider a sorted array of n numbers. What would be the time complexity of the best known algorithm to find a pair 'a' and 'b' such that |a-b| = k , k being a positive integer.

O(n)

O(n log n)

O(n ^ 2)

O(log n)