Binary Search

Farhan Ahmad

Date

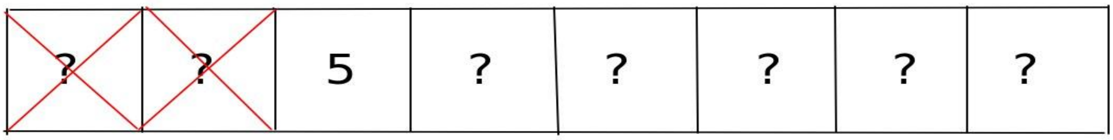
সূচীপত্র

1 Introduction: The Guessing Game!

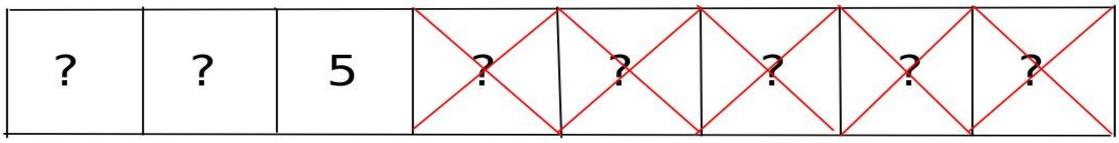
Binary Search শব্দটি শুনে একটু আজব লাগতে পারে, তাই এটা আপাতাত থাক! আমরা অন্য একটি প্রব্লেমে যাই। একটা খুব মজার গেম আছে, নাম তার দা গেসিং গেম! তো এখানে তোমার সামনে কম্পিউটার একটা N size এর array A লুকায় রাখবে, যেখানে সব সংখ্যাগগুলো sorted এবং distinct। এখন কম্পিউটার তোমাকে আরেকটা সংখ্যা K দিবে, কম্পিউটার নিশ্চিত করেছে যে সংখ্যা k array তেই আছে! এখন array এর কোন index এ আছে, এইটা বের করতে হবে আমাদের। অর্থাৎ, আমাদের এমন একটা index j guess করতে হবে যেন, A_j = k হয়! তো আমরা একের পর এক index guess করব, যদি ঠিক index আমরা guess করে ফেলি, তাহলে আমরা জিতে যাব! আর যদি না হয়, তাহলে ওখানে কোন সংখ্যাটি আছে, টা আমাদের কম্পিউটার বলবে! যেই কইটা guess করে আমারা বের করতে পারব, ঠিক ততো score পাব, যত কম score, ততো ভালো! এখন কিভাবে guess করলে worst posssible case এ আমাদের সব চেয়ে কম score হবে? তোমরা একটু চিন্তা করে দেখো তো!

আমরা randomly guess করা শুরু করতে পারি, কিন্তু এতে সমস্যা হল আমাদের ভাগ্য বেশি খারাপ হলে লাস্ট guess, অর্থাৎ N তম guess এ আমরা আমাদের K এর index খুঁজে পাব :(আমাদের অন্য কিছু একটা করতে হবে, এইটা কে বেশি efficient করতে! যদি আমরা index i guess করি, এবং কম্পিউটার আমাদের বলে দিবে যে, i index এ A_i আছে। একটা জিনিস কিন্তু খেয়াল করা যাচ্ছে যে, যদি A_i < k হয়, তাহলে অন্য কোন একটি index j $(j < i)$ এর A_j কখনই k এর সমান হতে পারবে না! কারণ, $A_j < k$ হবে! তাই আমাদের কিন্তু এরকম কোন j কে guess না করলেও হবে!

একটা উদাহরণে দেখা যাক, ধরি array এর length, $N = 8$ এবং $k = 12$। ধর, আমরা $3rd$ index কে guess করলাম, যেহেতু, A_3 = 5, তাই কম্পিউটার আমাদের বলবে 5। এখান থেকে আমরা বলতে পারি যে 1 বা 2 index এ k সমান কেউ নাই, কারণ $A_3 < k$ বা $5 < 12$, তাই $A_j = K$ এমন index j অবশ্যই 3 এর থেকে বেশি। যেহেতু, A_1 বা A_2 এর value এর মান 5 অপেক্ষা কম (Array টা sorted!)।



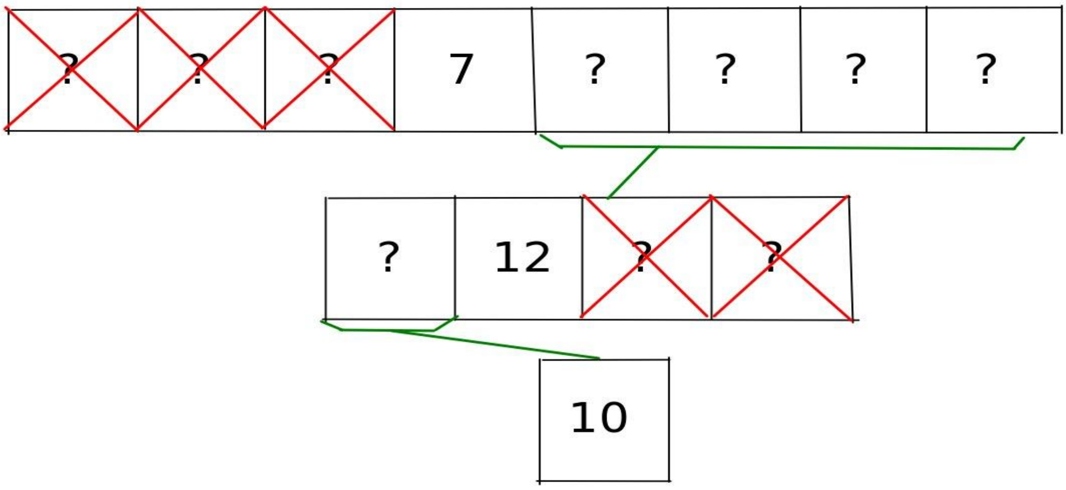
এখন একটু চিন্তা করে দেখা যাক, যদি এই ক্ষেত্রে $K < A_3$ হতো, তাহলে কি হতো? এবার ধরলাম $K = 2$, এখন ধর আবার $3rd$ index কেই guess করলাম। আবার পেলাম, $A_3 = 5$। এখন, $A_j = k = 2$, এমন index j অবশ্যই 3 এর বামদিক এ আছে, কারণ বাম দিকের সব element এর value 5 এর থেকে কম। অন্য দিকে ডানপাশে সব element এর value 5 এর থেকে বেশি, তাই যেহেতু $k < A_3$, ফলে এখানে k এর সমান কোন value থাকা এর সম্ভবনা নেই!



এখান থেকে আমরা একটা observation পাচ্ছি! যদি ঠিক index guess করি, তাহলে তো খুব এই ভালো! তা ছাড়াও আমরা array এর একটা বড় অংশ বাদ দিতে পারছি! তাহলে আমাদের এমন একটা index guess করেতে হবে, যেন ভুল হলেও array এর একটা বড় অংশ বাদ দিতে পারি। যদি আমাদের guess করা index এর value k এর থেকে বড় হয়, তাহলে ওই index এর ডানপাশ বাদ যাবে, টা ছাড়া বামপাশ। এখন worst case এ যেন সবচেয়ে বেশি বাদ যায়, এমন index choose করতে হবে! ভালোমতো খেয়াল করে দেখো, মাজখান এর index টা choose করলে worst case এ

$N/2$ টা element কে array থেকে বাদ দিতে পারব। এভাবে প্রতিবার বাদ দিতে থাকব যতক্ষণ না, আমরা ঠিক index কে guess করছি! এভাবে করলে প্রতি guess শেষ এ array এর সাইজ হাফ হয় যাচ্ছে, এভাবে কমতে থাকলে মাত্র $logN$ বার পর array size 1 হয়ে যাবে, তাই worst case এ $logN$ বার guess করা লাগবে!

একটা উদাহরম দিয়ে এই প্রব্লেম টা শেষ করা যাক! ধরি, $N = 8$, $K = 10$ এবং $A = [1, 3, 4, 7, 10, 12, 100, 2022]$। এখন এইটার ক্ষেত্রে আমারা কিভাবে বের করব, এইটা দেখব!



প্রথমে, array এর middle index কে guess করব। এই ক্ষেত্রে middle index = 4। Guess করার পর জানতে পারি, $A_4 = 7$, তাই আমরা 4 index এর বামপাশ কে বাদ দিয়ে দিতে পারি! সাথে $4th$ index কেও বাদ দিব, কারণ এইটা আমাদের k এর সমান না! এখন যেগুলো বাদ দেওয়ার, সব বাদ দিয়ে নতুন একটা array বানাই, যার size = 4। এই array এর middle index হচ্ছে 2 (কিন্তু, আসল array এর 6), এইটা কে guess করলে জানতে পারছি যে $A_6 = 12$। 12, k এর থেকে বেশি বলে ডানপাশ এ k থাকতে পারবে না! আবার আমারা তো জানতেই পারেছি যে new array এর $2nd$ index এ নাই! তাই থাকলে, $1st$ index এই আছে (আসল array এর $5th$ index)! এভাবেই আমরা k কে খুঁজে পেতে পারি!

এখন মনে হতে পারে, binary search কোথায় গেল? আসলে কোন কিছুকে এভাবে 2 ভাগে ভাগ করে search করার উপায় এই হচ্ছে binary search!

2 Problem: নকল মুদ্রা!

আরেকটা সমস্যা নিয়ে এখন কথা বলা যাক। ধর, তোমার কাছে N টা স্বর্ণের মুদ্রা রয়েছে। একদিন তুমি জানতে পারলে, N টা মুদ্রার মধ্যে একটা মুদ্রা নকল! এখন সব মুদ্রাগুলো দেখতে একই রকম, এমনকি নকল মুদ্রাটি ও! শুধুমাত্র নকল মুদ্রাটি এর ওজন একটু কম, আসল এর থেকে। এখন ওজন তুলনার একমাত্র উপায় হচ্ছে একটি খুব এই পুরাতন দাঁড়িপাল্লা। দাঁড়িপাল্লাতে তুমি দুইপাশে কিছু পরিমাণ মুদ্রা রাখবা, এরপর দাঁড়িপাল্লাটি দুই পাশের মুদ্রাসমূহের তুলনা দিবে। অর্থাৎ তুলনা থেকে জানা যাবে, কোন পাশের মুদ্রাসমূহের ওজন কম বা বেশি নাকি দুই পাশের মুদ্রাসমূহের ওজন একই! যেহেতু দাঁড়িপাল্লাটি অনেক পুরতান, তাই যত কম ব্যবহার ততই ভাল। কারণ বেশি ব্যবহার করলে ভেঙ্গে যেতে পারে! তোমাদের উদ্দেশ হচ্ছে, যত পারা যায়, কম ব্যবহার করে নকল মুদ্রাটি বের করা!

উপায় ১: $(N-1)*N/2$

আমরা সব মুদ্রার সাথে অন্য সব মুদ্রায়ের তুলনা করতে পারি, এভাবে করলে সর্বমোট $(N-1)*N/2$ বার দাঁড়িপাল্লা ব্যবহার করতে হচ্ছে। আর এতে যেই মুদ্রটি সব তুলনাতেই কম ওজনের হবে, ওই মুদ্রাটিই হল আমাদের নকল মুদ্রাটি!

উপায় ২: $N/2$

আগের মতো করলে অনেক অপ্রয়োজনীয় তুলনা করা হয়, যেগুলো না করলেও হত। যেমন, যদি দুইটা মুদ্রা তুলনার পর দেখি, দুইটাই একই ওজনের, তাহলে ওই দুইটা মুদ্রার কোনটিই নকল নয়, ওদের নিয়ে অন্য হিসাব করে কোন লাভ নাই। একইভাবে যদি কোন তুলনাতে কোন একটা মুদ্রায়ের ওজন কম হয়, তাহলে আসলে আর তুলনার দরকার নেই! কারন আমারা আমাদের নকল মুদ্রাটি পেয়ে গিয়েছি! এখন আমরা তাহলে জোড়ায় জোড়ায় সব মুদ্রা কে ভাগ করতে পারি (মনে করি, জোড় সংখ্যক মুদ্রা আছে)। এভাবে সর্বমোট $N/2$ টা জোড়া হবে, প্রতি জোড়ার দুইটি মুদ্রা দাঁড়িপাল্লা তে তুলব, এখন যদি এরা একই ওজনের হয়, তাহলে দুইটাই আসল মুদ্রা। আর যদি না হয়, তাহলে যেই মুদ্রাটির ওজন কম, সেইটাই নকল মুদ্রা! এভাবে করলে $N = 128$ এ মাত্র 64 বার দাঁড়িপাল্লা ব্যবহার করলেই হচ্ছে, যেখান আগের উপায়ে 8128 বার করতে হতো!

তোমরা চিন্তা করে দেখো তো, N বিজোড় হলে কি করা হতো!

উপায় ৩: $\log(N)$

$N = 128$ এর জন্য 64 বার মাপাও আসলে অনেক বেশি হয়ে যাচ্ছে! এখানে binary search ব্যবহার করে মাত্র 7 বার তুলনা করেই বের করা সম্ভব! এটা কিভাবে করা যায়, তোমরা একটু চিন্তা করে দেখো তো! যত সময় দিতে পারবা, তত ভালো! আচ্ছা যদি না পার, কোন সমস্যা নেই! তো আমরা দাঁড়িপাল্লার দুই পাশে একটি করে মুদ্রা না রেখে $N/2$ করে মুদ্রা রাখব। এখন মাপা এর পর যেই পাশের ওজন কম হবে, সেই পাশেই অবশই আমাদের নকল মুদ্রাটি রয়েছে! কারণ দুই পাশে সমান সংখ্যক মুদ্রা, যদি সব আসল মুদ্রা হতো, তাহলে এবার ওজনও সমান হতো। কিন্তু এখানে এক পাশের ওজন কম হয়েছে, কারণ ওই পাশে একটি নকল মুদ্রা আছে! আমরা জেনে যাচ্ছি যে কোন 64 টা মুদ্রা এর মধ্যে আমাদের নকল মুদ্রাটি রয়েছে! আবার এই মুদ্রাগুলোকে 32 ভাগে দুই ভাগ করে তুলনা করা পর জানতে পারব কোন 32 টা মুদ্রা এর মধ্যে আমাদের নকল মুদ্রাটি আছে। এভাবে 128 থেকে 64, 64 থেকে 32, 32 থেকে 16 ... 4 থেকে 2 এবং 2 থেক 1 টা মুদ্রাতে আসবে। আর এই মুদ্রাটি এই হল আমাদের নকল মুদ্রাটি!
এখানে N এর একটি বিশেষ মান নেওয়া তে প্রতি বার ভাগের সময় জোড় সংখ্যক মুদ্রা ছিল। কিন্তু যদি যদি জোড় সংখ্যক মুদ্রা না থাকত, তাহলে কি করা যেত? তোমরা একটু চিন্তা করে দেখো তো!

Bonus: $N = 128$ এর জন্য অন্য একটা উপায়ে মাত্র 5 বার দাঁড়িপাল্লা ব্যবহার করলেই হবে! নিজেরা চেষ্টা করে দেখতে পার, কিভাবে করা যায়!

3 কোড করে ফেলা যাক!

কোড করতে গেলে তো কোন প্রব্লেম লাগবে, এমনি এমনি তো কিছু কোড করা সম্ভব না! তো এই প্রব্লেম এর কথা চিন্তা করা যাক, ধর তোমার কাছে একটা sorted array আছে। তোমাকে বলতে হবে যে, কোন একটা সংখ্যা x এর থেকে বড় বা সমান সব ছেয়ে ছোট element এর মান কি, আর না থাকেলে বলতে হবে নাই! আমাদের কাছে array A থাকবে এবং একটা সংখ্যা x। মনে করি, array এর সাইজ N। এইটা কোড আগে তোমরা নিজেরা করার চেষ্টা করো!

আমরা binary search ব্যবহার করব এখন! তো new array এর দুইপাশের index কে l এবং r মনে করি। প্রথমে তো $l = 1$ এবং $r = N$। সাথে আরেকটা variable "ans" declare করি।

এখানে, middle element, m = $(l + r)/2$

এখানে দুইটা জিনিস হতে পারে,

১) $A_m < x$: এখানে অবশ্যই [l, m] range এ x এর থেকে বড় কোন element নাই! কারণ এই range এর সবচেয়ে বড় element A_m এবং আমরা দেখছি যে এই A_m হচ্ছে x এর চেয়ে ছোট! তাই, যদি x এর থেকে বড় কোন element থাকে, সেইটা [m+1, r] range এ থাকবে। তাই আমারা $l = m + 1$ করে দিব।

২) $A_m \geq x$: $ans = A_m$ করব, কারণ এইটা আমাদের জানা মতে এখন পর্যন্ত সবচেয়ে ছোট element যেইটা x এর থেকে বড়। আমরা জানি যে, [m, r] range এ সবাই $\geq A_m$, তাই এদের মধ্যে আর দেখে লাভ নাই, আমাদের দেখেতে হবে [l, m-1] range এ x এর থেকে আর বড় element আছে নাকি, থাকলে তখন আবার ans কে আপডেট করতে হবে। তাই আমরা এখন, $r = m - 1$ করব!

একটা sample code দেখা যাক!

```
void print_lowerbound(int x){
    int l = 1, r = N;
    int ans = -1; //A marker, in case we don't have a lowerbound

    while(l <= r){
        int m = (l+r)/2;
        if(A[m] < x){
            l = m + 1;
        }else{
            ans = A[m];
            r = m - 1;
        }
    }
    if(ans == -1) cout << "Lowerbound nai :( \n";
    else{
        cout << ans << "\n";
    }
    return;
}
```

4 Binary Search on Answer!

জিনিসটা কি এটা এমনি বুঝার থেকে একটা প্রব্লেম দিয়ে বুঝা অনেক সহজ, তাই এখানে CSES এর প্রব্লেম 1620: Factory Machines দেখানো হলো।

Problem: একটা কারখানাতে N টা candy machine আছে। ith candy machine এ candy বানাতে A_i সময় লাগে। তুমি k টা candy বানাতে চাও। একটা candy machine এ যত ইচ্ছা candy বানাতে পারবা এবং একই সাথে অনেক গুলো machine একসাথে চলতে পারবে। সর্বনিম্ন কতক্ষণ লাগবে, এইটা বের করতে হবে।

Solution: প্রতিবারের মতো এবারও বলব নিজে থেকে চেষ্টা করার! এখানে আমরা Answer এর ওপর binary search করব। আমরা দেখব, t time এ কইটা candy বানানো যায়। যদি দেখি, t সময়ে কম k এর থেকে কম candy বানানো হচ্ছে, তাহলে আরও বেশি টাইম লাগবে, টা ছাড়া দেখব আর কম টাইমে করা যায় নাকি। এভাবে binary search করেই আমরা টাইম বের করে ফেলতে পারব, কঠিন কোন অংক করা ছাড়াই! Code টা দেখে ফেলা যাক এখন।

```cpp
//A huge number, which apperantly would be considered as infinity
const ll INF = numeric_limits<ll>::max()-1;

long long cal(int t){ // calculates how many candies would be made in t time!
  long long ans = 0;
  for(int i = 1; i <= n; i++){
      ans += t / A[i];
      if(ans >= 1e18) return INF; //avoiding overflow
  }
  return ans;
}

void solve(int x){
    long long l = 0; // Could ask to make 0 candies
    long long r = 1e18; // Depends on the constraints
    //Let's assume it would take at most 1e18 time to make any amount of Candy

    long long ans = 1e18; // Setting up the worst possible time it could take

    while(l <= r){
        long long m = (l + r)/2;
        if(cal(m) < x){
            l = m + 1; //only [m+1, r] range is relevant
        }else{
            ans = m;
            //Only [l, m] is relevant, but we have the answer for m, so we
            //need to check only [l, m-1] range
            r = m - 1;
        }
    }
    cout << ans << '\n';
    return;
}
```

এইভাবে binary search করা অনেক famous একটা trick। অনেক প্রব্লেমেই সাধারণভাবে হিসাব করা সম্ভব হয় না, তাই answer এর ওপর binary search করলে অনেক সহজ হয়ে যায়। তাই সব সময়, এই জিনিস টার কথা মাথায় রেখে প্রব্লেম solve করা দরকার। কবে কাজে লেগে যায়, বলাও যায় না!

5 STL in-built functions

STL এর মধ্যে set, multiset এবং vector এ কিছু in-built, binary search করার জন্য functions আছে। এদের মধ্যে এখানে vector এবং set নিয়ে কথা বলা হবে। multiset নিয়ে কিছু বলা হল না কারণ, set এর সাথে অনেক মিল আছে। set এর ধারনাই multiset এ ব্যবহার করা যায়।

5.1 Vector

vector এর জন্য mainly দুইটা binary search এর জন্য in-built function আছে। এদের কাজ দেওয়া হল:

5.1.1 Lower Bound

x এর Lower Bound কোন একটা vector এর মানে বুঝায় যে, vector এর smallest element p, যেন $p \geq x$ হয়। এইটা তো আমারা খুব সহজেই binary search করে বের করতে পারব, কিন্তু আমাদের কাজ সহজ করতে vector এর in-built function এই আছে! function টি হল এমন,

```
  Iterator lower_bound (Iterator first, Iterator last, const val)
```

function টি কাজ করতে হলে কিন্তু অবশ্যই vector টা sorted রাখতে হবে! কিছু উদাহরণ দেখা যাক,

```
vector < int > v = {1, 4, 8, 12, 77, 333};

auto it = lower_bound(v.begin(), v.end(), 8);
if(it == v.end()) cout << "lowerbound nai :( \n";
else cout << *it << '\n'; // prints 8

auto it2 = lower_bound(v.begin() + 3, v.end(), 2); // taking the range [3, 6);

if(it2 == v.end()) cout << "lowerbound nai :( \n";
else cout << *it << '\n'; // prints 12
```

যদি আমরা বের করতে চাই, lowerbound কোন index এ আছে, তাহলে আমারা এমন করতে পারি!

```
  vector < int > v = {1, 2, 3, 6, 10};
  int index = lower_bound(v.begin(), v.end(), 4) - v.begin();
  if(index == v.size()) cout << "lowerbound nai :( \n";
  else cout << index << '\n';
```

5.1.2 Upper Bound

x এর Upper Bound কোন একটা vector এর মানে বুঝায় যে, vector এর smallest element p, যেন $p > x$ হয়। এইটা তো আমারা খুব সহজেই binary search করে বের করতে পারব, কিন্তু আমাদের কাজ সহজ করতে vector এর in-built function এই আছে! function টি হল এমন,

```
    Iterator upper_bound (Iterator first, Iterator last, const val)
```

function টি কাজ করতে হলে কিন্তু অবশ্যই vector টা sorted রাখতে হবে! কিছু উদাহরণ দেখা যাক,

```
  vector < int > v = {1, 4, 8, 12, 77, 333};

  auto it = upper_bound(v.begin(), v.end(), 8);
  if(it == v.end()) cout << "upperbound nai :( \n";
  else cout << *it << '\n'; // prints 12

  auto it2 = upper_bound(v.begin() + 3, v.end(), 2); // taking the range [3, 6);

  if(it2 == v.end()) cout << "upperbound nai :( \n";
  else cout << *it << '\n'; // prints 12
```

যদি আমরা বের করতে চাই, lowerbound কোন index এ আছে, তাহলে আমারা এমন করতে পারি!

```
    vector < int > v = {1, 2, 3, 6, 10};
    int index = upper_bound(v.begin(), v.end(), 4) - v.begin();
    if(index == v.size()) cout << "upperbound nai :( \n";
    else cout << index << '\n';
```

5.2 Set

Set এর অনেক binary search এর জন্য in-built function আছে। টার মধ্যে দুইটা নিয়ে এখানে কথা বলা হবে। set আর vector মধ্যে মুল পার্থক্য হচ্ছে set এ index access যাবে না!

5.2.1 Lower Bound

x এর lower bound set s এ p হবে, যদি $p \in s$, $p \geq x$ এবং অন্য কোন $p' < p$ নেই, যেইটা প্রথম দুইটা শর্ত satisy করে! Set এর in-built function টি হল,

```
    set_name.lower_bound(key)
```

set এর সব sorted থাকে বলে, আলাদা করে কিছু করতে হয় না, এখান এ কিছু উদাহরণ দেওয়া হল:

```
    set < int > s = {1, 2, 4, 6, 8 , 10, 12};
    auto it = s.lower_bound(6);
    cout << *it << '\n'; // prints 6
    it--;
    cout << *it << '\n'; // prints 4
```

5.2.2 Upper Bound

x এর upper bound set s এ p হবে, যদি $p \in s$, $p > x$ এবং অন্য কোন $p' < p$ নেই, যেইটা প্রথম দুইটা শর্ত satisy করে! Set এর in-built function টি হল,

```
    set_name.upper_bound(key)
```

set এর সব sorted থাকে বলে, আলাদা করে কিছু করতে হয় না, এখান এ কিছু উদাহরণ দেওয়া হল:

```
    set < int > s = {1, 2, 4, 6, 8 , 10, 12};
    auto it = s.upper_bound(6);
    cout << *it << '\n'; // prints 6
    it--;
    cout << *it << '\n'; // prints 4
```