Operating Systems and Algorithms

Asssignment-1

Name:Ajay Ray Roll:2021102032

Q.1. Explanation: Given q and n two integers and according to the question our final goal is to find

$$f(n) = k * f(n-1) = k^t, t = n-1$$

algorithm: I just divided the power in two half. And finally returning the final ansewer is product of both halves and modulo multiplication is done accordingly;

```
    ll left=find(k,n/2);
    ll right=l;
    if(n%2)right=(right%M* k%M)%M;
    return(left%M * right%M)%M;
```

Time complexcity: O(log N)

```
T(N)=T(N/2)+O(1) Here a=1,b=2 and c=0;
log(a) base b=c  so by master's theorem time complexcity is T(n)= log n(beacuse c is 0)
```

Space complexcity:O(log N)

In the worst case, when N is a power of 2, the maximum depth of the recursion stack is $\log_2(N)$

Q.2. The solution employs a binary search approach to efficiently find the minimum maximum value. It iteratively adjusts the search range and counts the operations required to ensure each element is less than or equal to a candidate maximum value. The result represents the minimum maximum value achievable within the given operation limit K. Now the upper limit is adjusted to mid-1 to search in the left half if there is any no exist such that all the condition are met .

```
This is the function for checking if possible to get all the array elements less
than equal to mid by at most operation of k.

ll find(vector<ll>& A,ll k,ll mid){
    ll op=0;
    ll n=A.size();
    // ll temp;
    for(ll i=0;i<n;i++){
        if(A[i]>mid){
            ll temp=ceil(A[i]/mid)+1;
            temp--;
            op+=temp;

        }
    }
    return op>k?0:1;
}
```

Time complexcity: O(N*log INT_MAX)

```
Initially the range is from 0 to INT_MAX. so in each step the range is reduced to
half and extran O(n) cost to scan the array . So the time complexcity is
O(N*log(INT_MAX)).
```

Space complexcity:O(1)

NO space required accept some local variable.

Q.3.

Explanation: Here first the elements of the two given array i am storing them as pair and sorting them. so after sorting the vector of the array is sorted by first element of the pair and now the problem is left to find the no of inversion.

Time complexcity: O(N*log N)

```
For sorting the time complexcity is O(Nlog N) and then for finding inversion is done
using the same technique as merge sort so again o(Nlog N). Hence overall complexcity is
oO(N log N).
```

space complexcity is O(2*N) for storing the elements in vector of pair.