Greedy Algorithms

Farhan Ahmad

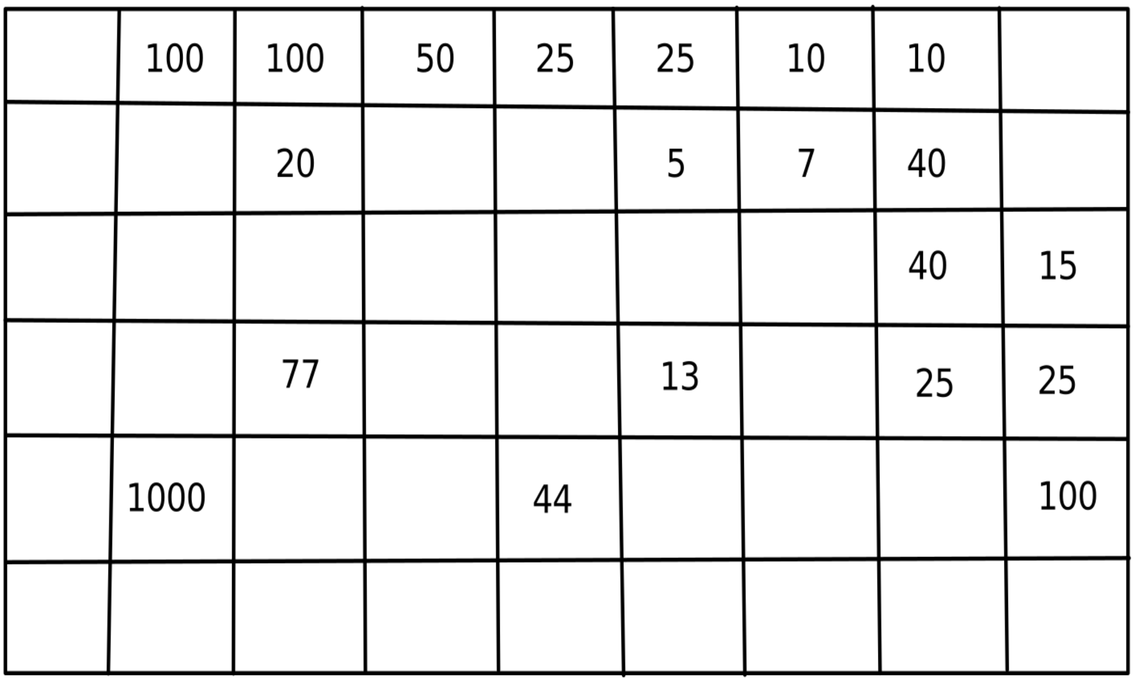
28 Jan 2022

সূচীপত্র

1 Introduction to Greedy Algorithms

Greedy শব্দটা শুনেই বুঝা যাচ্ছে যে লোভ এর কিছু একটা রয়েছে। আসলে greedy algorithm এ আমরা লোভীর মতো যেইটা করা সব থেকে ভালো মনে হবে ওই পরিস্থিতিতে, সেইটা করি। এক্ষেত্রে আমরা কোন কিছু লোভীর মতো নিয়ে আবার রেখে দিব এমন করব না! সহজ কথাই যেই সিধান্ত একবার নিব, ওইটা থেকে আর নড়বো না! এখন এভাবে লোভীর মতো নেওয়া তো সব সময় ভালো নাও হতে পারে, আবার অনেক সময় এভাবে লোভীর মতো নেওয়া optimal solution দিতে পারে।

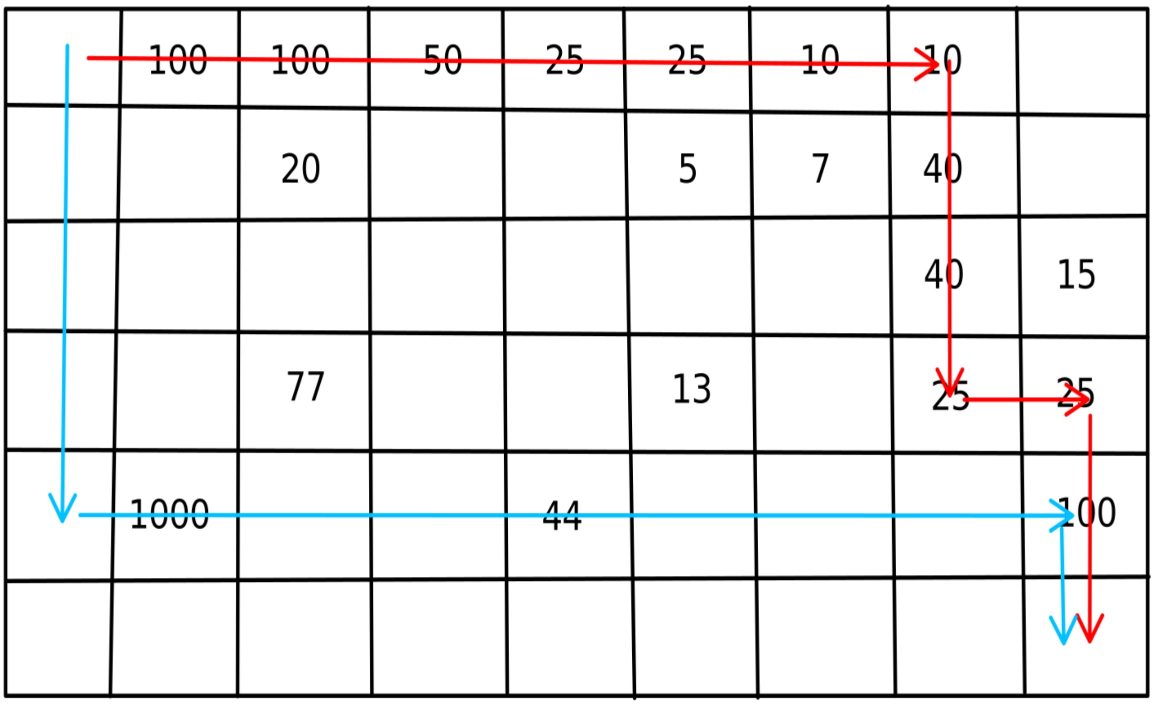
একটা প্রব্লেমটার কথাই আসি, ধরা যাক একটা grid আছে এবং grid এর কিছু কিছু cell এ কিছু পরিমাণ টাকা আছে। নিচে এমন একটা grid দেওয়া হয়েছে।



	100	100	50	25	25	10	10	
		20			5	7	40	
							40	15
		77			13		25	25
	1000			44				100

আমরা top-left corner এ আছি আর আমাদের যেতে হতে bottom-right corner এ। আমরা যেই cell এ আছি সেইখান থেকে হয় ডান বা নিচে এ যেতে যদি ওখান এ কোন cell থাকে, কোন cell টাকা থাকলে ওই টাকা আমরা নিয়ে নিব। এখন আমরাদের এমন ভাবে যেতে হবে যেন সবচেয়ে বেশি টাকা জোগাড় করতে পারি! আমরা লোভীর মতো মতো যেইদিক এ বেশি টাকা, সেইদিক যেতে পারি। এভাবে গেলে আমরা যেই path হয়ে যাব, এইটা নিচের ছবিতে লাল দিয়ে নির্দেশ করা হল, এতে আমরা মোট ৫৫০ টাকা পেতে পারি! কিন্তু এর থেকেও কিন্তু ভালো আরেকটা path ছিল, যেটায় আমরা আর বেশি টাকা পেতে পারতাম। path টি নীল রঙ দিয়ে নির্দেশ করা হয়েছে এখানে।

এখানে দেখা যাচ্ছে যে, greedy approach কাজ করছে না। আসলে greedy কাজ করবে কি করবে না, এইটা বের করাই সবচেয়ে কঠিন কাজ। আবার একটা প্রব্লেম এ অনেক greedy approach থাকতে পারে, কোনটা ঠিক এইটা বের করাটাই চ্যালেঞ্জ হয়ে দাড়ায়। codeforces এ ৮০০ rated প্রব্লেম ও greedy প্রব্লেম হতে পারে, আবার ৩৫০০ rated প্রব্লেম ও greedy প্রব্লেম হতে পারে, তাই বুঝাই যাচ্ছে কত ধরনের greedy approach থাকতে পারে যে সব ধরনের লেভেলেরই মানুষ কে চ্যালেঞ্জ করে!



	100	100	50	25	25	10	10	
		20			5	7	40	
							40	15
		77			13		25	25
	1000			44				100

2 Coin Problem

ধরা যাক, তোমার কাছে 1,2,5 টাকার কয়েন আছে। প্রতিটা কয়েন অসীম পরিমাণ এ রয়েছে। এখন তুমি একটা x টাকার জিনিস কিনবা, তো সবচেয়ে কম কয়টি কয়েন ব্যবহার করে কেনা সম্ভব বের করতে হবে। তো আমাদের মাথায় প্রথমেই যেই চিন্তা আসবে যে সবচেয়ে বেশি টাকার কয়েন যত পারি ব্যবহার করব, তারপর ওই কয়েনকে বাদ দিলে যেই কয়েন সবচেয়ে বেশি টাকার, সেই কয়েনটি ব্যবহার করব। এভাবে করতেই থাকব শেষ পর্যন্ত! অর্থাৎ লোভীর মতো সবচেয়ে বড় কয়েন c ব্যবহার করব, যেন $c \leq x$। এরপর $x - c$ এর জন্য উত্তর বের করা এর চেষ্টা করব এবং ওই উত্তর এর সাথে ১ যোগ করে দিব। এভাবে এই greedy approach যে এই সেট কয়েনর জন্য যে সব সময় optimal উত্তর দেয়, এইটা প্রমাণ করা সম্ভব। এই approach টার একটা কোড দেওয়া হলঃ

```
vector < int > coins = {1, 2, 5};

int ans(int x){
    if(x == 0) return 0;
    int res = 1e9; // setting something big as inf
    for(int i = coins.size() - 1; i >= 0; i--){
        if(coins[i] <= x){
            res = 1 + ans(x - c);
            break;
        }
    }
    return res;
}
```

এই সেট কয়েনের জন্য কাজ করলেও সব সেট কয়েনর জন্য কাজ করবে না! যেমন ধরা যাক, 2, 5 এই সেট কয়েনর কথা। যদি $x = 6$ মনে করি, তাহলে প্রথমে আমরা 5 কয়েনটা ব্যবহার এর চেষ্টা করব, এরপর আর বাকি থাকে 1। কিন্তু 1 এর সমান বা ছোট কোন কয়েন নাই! তাই এভাবে করলে x এই

বানাতে পারব না! কিন্তু 2 আর 5 কয়েন দিয়ে কিন্তু $x = 6$ টকার জিনিস কেনা সম্ভব ছিল, 3 টা 2 টাকার কয়েন ব্যবহার করলেই হতো! তাই সব সেট কয়েনের জন্য এরকম greedy approach কাজ করবে না!

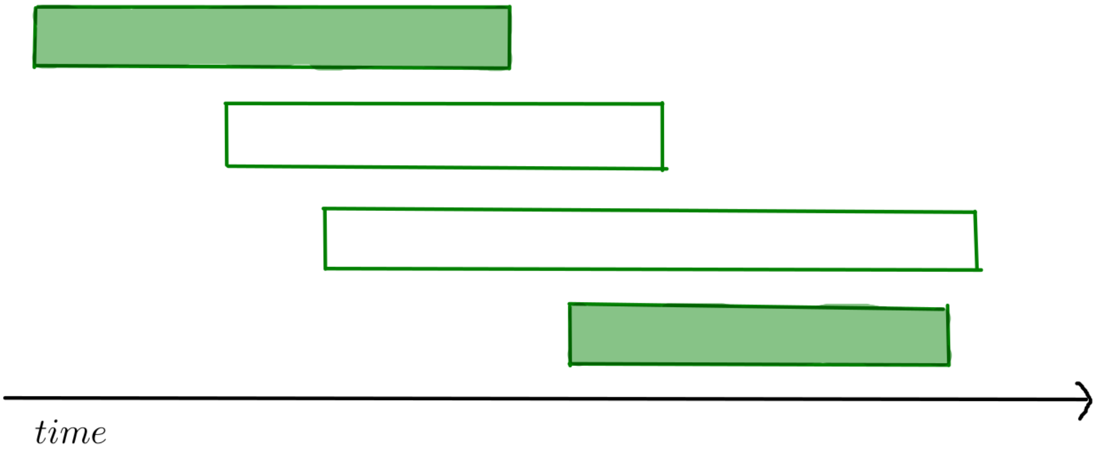
3 Scheduling Problem

একটা classic প্রব্লেম দেখাযাক এখন। আমরা ধর মুভি দেখতে যাব, আমরা মুভি হলে গিয়ে দেখছি i_{th} মুভি শুরু হবে $start_i$ এবং শেষ হবে end_i সময়ে। আমরা যত পারি ততো মুভি দেখার চেষ্টা করব, কিন্তু একটা মুভি শুরু থেকে দেখব এবং শুরু করলে শেষ করা পর্যন্ত উঠবো না! অর্থাৎ, যদি ঠিক করি i_{th} এবং j_{th} দেখব, তাহলে, এদের সময় overlap করা যাবে না। এখন আমাদের বের করতে হবে সব চেয়ে বেশি কইটা মুভি দেখতে পারব, আর ওই মুভিগুলো কি কি।

এই প্রব্লেমে অনেক greedy approach ব্যবহার করা যাবে। কিন্তু বিষয় হচ্ছে, কোন গুলো কাজ করবে? নাকি কোনটাই করবে না :(

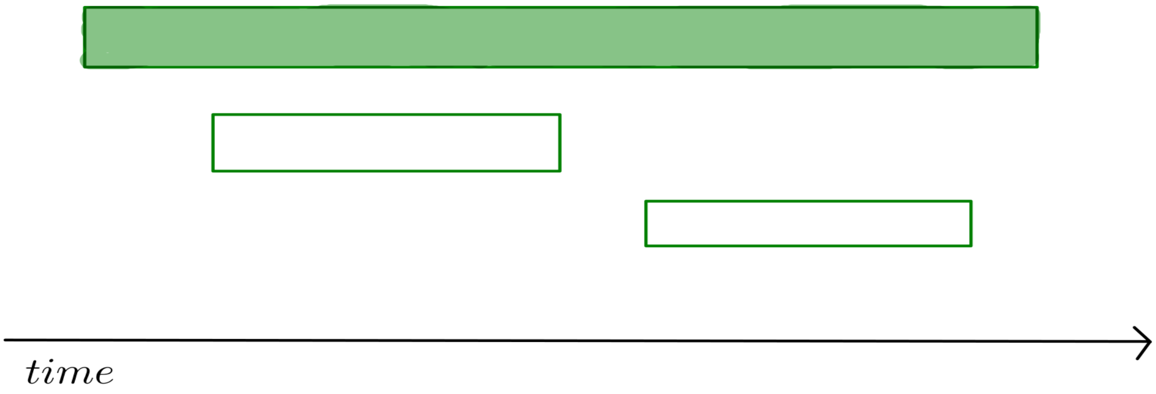
Greedy Approach ১

আমরা যেই মুভি আগে শুরু হয়, ওইটা প্রথম এ দেখব। ওই শেষ করার পর যেই মুভি প্রথম শুরু হবে, ওইটা দেখা শুরু করব। এভাবে দেখতেই থাকব যতক্ষণ দেখতে পারব! এমন একটা উদাহরণ দেখা যাক,



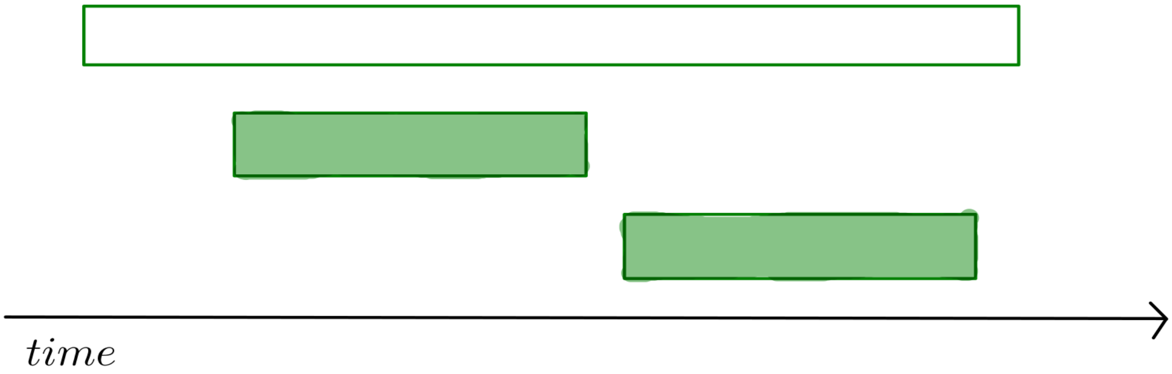
এখানে প্রতিটি আয়ত একটা মুভি এর সময়কে নির্দেশ করে। এখানে যদি আমরা এই greedy aprroach এ মুভি ঠিক করি, তাহলে প্রথম আর চতুর্থ মুভি দেখতে পারব। আর এইক্ষেত্রে সবচেয়ে বেশি ২ টা মুভি দেখাই সম্ভব!

কিন্তু এই greedy approach সবসময় ঠিক উত্তর নাও দিতে পারে। যেমন এই ক্ষেত্রে দেখা যাচ্ছে। আমরা প্রথম মুভি টা দেখা শুরু করেছি কারণ এইটা সবার আগে শুরু হয়েছে। কিন্তু এই মুভিটা না দেখে যদি আমরা 2_{nd} বা 3_{rd} মুভি টা দেখতাম, তাহলে মোট দুইটা মুভি দেখা হতো। তাই যেই মুভি আগে শুরু হয়েছে, ওই মুভি যে সবার আগে দেখতে হবে এমন কোন কথা নেই। তাই এই greedy approach সবক্ষেত্রে optimal answer দিবে না!

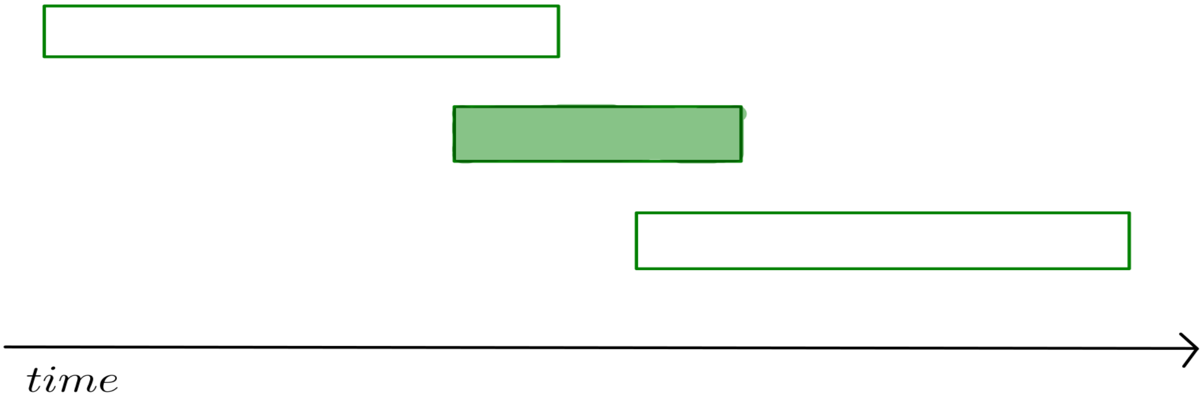


Greedy Approach ২

এখন আমরা ভাবতে পারি সবচেয়ে ছোট সময়ের মুভি গুলো দেখার চেষ্টা করব। অর্থাৎ যেই মুভি এর $duration$ সবচেয়ে কম, সেইটা দেখব বলে ঠিক করব, তারপর $duration$ ২য় ছোট মুভি দেখার চেষ্টা করব। এভাবে যেই কইটা মুভি দেখতে পারি, সেই কইটা মুভি দেখব। তাহলে আগের বার যেই উদাহরণে সমস্যা হয়েছিল, এইবার সেইটা আর হবে না!



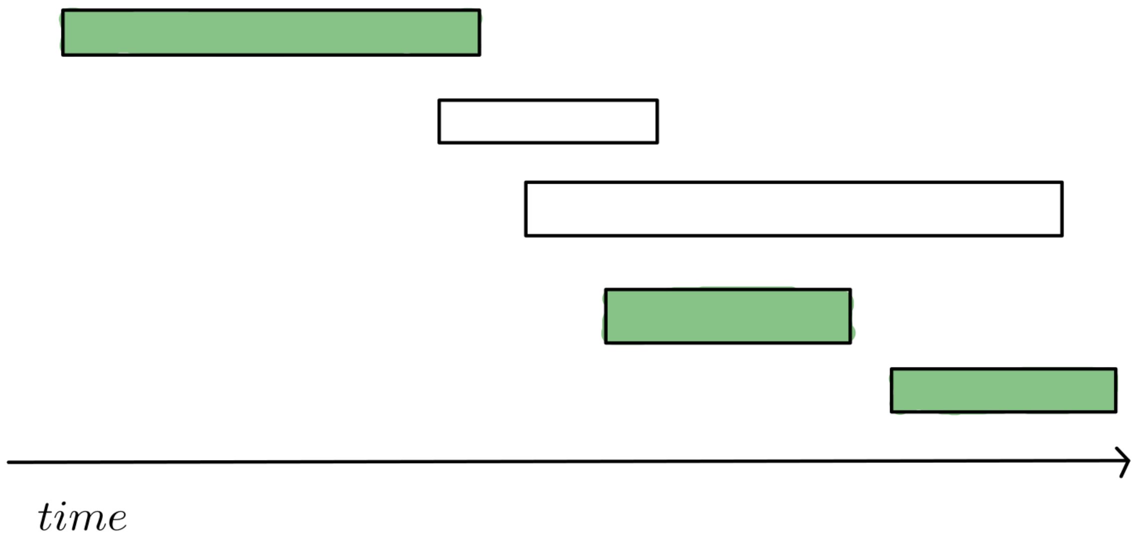
এইবার approach টা দেখে ঠিক মনে হলেও আসলে এইটাও সবসময় ঠিক উত্তর দিবে না :(এমন একটা case দেখা যাক।



এখানে সবচেয়ে ছোট মুভিটি দেখার চিন্তায় শুধু একটাই মুভি দেখা হয়েছে। কিন্তু যদি প্রথম আর শেষ মভি টা দেখতাম তাহলে দুইটা মভি দেখা যেত, তাই এই approach ও কাজ করবে না সব সময়।

Greedy Approach ৩

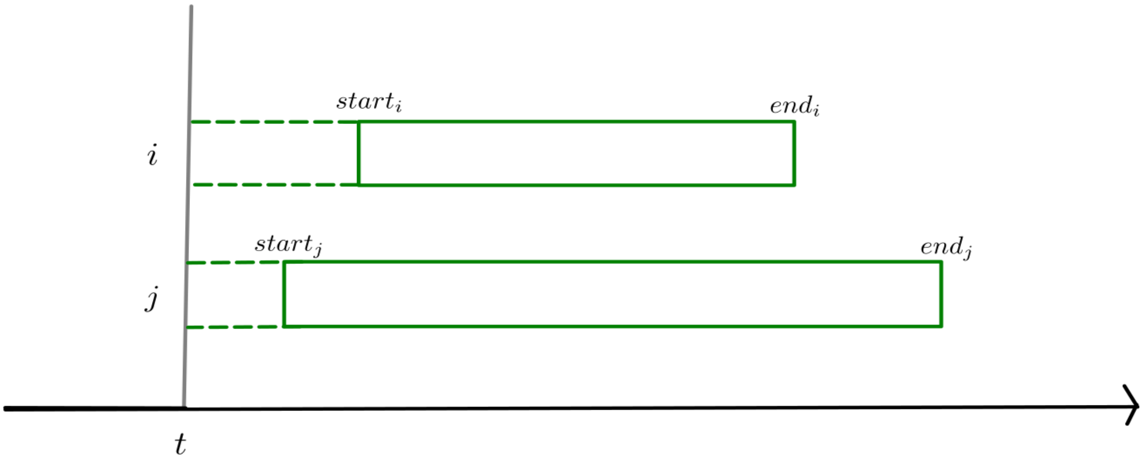
এবার আমরা কখন মুভি শেষ হচ্ছে, টার ভিত্তিতে মুভি ঠিক করব। যেই মুভি আগে শেষ হবে, সেই মুভি আগে দেখার চেষ্টা করব। অর্থাৎ, t সময়ে যেই মুভি গুলো এখন শুরু হয়নাই, ওই মুভিগুলোর মধ্যে যেই মুভি সবার আগে শেষ হবে, সেই মুভিটি দেখব।



এইক্ষেত্রে আমরা ৩ টা মুভি দেখতে পারব এই appraoch দিয়ে। কিন্তু আগের appraoch দুইটি দিয়ে মাত্র দুইটা মুভি দেখতে পারতাম। এখন মজার বিষয় হচ্ছে এই greedy appraoch সব সময় optimal solution দিবে! কেন সব সময় ঠিক উত্তর দিবে, নিজেরা প্রমাণ এর চেষ্টা করে করে দেখো তো! greedy algorithm ঠিক কি ভুল এইটা প্রমাণ করাও algorithm টা বের করার মতোই জরুরি। কারণ এই প্রব্লেম এর মতো অনেক প্রব্লেম এই অনেক greedy appraoch থাকে, তো কোনটা ঠিক কোনটা ভুল এইটা বের না করতে পারলে অনেক wrong submission হতে পারে :(

এভাবে প্রমাণ করা যেতে পারে, ধর আমরা সময় t তে আছি এবং ওই মুহূর্তে কোন মুভি দেখছি না। এখন যেই মুভিগুলোর শুরুর সময় $start_i >= t$, তাদের মধ্যে যেই i এর end_i সবচেয়ে কম হবে, সেই মুভিটা দেখব। এখন আমরা $[t, start_i)$ সময় পর্যন্ত বসে থাকব, আর $[start_i, end_i)$ পর্যন্ত মুভি দেখব এবং end_i এ আবার মুভি ঠিক করব পরের মুভি কোনটা দেখব। সহজ কোথায় এর পরে i মুভি দেখলে $[t, end_i)$ পর্যন্ত অন্য কোন কাজ করা হবে হবে, শুধু i মুভি এই দেখা হবে! এখন যদি মুভি i না দেখে মুভি j দেখতাম, যেন $end_j > end_i$, তাহলে একইভাবে আমরা $[t, end_j)$ পর্যন্ত j পিছেই পরে থাকব, ফলে আমরা end_i এ ফ্রি না হয়ে end_j হচ্ছি। দুই ক্ষেত্রেই ১ টাই মুভি দেখা হচ্ছে, তাই এভাবে j মুভিকে কে দেখা optimal না।

৩ টা appraoch এই কোড করে ফেলার চেষ্টা কর! প্রতিটা approach $O(NlogN)$ (এখানে N মোট মুভি সংখ্যা) complexity তে কোড করে ফেলা সম্ভব।



4 Min Split

আরেকটা classic প্রব্লেম দেখা যাক। ধর তোমার কাছে একটা array A এবং একটা পূর্ণসংখ্যা k আছে। তুমি এই array টাকে কিছু ভাগে ভাগ করতে পারবা। শর্ত হচ্ছে, যেন প্রতি ভাগের যোগফল যেন k এর বেশি না হয়। array টাকে যত কম ভাগে ভাগ করে এই শর্ত অর্জন টা বের করতে হবে, আর সম্ভব না হলেও সেটা বলতে হবে। যেমন একটা উদাহরণ দেখা যাক, মনে করি $k = 15$ এবং $A = [12, 1, 5, 6, 7, 2, 3]$।

$$[12, 1, 5, 6, 7, 2, 3] \rightarrow [12] + [1, 5, 6] + [7, 2, 3]$$

এখানে array টাকে ২ বার ভাগ করা হয়েছে, এর কম বার ভাগ করা সম্ভব না। এখন যেকোনো array A এবং যেকোনো পূর্ণসংখ্যা k এর জন্য কিভাবে এটা সমাধান করা যায়, বের করার চেষ্টা করো তো!

Observation ১: A array এর সাইজ যদি n হয়, তাহলে আমরা সবচেয়ে বেশি $n - 1$ বার ভাগ করতে পারব। এই ভাগ করার পরও যদি প্রতিটা ভাগের যোগফল k এর বেশি হয়, তাহলে সেই ক্ষেত্রে ওই k এর মান এর জন্যও array টা ভাগ করে সম্ভব না। $n - 1$ বার ভাগ করলে প্রতিটি ভাগ এ মাত্র একটা সংখ্যা থাকবে। তাই $n - 1$ বার ভাগ করলে করে n ভাগ হবে, প্রতি ভাগ হচ্ছে আসলে array এর একটা element। তাই $maxA_i > k$ হলে কোন সমাধান নেই। আর যদি, $maxA_i \leq k$ হয়, তাহলে অবশ্যই সমধান রয়েছে, যেহেতু $n - 1$ বার ভাগ করা একটা সমাধান।

Observation ২: ধরি, আমাদের কোন এক ভাবে ভাগ করা সম্ভব। ভাগ করার করার পর প্রতিটা ভাগ কে একটা গ্রুপ ভাবি। প্রথম যেই গ্রুপ হবে, সেইটা হচ্ছে A array রএকটা prefix। ওই prefix বাদ দিয়ে বাকি array এর আমরা independently solution বের করতে পারি। কারণ বাকি অংশের সাথে প্রথম prefix এর কোন সম্পর্কই নাই। তো প্রথম গ্রুপ কে যত বড় করতে পারি, ততো ভালো হবে। কারণ ছোট prefix নিলে পরের অংশে বেশি element থাকবে, এতে উত্তর বেশি চলে আসতে পারে। কিন্তু যেহেতু, বড় prefix নিতে পারছি, তাই ওইটা নিলে পরের অংশের element সংখ্যা কমে যাচ্ছে, তাই এইটা বেশি লাভজনক হচ্ছে।

প্রথমেই আমরা দেখব কোন সমাধান আছে নাকি অর্থাৎ $maxA_i \leq k$, সব i এর জন্যও সত্য নাকি। যদি সমাধান থাকে, তাহলে সবচেয়ে বড় prefix নিব যেন সেই prefix এর যোগফল k এর থেকে বড় না হয়। তারপর বাকি অংশের জন্য একই ভাবে উত্তর বের করব।

আবার $A = 12, 1, 5, 6, 7, 2, 3$ এবং $k = 15$ এর জন্যও উত্তর বের করার চেষ্টা করা যাক।

যেহেতু, $maxA_i$ = 12 এবং $12 \le 15$, তাই এই ক্ষেত্রে সমাধান আছে। $[12, 1, 5, 6, 7, 2, 3]$ সবচেয়ে বড় prefix যেইটার যোগফল k এর বড় না হচ্ছে $[12, 1]$

$$[12, 1, 5, 6, 7, 2, 3] \rightarrow [12, 1] + [5, 6, 7, 2, 3]$$

এখন $[5, 6, 7, 2, 3]$ এর সমাধান করা লাগবে। এই array এর সবচেয়ে বড় prefix যেইটার যোগফল k এর সমান বা কম হচ্ছে, $[5, 6]$।

$$[5, 6, 7, 2, 3] \rightarrow [5, 6] + [7, 2, 3]$$

এখন $[7, 2, 3]$ এর সমাধান করা লাগবে। এই array এর সবচেয়ে বড় prefix যেইটার যোগফল k এর সমান বা কম হচ্ছে, $[7, 2, 3]$।

$$[7, 2, 3] \rightarrow [7, 2, 3] + []$$

যেহেতু আর কোন element বাকি নেই, তাই উত্তর হচ্ছে ২। কারণ মতো দুইবার ভাগ করেছি (শেষবারের টা ধরা হচ্ছে না কারণ, প্রকিতপক্ষে কিছু ভাগ করা হয়নাই। কারণ ডানদিকে একটা element ও ছিল না!)

একটা sample কোড দেওয়া হল:

```
int n , k;
cin >> n >> k;

vector < int > A(n);
int mx = 0; // for storing the element of A

for(int i = 0; i < n; i++){
    cin >> A[i];
    mx = max(mx, A[i]);
}
if(mx > k){
    cout << "NO SOLUTION";
    return 0;
}
int sum = 0; // storing the sum of prefix
int answer = 0;

for(int i = 0; i < n; i++){
    if(sum + A[i] <= k){ // can take A_i
       sum += A[i];
    }else{
        // Can't take A_i,so we are going to ingore the elements of [0, i)
        sum = 0; // Now we have no relation with the previous elements
        // from now we will find the answer for [i,n) range
        sum += A[i];
        // As we spliting the array, we need to increment the answer by 1
        answer++;
    }
}
cout << answer << '\n'; // Our Answer
```

5 Sum Minimization

মনে করি আমাদের একটা array A আছে। আমদের এমন একটা একটা পূর্ণসংখ্যা x বের করতে হবে যেন,

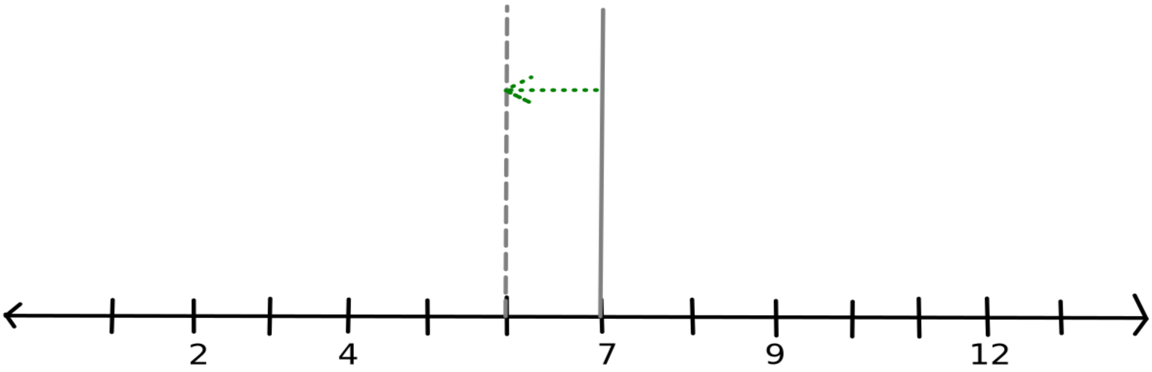
$$|a_1 - x| + |a_2 - x| + |a_3 - x| ... + |a_n - x|$$

এই মান টা যত পারা যায়, কম হয়। একটা উদাহরণ দেখা যাক। একটা array $a = [12, 2, 9, 7, 4]$। এক্ষেত্রে, $x = 7$ হল optimal solution। x যেকোনো মান 7 ছাড়া হলে আরও বড় যোগফল দিবে।

$$|12 - 7| + |2 - 7| + |9 - 7| + |7 - 7| + |4 - 7| = 15$$

x এর optimal মান হচ্ছে array টার মধ্যক। একটা array এর মধ্যক হচ্ছে সেই সংখ্যা যেটা array টা sort করলে মাঝখানে থাকে। যেমন, $[12, 2, 9, 7, 4]$ কে sort করলে $[2, 4, 7, 9, 12]$ মাঝখানের element হচ্ছে 7।

কেন মধক এই optimal? বুঝার জন্য একটা fact দেখা যাক। $|x - y|$ মানে আসলে নাম্বার লাইনে x এবং y এর মধ্যের distance। এইটা মাথায় রেখে উদাহরণের array এর x মান কমায় বা নাম্বার লাইন এ বাম দিক এ সরালে কি হয়, দেখা যাক।



এখানে x এর মান 1 কমালে 2 এবং 4 সাথে distance 1 করে কমে যায়। আবার 7, 9 এবং 12 এর সাথে distance 1 করে বের যায়। ফলে মোট যোগফল পরিবর্তন হয় = (-2 + 3) = +1। তাই x এর মান এক কমালে যোগফল ও বের যায়। একই ভাবে প্রমাণ করা যায়, x এর মান বাড়ালেও যোগফল বৃদ্ধি পায়। তাই সবচেয়ে কম যোগফল মধ্যক এই পাওয়া যায়।

যদি array সাইজ জোড় হতো, তাহলে দুইটা মধ্যক থাকত। এইক্ষেত্রে x এর optimal মান দুইটা মধ্যক এবং এদের মধ্যের সকল সংখ্যা।

Problem: তোমাকে 2D তে N টা পয়েন্ট দেওয়া হল। তোমাকে এমন একটা পয়েন্ট (x, y) বের করতে হবে যেন এই পয়েন্ট এর সাথে বাকি N টা পয়েন্টের manhattan distance এর যোগফল মিনিমাম হয়।

i.e দুইটা পয়েন্ট $(x1, y1)$ এবং $(x2, y2)$ এর manhattan distance = $|x1 - x2| + |y1 - y2|$