Minimum Difficulty of a job schedul

By :Manar Mahmoud Eldosouky

Menna gamal Mohamed

Problem Description

We are given an array jobs, where each element represents the difficulty of a job.
We are also given an integer d representing the number of days.
The goal is to schedule all jobs over exactly d days under the following constraints:
Each day must have at least one job.
Jobs must be scheduled in order (no reordering).
The difficulty of a day is defined as the maximum difficulty of the jobs done on that day.
The total difficulty is the sum of the difficulties of all days.
problem in letcode:
https://leetcode.com/problems/minimum-difficulty-of-a-job-schedule/description/

Input–Output Examples

1

jobs = [6, 5, 4, 3, 2, 1]

d = 2

Day 1: [6, 5, 4, 3, 2] → max = 6

2

jobs = [9, 9, 9]

d=4

Number of days is greater than number of jobs → impossible

Output:-1

3

jobs = [1, 1, 1]

total Difficulty =1+1+1=3

Algorithm 1: Naive Recursive Solution (Brute Force)

Description

This approach tries all possible ways to split the jobs into d days using recursion.

At each step:

Assign a prefix of jobs to the current day.

Recursively solve the remaining jobs for the remaining days.

Compute the total difficulty and choose the minimum.

This approach does not use memoization, which leads to repeated computations.

Pseudocode of naive solution

```
function dfs(jobs, d, start):
    if remaining jobs < d:
        return -1

    if d == 1:
        return max(jobs[start ... end])

    minCost = INF
    maxToday = 0

    for i from start to n - d:
        maxToday = max(maxToday, jobs[i])
        remaining = dfs(jobs, d - 1, i + 1)
        if remaining != -1:
            minCost = min(minCost, maxToday +
remaining)

    return minCost
```

Time Complexity:O(n^d)

Space Complexity :O(d)

Problem List

Time Limit Exceeded | Editorial | Solutions | Submissions

Limit Exceeded

9 / 34 testcases passed

Executed Input

Use Testcase

```
difficulty =
[302,102,681,863,676,243,671,651,612,162,561,394,856,601,30,6,257,921,405,716,126,15
6,889,699,668,930,139,164,641,801,480,756,797,915,275,709,161,358,461,938,914,557,12
4,315]
```

C++

Analyze Complexity

```cpp
class Solution {
public:
    int dfs(vector<int>& jobs, int d, int start) {
        int n = jobs.size();
        if (n - start < d) return -1;

        if (d == 1) {
            return *max_element(jobs.begin() + start, jobs.end());
```

View more

Write your notes here

Code

C++ | Auto

```cpp
class Solution {
public:
    int dfs(vector<int>& jobs, int d, int start) {
        int n = jobs.size();
        if (n - start < d) return -1;

        if (d == 1) {
            return *max_element(jobs.begin() + start, jobs.end());
        }

        int min_cost = INT_MAX;
        int max_today = 0;

        for (int i = start; i <= n - d; ++i) {
            max_today = max(max_today, jobs[i]);
            int remaining = dfs(jobs, d-1, i+1);
            if (remaining != -1)
                min_cost = min(min_cost, max_today + remaining);
        }

        return min_cost;
    }

    int minDifficulty(vector<int>& jobDifficulty, int d) {
        return dfs(jobDifficulty, d, 0);
    }
};
```

Ln 5, Col 38 | Saved

Run

Testcase | Test Result

Algorithm 2: Optimized Dynamic Programming (Memoization)

This approach improves the naive solution by using memoization.
dp[start][d] = minimum difficulty to schedule jobs from index start using d days
Before computing a state, we check if it was already solved.

Pseudocode of optimized solution”Memoization”

```
function dfs(jobs, d, start):
  if memo[start][d] != -1:
    return memo[start][d]

  if d == 1:
    memo[start][d] = max(jobs[start ... end])
    return memo[start][d]

  minCost = INF
  maxToday = 0

  for i from start to n - d:
    maxToday = max(maxToday, jobs[i])
    remaining = dfs(jobs, d - 1, i + 1)
    minCost = min(minCost, maxToday + remaining)

  memo[start][d] = minCost
  return minCost
```

time complexity :O(n^2 ×d)

Space complexity :O(n×d)

Description | Editorial | Solutions | Submissions | Accepted ×

All Submissions

Menna_Gamal2006 submitted at Dec 15, 2025 05:07

Editorial | Solution

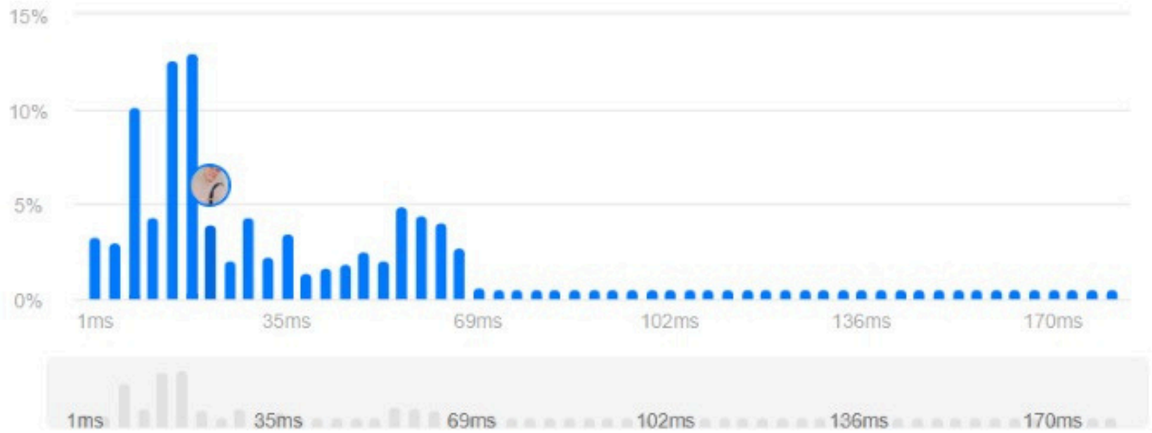
Runtime

21 ms | Beats 54.50%

Analyze Complexity

Memory

11.05 MB | Beats 66.19%



Code | C++

```cpp
class Solution {
public:
    int minDifficultyHelper(vector<int>& jobs, int d, int start, vector<vect
        int n = jobs.size();
        if (memo[start][d] != -1) return memo[start][d];

        if (d == 1) {
            memo[start][d] = *max_element(jobs.begin() + start, jobs.end());
```

View more

More challenges

Code

C++ | Auto

```cpp
class Solution {
public:
    int minDifficultyHelper(vector<int>& jobs, int d, int start, vector<vector<int>>& mem
        int n = jobs.size();
        if (memo[start][d] != -1) return memo[start][d];

        if (d == 1) {
            memo[start][d] = *max_element(jobs.begin() + start, jobs.end());
            return memo[start][d];
        }

        int min_cost = INT_MAX;
        int max_today = 0;

        for (int i = start; i <= n - d; ++i) {
            max_today = max(max_today, jobs[i]);
            int remaining = minDifficultyHelper(jobs, d-1, i+1, memo);
            min_cost = min(min_cost, max_today + remaining);
        }

        memo[start][d] = min_cost;
        return min_cost;
    }

    int minDifficulty(vector<int>& jobDifficulty, int d) {
        int n = jobDifficulty.size();
        if (n < d) return -1;
        vector<vector<int>> memo(n, vector<int>(d+1, -1));
        return minDifficultyHelper(jobDifficulty, d, 0, memo);
    }
};
```

Ln 32, Col 1 | Saved

Run

optimized solution”BOTTOM UP”

This approach improves the naive solution by using Dynamic Programming.
.dp[i] represents the minimum difficulty to schedule the first i jobs
.For each day, we try all valid split points and choose the minimum total difficulty
The maximum difficulty of the current day is added to the best result of the .previous days
Space optimization is applied using one-dimensional DP arrays instead of a 2D .table

time complexity :O(n^2 ×d)
Space complexity :O(n)

optimized solution”BOTTOM UP”

Problem List

Description | Accepted | Editorial | Solutions | Submissions

All Submissions

Accepted 34 / 34 testcases passed

Menna_Gamal2006 submitted at Dec 23, 2025 03:54

Editorial | Solution

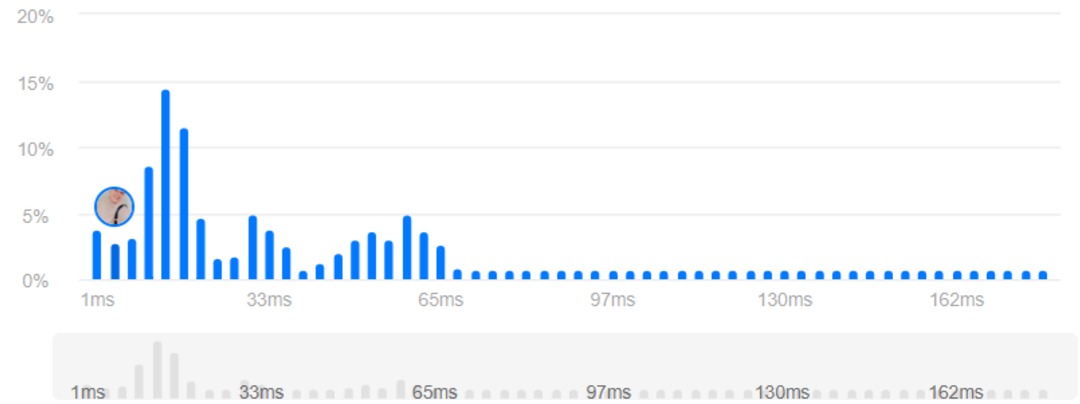
Runtime

7 ms | Beats 95.75%

Analyze Complexity

Memory

10.94 MB | Beats 73.68%



Code | C++

```cpp
class Solution {
public:
    int minDifficulty(vector<int>& jobDifficulty, int d) {
        int n = jobDifficulty.size();

        // If the number of days is greater than the number of jobs,
        // it is impossible to schedule them
        if (n < d) return -1;
```

View more

Code

C++ | Auto

```cpp
class Solution {
public:
    int minDifficulty(vector<int>& jobDifficulty, int d) {
        int n = jobDifficulty.size();

        if (n < d) return -1;
        vector<int> prev_dp(n + 1, 1e9);

        int max_so_far = 0;
        for (int i = 1; i <= n; i++) {
            max_so_far = max(max_so_far, jobDifficulty[i - 1]);
            prev_dp[i] = max_so_far;
        }

        for (int day = 2; day <= d; day++) {
            vector<int> current_dp(n + 1, 1e9);

            for (int i = day; i <= n; i++) {
                int current_max = 0;

                for (int j = i; j >= day; j--) {
                    current_max = max(current_max, jobDifficulty[j - 1]);
                    current_dp[i] = min(
                        current_dp[i],
                        prev_dp[j - 1] + current_max
                    );
                }
            }
            prev_dp = current_dp;
        }
        return prev_dp[n];
    }
};
```

Ln 29, Col 13 | Saved

Run | Submit

optimized solution”Monotonic Stack"

Reducing Complexity with Monotonic Stack

1. Naive DP checks all split points → $O(d \times n^2)$.
2. Use a monotonic stack to keep job indices in decreasing difficulty.
3. Pop all jobs with smaller or equal difficulty to skip unnecessary computations.
4. Each job is pushed/popped once → transitions become O(n) per day, total $O(d \times n)$.

time complexity :O(n×d)
Space complexity :O(n)

"optimized solution”Monotonic Stack

Description | Accepted | Editorial | Solutions | Submissions

All Submissions

Accepted 34 / 34 testcases passed

Menna_Gamal2006 submitted at Dec 23, 2025 04:23

Editorial | Solution

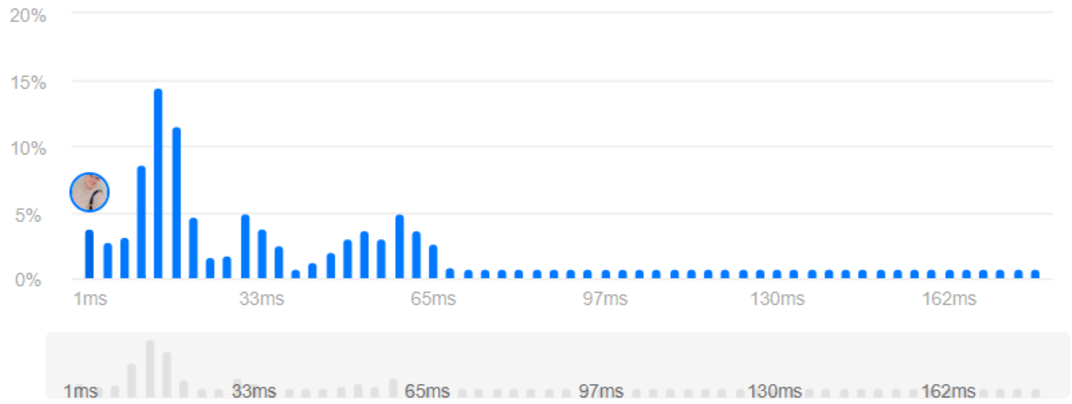
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

11.56 MB | Beats 18.58%



Code | C++

```cpp
class Solution {
public:
    int minDifficulty(vector<int>& jobDifficulty, int d) {
        int n = jobDifficulty.size();
        if (n < d) return -1;

        vector<int> dp(n, 1e9);
        vector<int> prev_dp(n);
```

View more

Code

C++ | Auto

```cpp
public:
    int minDifficulty(vector<int>& jobDifficulty, int d) {
        int n = jobDifficulty.size();
        if (n < d) return -1;

        vector<int> dp(n, 1e9);
        vector<int> prev_dp(n);

        int max_val = 0;
        for (int i = 0; i < n; i++) {
            max_val = max(max_val, jobDifficulty[i]);
            prev_dp[i] = max_val;
        }
        for (int day = 2; day <= d; day++) {
            stack<pair<int, int>> st;

            for (int i = day - 1; i < n; i++) {
                int min_prev_dp = prev_dp[i - 1];

                while (!st.empty() && jobDifficulty[st.top().first] <= jobDifficulty[i]) {
                    min_prev_dp = min(min_prev_dp, st.top().second);
                    st.pop();
                }
                int current_total = jobDifficulty[i] + min_prev_dp;

                if (!st.empty()) {
                    current_total = min(current_total, dp[st.top().first]);
                }

                dp[i] = current_total;
                st.push({i, min_prev_dp});
            }
            prev_dp = dp;
        }
        return prev_dp[n - 1];
    }
```

Ln 24, Col 18 | Saved

Run | Submit

Empirical Results

Algorithm	Input Size (N)	Time (ms)	Memory (MB)
Memoization	300	25ms	12.5MB
Bottom-Up	300	18ms	2.1MB
Monotonic Stack	300	2ms	2.1MB

Comparison Discussion

Feature	(1) Naive Recursion	(2) Top-Down (Memo)	(3) Bottom-Up (Space Opt)	(4) Monotonic Stack
Approach	Brute Force	Dynamic Programming	Dynamic Programming	DP + Advanced DS
Logic	Explores all paths	Top-Down + Cache	Bottom-Up (Iterative)	Optimized Iterative
Time Complexity	$O(n^d)$	$O(d \times n^2)$	$O(d \times n^2)$	$O(d \times n)$
Space Complexity	$O(d)$ (Stack)	$O(d \times n)$ (Table)	$O(n)$ **(Array)**	$O(n)$ **(Array)**
Efficiency	Extremely Slow	Good	Very Good	**Elite / Fastest**
LeetCode Status	Time Limit Exceeded	Accepted	Accepted	**Accepted (Fastest)**