তুমি নিশ্চয়ই লক্ষ্য করেছো ডিকশনারিতে লাখ লাখ শব্দ থাকলেও প্রয়োজনীয় শব্দটা খুজে পেতে কখনো খুব বেশি সময় লাগে না। এটার কারণ হলো শব্দগুলো অক্ষর অনুযায়ী সাজানো থাকে। তাই তুমি যদি dynamite শব্দটা ডিকশনারিতে খোজার চেষ্টা করো এবং এলোমেলোভাবে কোনো একটা পাতা খুলে kite শব্দটা খুজে পাও তাহলে তুমি নিশ্চিত হয়ে যেতে পারো যে তুমি যে শব্দটা খোজার চেষ্টা করছো সেটা বাম দিকে কোথাও আছে। আবার যদি তুমি dear শব্দটা খুজে পাও তখন তুমি আর বাম দিকের পাতাগুলোয় খোজার চেষ্টা করবে না। এভাবে অল্প সময়ের মধ্যে ডিকশনারিতে যেকোনো শব্দ খুজে পাওয়া যায়।

বাইনারি সার্চ হলো অনেকটা এরকম একটা পদ্ধতি যেটা ব্যবহার করে একটা অ্যারে থেকে কোনো একটা তথ্য খুজে বের করা যায়।

ধরো তোমার কাছে একটা অ্যারেতে অনেকগুলো সংখ্যা আছে এরকম:

100,2,10,50,20,500,100,150,200,1000,100100,2,10,50,20,500,100,150,200,1000,100

সংখ্যাগুলো এলোমেলোভাবে সাজানো থাকায় এখান থেকে একটা নির্দিষ্ট সংখ্যা খুজে পাওয়া সহজ না। তুমি যদি অ্যারে থেকে ৫০০ সংখ্যাটা খুজতে চাও তাহলে সবগুলো ইনডেক্সই পরীক্ষা করে দেখতে হবে, এটাকে বলা হয় লিনিয়ার সার্চ। অ্যারের আকার যদি nn হয় তাহলে লিনিয়ার সার্চের টাইম কমপ্লেক্সিটি হলো O(n)O(n)।
কিন্তু যদি সংখ্যাগুলো নিচের মত সাজানো থাকে তাহলে খুজে পাওয়া অনেক সহজ হয়ে যায়:

2,10,20,50,100,100,100,150,200,500,10002,10,20,50,100,100,100,150,200,500,1000

এখন যদি ১৫০ সংখ্যাটা খুজে বের করতে হয় তাহলে আমরা প্রথমে ঠিক মাঝের ইনডেক্সটা পরীক্ষা করবো। প্রথম ইনডেক্স ০ এবং শেষের ইনডেক্স ১০ হলে মাঝের ইনডেক্সটা হলো (০+১০)/২ বা ৫ তম ইনডেক্স।

*$2, 10, 20, 50, 100, **100**, 100, 150, 200, 500, 1000*
মাঝের ইনডেক্সের সংখ্যাটা ১০০ যা ১৫০ এর থেকে ছোট, আমরা জেনে গেলাম যে সংখ্যাটা ডান পাশে কোথাও আছে, বামের অংশ আমাদের আর দরকার নাই।

*100, 150, **200**, 500, 1000*
এখন বাকি অংশটা নিয়ে আবারো একই কাজ করবো। এবার মাঝের ইনডেক্সের সংখ্যাটা হলো ২০০ যেটা ১৫০ এর থেকে বড়। তারমানে ডানের অংশটা আমরা ফেলে দিতে পারি।

***100**, 150*
এবার মাঝের সংখ্যাটা হলো ১০০ যা ১৫০ এর থেকে ছোট। আবারো বামের অংশ ফেলে দিবো, থাকবে শুধু:

150

এবার মাঝের ইনডেক্সের সংখ্যাটা হলো ১৫০। তারমানে কাঙ্খিত সংখ্যাটাকে খুজে পাওয়া গেছে।

বাইনারি সার্চে প্রতিবার অ্যারের ঠিক অর্ধেক অংশ আমরা বাতিল করে দিচ্ছি এবং বাকি অর্ধেক অংশে খুজছি। একটা সংখ্যা nn কে সর্বোচ্চ কয়বার ২ দিয়ে ভাগ করা যায় যতক্ষণ না সংখ্যাটা ১ হয়ে যাচ্ছে? উত্তর হলো $log_2nlog2n$। তাই বাইনারি সার্চে সর্বোচ্চ $log_2(n)log2(n)$ সংখ্যক ধাপের পর আমরা দরকারি সংখ্যাটা খুজে পাবো, কমপ্লেক্সিটি $O(log_2n)O(log2n)$।
কিন্তু একটা সমস্যা হলো বাইনারি সার্চ করার আগে অবশ্যই সংখ্যাগুলোকে ছোট থেকে বড় বা বড় থেকে ছোট সাজিয়ে নিতে হবে, এই প্রক্রিয়াটাকে বলা হয় সর্টিং। তুমি যতই চেষ্টা কর না কেন একটা অ্যারে $O(n\times log_2n)O(n\times log2n)$ এর কম কমপ্লেক্সিটি সর্ট করতে পারবে না।* তাহলে কেও বলতে পারে যে সর্ট করতে যে সময় লাগছে তার থেকে অনেক কম সময়ে লিনিয়ার সার্চ করেই আমরা সংখ্যাটা খুজে পেতে পারি, বাইনারি সার্চ কেন করবো? তোমার যদি একটা অ্যারেতে মাত্র ১ বার সার্চ করা দরকার হয় তাহলে কষ্ট করে সর্ট করে বাইনারি সার্চ করার থেকে লিনিয়ার সার্চ করা অনেক ভালো। কিন্তু যদি এমন হয় একটা অ্যারেতে অনেকবার সার্চ করা দরকার হবে? যেমন একটা ডিকশনারিতে বিভিন্ন সময় বিভিন্ন শব্দ খোজা দরকার হয়, সেক্ষেত্রে সর্ট করে রাখাই বুদ্ধিমানের কাজ।
নিচের পাইথন কোডে বাইনারি সার্চের ইমপ্লিমেন্টেশন দেখানো হয়েছে:

```
def search(array, key):
    begin=0
    end=len(array)-1
    index=None
    while begin<=end:
        mid=(begin+end)/2
        if key == array[mid]:
            index=mid #The value is found, save the index.
            break
        elif key > array[mid]: begin=mid+1 #Search the right portion
        elif key < array[mid]: end=mid-1 #Search the left portion
    return index #If the number is not found, index will contain None.


info=[100,2,10,50,20,500,100,150,200,1000,100]
info=sorted(info)
while True:
    key = int(raw_input())
    print search(info,key)
```

উপরের কোডে যদি আমরা ১০০ ইনপুট দেই তাহলে ৫ রিটার্ন করবে, কারণ অ্যারেটা সর্ট করার পর ৫ নম্বর ইনডেক্সে ১০০ আছে। কিন্তু সর্টেড অ্যারের দিকে তাকালে আমরা দেখছি যে ৪,৫,৬ এই সবগুলো ইনডেক্সেই ১০০ আছে। আমরা যদি চাই কোনো সংখ্যা একাধিকবার থাকলে সবথেকে বামের ইনডেক্সটা রিটার্ন করতে, তাহলে কোডটা কিভাবে পরিবর্তন করতে হবে? সেক্ষেত্রে বাইনারি সার্চ ফাংশনটা হবে এরকম:

```
def search(array, key):
    begin=0
    end=len(array)-1
    index=None
    while begin<=end:
        mid=(begin+end)/2
        if key == array[mid]:
            index=mid #One occurance of the value is found, save the index
            end=mid-1 #Continue searching the left portion after one occurrence is found
        elif key > array[mid]: begin=mid+1
        elif key < array[mid]: end=mid-1
    return index #Index will contain None if the value is not found
```

শুধুমাত্র ১টা মাত্র লাইন পরিবর্তন করা হয়েছে, এবার কোনো সংখ্যা খুজে পাবার পর খোজা বন্ধ না করে বামের বাকি অংশটুকুতে খুজতে থাকবো।

লোয়ার বাউন্ড: তোমাকে একটা সর্টেড অ্যারে দেয়া আছে। তুমি নতুন একটা সংখ্যা XX সেই অ্যারেতে ঢুকাতে চাও। লোয়ার বাউন্ড হলো সবথেকে বামের ইনডেক্স যেখানে তুমি সংখ্যাটা ঢুকিয়ে বাকি সংখ্যাগুলোকে একঘর ডানে সরালে অ্যারেটা তখনো সর্টেড থাকবে। মনে করো অ্যারেটা এরকম:

10 20 20 30 30 40 50

তাহলে 20 এর লোয়ার বাউন্ড হলো ইনডেক্স ১ (০ বেসড ইনডেক্স), কারণ ১ নম্বর ইনডেক্সে তুমি ২০ সংখ্যাটাকে বসিয়ে বাকি সংখ্যাগুলোকে ডানে সরিয়ে দেয়ার পরেও অ্যারেটা সর্টেড থেকে যাচ্ছে। ঠিক সেরকম ২৫ এর জন্য লোয়ার বাউন্ড হলো ইনডেক্স ৩।

সহজ কথায় সবথেকে বামের যে ইনডেক্সে x **এর সমান বা বড়** কোনো সংখ্যা আছে সেই ইনডেক্সটাই হলো লোয়ার বাউন্ড। XX যদি অ্যারের প্রতিটা সংখ্যার থেকে বড় হয় তাহলে সর্বশেষ ইনডেক্সের পরবর্তী ইনডেক্সটা অর্থাৎ nn তম ইনডেক্সটা হলো লোয়ার বাউন্ড যেখানে nn হলো অ্যারের আকার।

নিচের কোডে বাইনারি সার্চ করে লোয়ার ইনডেক্স খুজে বের করা হয়েছে, তারপর সেই ইনডেক্সে নতুন সংখ্যাটা বসিয়ে দেয়া হয়েছে।

```
def searchLowerBound(array, key):
        begin=0
        end=len(array)-1
        index=-1
        while begin<=end:
                mid=(begin+end)/2
                if key == array[mid]:
                        index=mid
```

```
                        end=mid-1
                elif key > array[mid]: begin=mid+1
                elif key < array[mid]: end=mid-1
        return begin



info=[100,2,10,50,20,500,100,150,200,1000,100]
info=sorted(info)
print info
while True:
        X = int(raw_input())
        lowerbound=searchLowerBound(info,X)
        info.insert(lowerbound,X)
        print "New array: ",info
```

(অ্যারেতে এভাবে সংখ্যা ইনসার্ট করতে O(n)O(n) সময় লাগে। অ্যারের জায়গায় লিংক লিস্ট ব্যবহার করে O(1)O(1) সময়ে ইনসার্ট করা যায় কিন্তু সেক্ষেত্রে বাইনারি সার্চ করা সম্ভব না কারণ লিংক লিস্টে যেকোনো ইনডেক্স রেন্ডম এক্সেস করা যায় না, পয়েন্টার ধরে আগাতে হয়। একটা উপায় হতে পারে বাইনারি সার্চ ট্রি আকারে তথ্যগুলো সাজিয়ে রাখা, সেটা আমরা এই লেখায় দেখবো না, তুমি চাইলে নিজে শিখে নিতে পারো।)

আপার বাউন্ড: আপার বাউন্ড হলো সব থেকে ডানের ইনডেক্স যেখানে তুমি নতুন সংখ্যাটা ঢুকালে অ্যারেটা সর্টেড থাকবে। তারমানে সবথেকে বামের যে ইনডেক্সে **x এর বড়** কোনো সংখ্যা আছে সেই ইনডেক্সটাই হলো আপার বাউন্ড। উপরের উদাহরণটায় ২০ এর আপার বাউন্ড হলো ইনডেক্স ৩। আপার বাউন্ড বের করার কোডটা তোমার বাড়ির কাজ হিসাবে থাকলো!

প্রথম পর্ব এখানেই শেষ। পরের পর্বে বাইসেকশন মেথড নিয়ে আলোচনা করবো।

প্র্যাকটিসের জন্য প্রবলেম:

https://uva.onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=1552http://www.lightoj.com/volume_showproblem.php?problem=1088

হ্যাপি কোডিং!

* কাউন্টিং সর্ট বা রেডিক্স সর্ট ব্যবহার করে O(n)O(n) এ সর্টিং করা যায় তবে সেগুলো কাজ করে বিশেষ ধরণের কিছু ইনপুটের ক্ষেত্রে। ইনপুট কি ধরণের হবে সেটার উপর কোনো শর্ত না থাকলে O(log$_2$n)O(log2n) এর কমে সর্টিং করা সম্ভব না সেটা গাণিতিকভাবে প্রমাণ করা যায়।