Allocate Books

```
bool isPossible(vector<int>& arr, int n, int m, int curr_min) {
    int studentsRequired = 1;
    int curr_sum = 0;

    for (int i = 0; i < n; i++) {
        if (arr[i] > curr_min) {
            return false;
        }

        if (curr_sum + arr[i] > curr_min) {
            studentsRequired++;
            curr_sum = arr[i];

            if (studentsRequired > m) {
                return false;
            }
        } else {
            curr_sum += arr[i];
        }
    }
    return true;
}

int findPages(vector<int>& arr, int n, int m) {
    long long sum = 0;

    if (n < m) {
        return -1;
    }

    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }

    int start = 0, end = sum;
    int result = INT_MAX;

    while (start <= end) {
        int mid = (start + end) / 2;
        if (isPossible(arr, n, m, mid)) {
            result = min(result, mid);
            end = mid - 1;
        } else {
            start = mid + 1;
        }
    }
    return result;
}
```

. Capacity To Ship Packages Within D Days

```
class Solution {
public:

    // function for finding req_days if we take mid as least weight
capacity

    bool is_possible(vector<int>& weights, int mid, int days)
    {
        int n = weights.size();

        int req_days = 1;

        int curr_weight = 0;

        for(int i = 0; i < n; i++)
        {
            if(curr_weight + weights[i] <= mid)
            {
                curr_weight += weights[i];
            }

            // if inclusion of curr. weight will exceed the mid value

            else
            {
                req_days++;

                curr_weight = weights[i];
            }
        }

        return req_days <= days;
    }

    int shipWithinDays(vector<int>& weights, int days) {

        int n = weights.size();

        // low will be = max. value of elements of array
```

```cpp
        int low = *max_element(weights.begin(), weights.end());

        // high will be = sum of whole array

        int high = 0;

        for(int i = 0; i < n; i++)
        {
            high += weights[i];
        }

        // apply binary search and find the least weight capacity

        int mini = low;

        while(low <= high)
        {
            // find mid

            int mid = low + (high - low) / 2;

            // check if it is possible to check mid as least weight 
capacity

            if(is_possible(weights, mid, days))
            {
                mini = mid;

                high = mid - 1;
            }
            else
            {
                low = mid + 1;
            }
        }

        return  mini;
    }
};
```