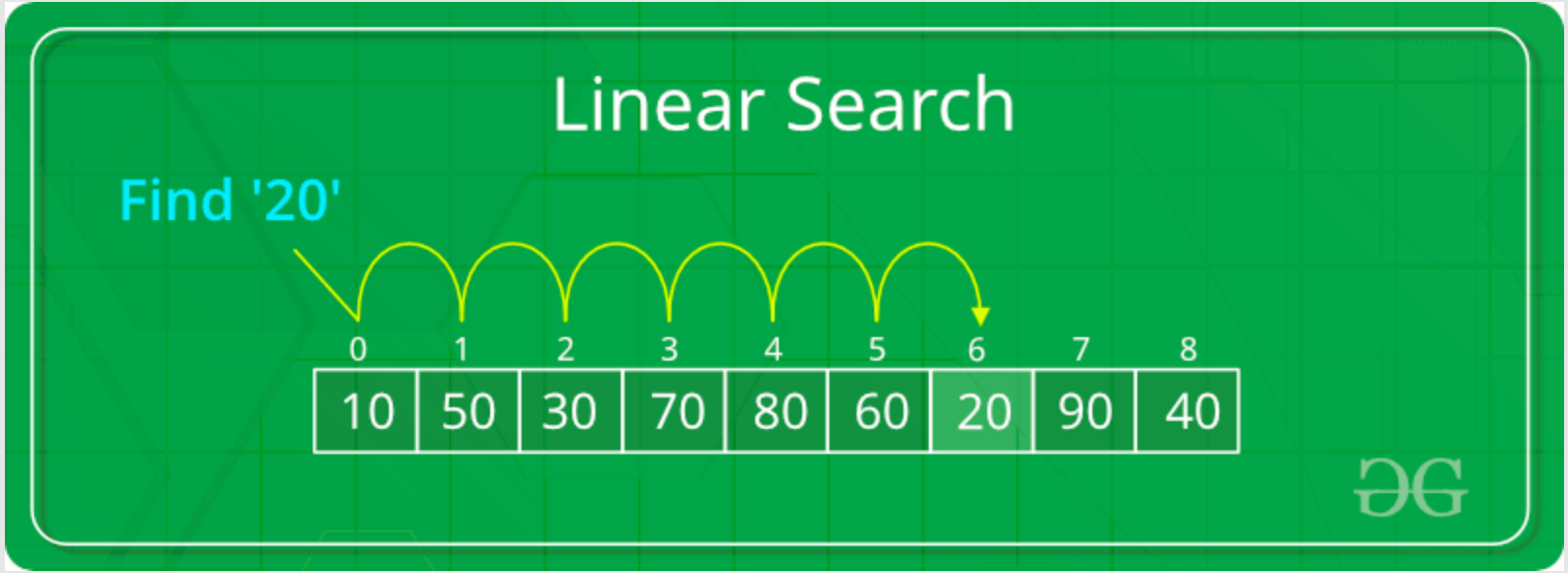
Binary search


Linear Search
Find '20'
0 1 2 3 4 5 6 7 8
10 50 30 70 80 60 20 90 40

```python
a = list(map(int, input().split()))

need_to_find = int(input())

for elem in a:
    if elem == need_to_find:
        print('Found')
        exit(0)
print('Not Found')
```

#stdin #stdout 0.03s 9732KB

stdin

```
1 4 3 2 5 1
2
```

stdout

```
Found
```

```python
a = list(map(int, input().split()))

need_to_find = int(input())

for elem in a:
    if elem == need_to_find:
        print('Found')
        exit(0)
print('Not Found')
```

#stdin #stdout 0.03s 9676KB

stdin

```
1 4 3 2 5 1
0
```

stdout

```
Not Found
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> a = {1, 4, 3, 2, 5, 1};
    int need_to_find = 2;
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == need_to_find) {
            cout << "Found";
            return 0;
        }
    }
    cout << "Not Found";
}
```

din #stdout 0.01s 5512KB

stdin

```
Standard input is empty
```

stdout

```
Found
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> a = {1, 4, 3, 2, 5, 1};
    int need_to_find = 0;
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == need_to_find) {
            cout << "Found";
            return 0;
        }
    }
    cout << "Not Found";
}
```

 #stdin #stdout 0.01s 5504KB

stdin

```
Standard input is empty
```

stdout

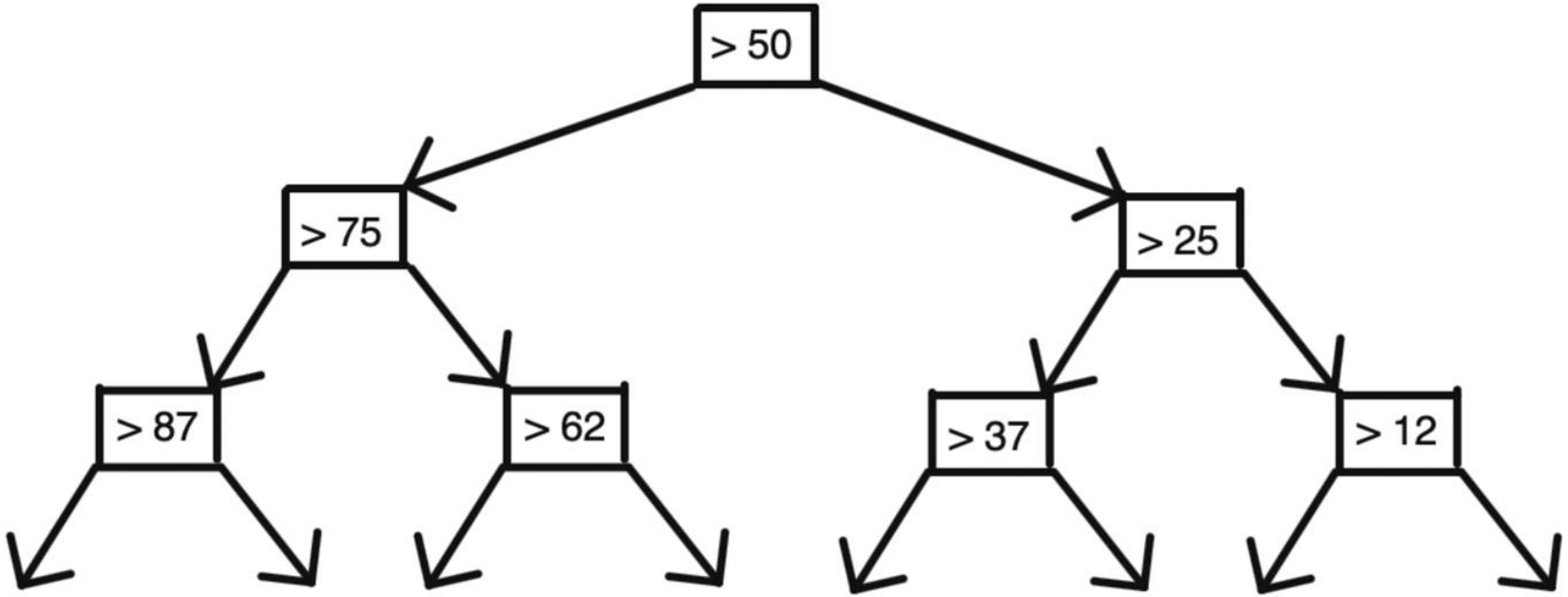
```
Not Found
```

Why O(n)

- We check each element in the worst case
- Amount of elements = |array| = n

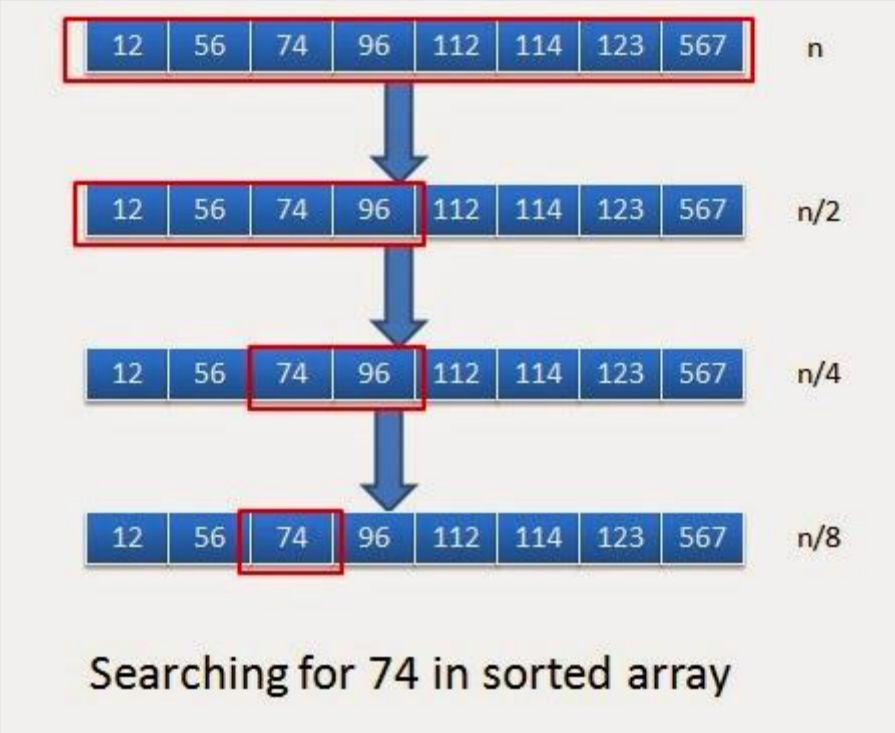
I guessed the number

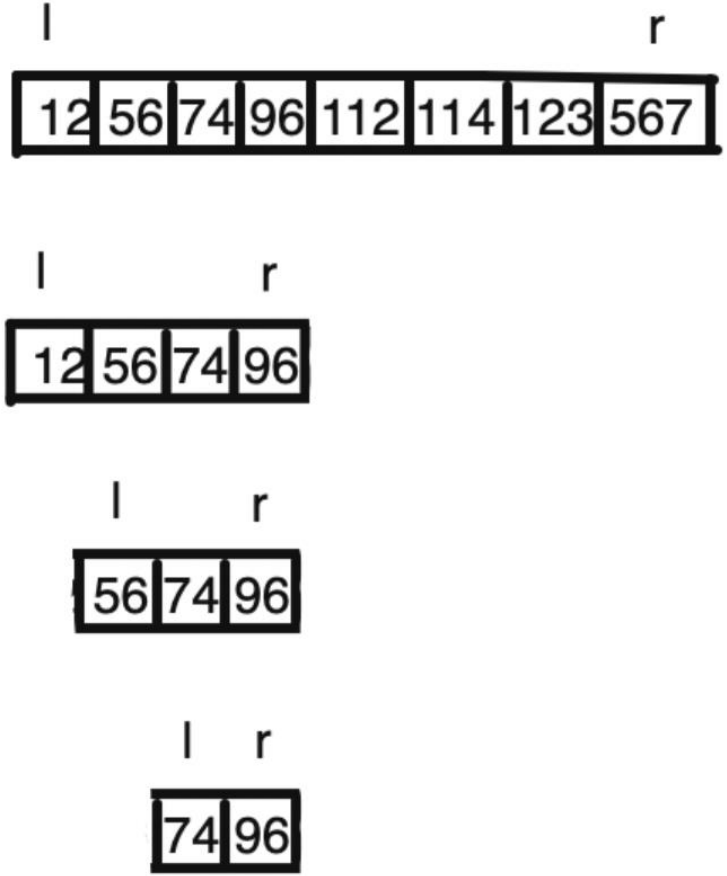
- I guessed the number between 1 and 100
- You can only ask the "> x" query
- You have to guess my number in the minimum number of attempts

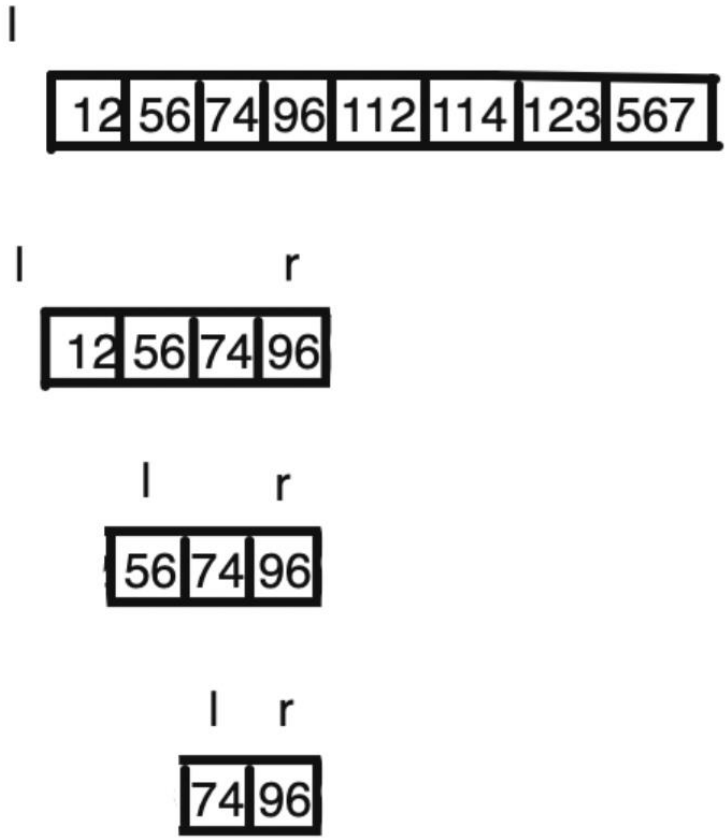

> 50
> 75
> 25
> 87
> 62
> 37
> 12

Binary search in array

- We have sorted array
- We need to find a number x in array


12 56 74 96 112 114 123 567
n
12 56 74 96 112 114 123 567
n/2
12 56 74 96 112 114 123 567
n/4
12 56 74 96 112 114 123 567
n/8
Searching for 74 in sorted array


l
r
12 56 74 96 112 114 123 567
l
r
12 56 74 96
l
r
56 74 96
l
r
74 96


l
r
12 56 74 96 112 114 123 567
l
r
12 56 74 96
l
r
56 74 96
l
r
74 96

Asymptotic

- Each time we divide our array in two equal parts.
- We need log(n) operations to get array of size 1.
- Every operation is simple(comparison and change)
- O(log(n))

```python
a = list(map(int, input().split()))

need_to_find = int(input())

l, r = -1, 8
while l < r - 1:
    mid = (l + r) // 2
    if a[mid] <= need_to_find:
        l = mid
    else:
        r = mid
print(l, r)
print(a[l] if l >= 0 else -1e9)
```

#stdin #stdout 0.03s 9696KB

stdin

```
1 2 3 4 5 6 7 8
0
```

stdout

```
-1 0
-1000000000.0
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> a = {1, 2, 3, 4, 5, 6, 7, 8};
    int l = -1;
    int r = 8;
    int need_to_find = 0;
    while (l < r - 1) {
        int mid = (l + r) / 2;
        if (a[mid] <= need_to_find) {
            l = mid;
        }
        else {
            r = mid;
        }
    }
    cout << l << " " << r << "\n";
    if (l >= 0) cout << a[l];
    else cout << -1e9;
}
```

#stdin #stdout 0.01s 5456KB

stdin

```
Standard input is empty
```

stdout

```
-1 0
-1e+09
```

```python
a = list(map(int, input().split()))

need_to_find = int(input())

l, r = -1, 8
while l < r - 1:
    mid = (l + r) // 2
    if a[mid] <= need_to_find:
        l = mid
    else:
        r = mid
print(l, r)
print(a[l] if l >= 0 else -1e9)
```

#stdin #stdout 0.03s 9688KB

stdin

```
1 2 3 4 5 6 7 8
3
```

stdout

```
2 3
3
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> a = {1, 2, 3, 4, 5, 6, 7, 8};
    int l = -1;
    int r = 8;
    int need_to_find = 3;
    while (l < r - 1) {
        int mid = (l + r) / 2;
        if (a[mid] <= need_to_find) {
            l = mid;
        }
        else {
            r = mid;
        }
    }
    cout << l << " " << r << "\n";
    if (l >= 0) cout << a[l];
    else cout << -1e9;
}
```

stdin #stdout 0.01s 5380KB

stdin

```
Standard input is empty
```

stdout

```
2 3
3
```

```python
a = list(map(int, input().split()))

need_to_find = int(input())

l, r = -1, 8
while l < r - 1:
    mid = (l + r) // 2
    if a[mid] <= need_to_find:
        l = mid
    else:
        r = mid
print(l, r)
print(a[l] if l >= 0 else -1e9)
```

#stdin #stdout 0.03s 9736KB

stdin

```
1 2 3 4 5 6 7 8
9
```

stdout

```
7 8
8
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> a = {1, 2, 3, 4, 5, 6, 7, 8};
    int l = -1;
    int r = 8;
    int need_to_find = 9;
    while (l < r - 1) {
        int mid = (l + r) / 2;
        if (a[mid] <= need_to_find) {
            l = mid;
        }
        else {
            r = mid;
        }
    }
    cout << l << " " << r << "\n";
    if (l >= 0) cout << a[l];
    else cout << -1e9;
}
```

#stdin #stdout 0.01s 5360KB

stdin

```
Standard input is empty
```

stdout

```
7 8
8
```

Binary search?

- arrays
- functions
- answer

```python
while l < r - 1:
    mid = (l + r) // 2
    if good(mid):
        l = mid
    else:
        r = mid
```

```java
while (l < r - 1) {
    int mid = (l + r) / 2;
    if (good(mid)) {
        l = mid;
    }
    else {
        r = mid;
    }
}
```

Floats

```
1. print(0.1 * 3)
```

#stdin #stdout 0.04s 9716KB

stdin

```
Standard input is empty
```

stdout

```
0.30000000000000004
```

Floats

```cpp
#include <iomanip>
#include <iostream>

using namespace std;

int main() {
    cout << setprecision(20) << 0.1 * 3;
}
```

#stdin #stdout 0.01s 5440KB

stdin

```
Standard input is empty
```

stdout

```
0.30000000000000004441
```

Floats

```
1.  EPS = 10**(-6)
2.
3.  print((0.1 * 3) == 0.3) # wrong
4.  print((0.1 * 3 - 0.3) <= EPS) # good
```

#stdin #stdout 0.04s 9560KB

stdin

```
Standard input is empty
```

stdout

```
False
True
```

Floats

```cpp
#include <iomanip>
#include <iostream>

using namespace std;
const float EPS = 0.0000001;

int main() {
    cout << (0.1 * 3 == 0.3) << " " << ((0.1 * 3 - 0.3) < EPS);
}
```

#stdin #stdout 0.01s 5512KB

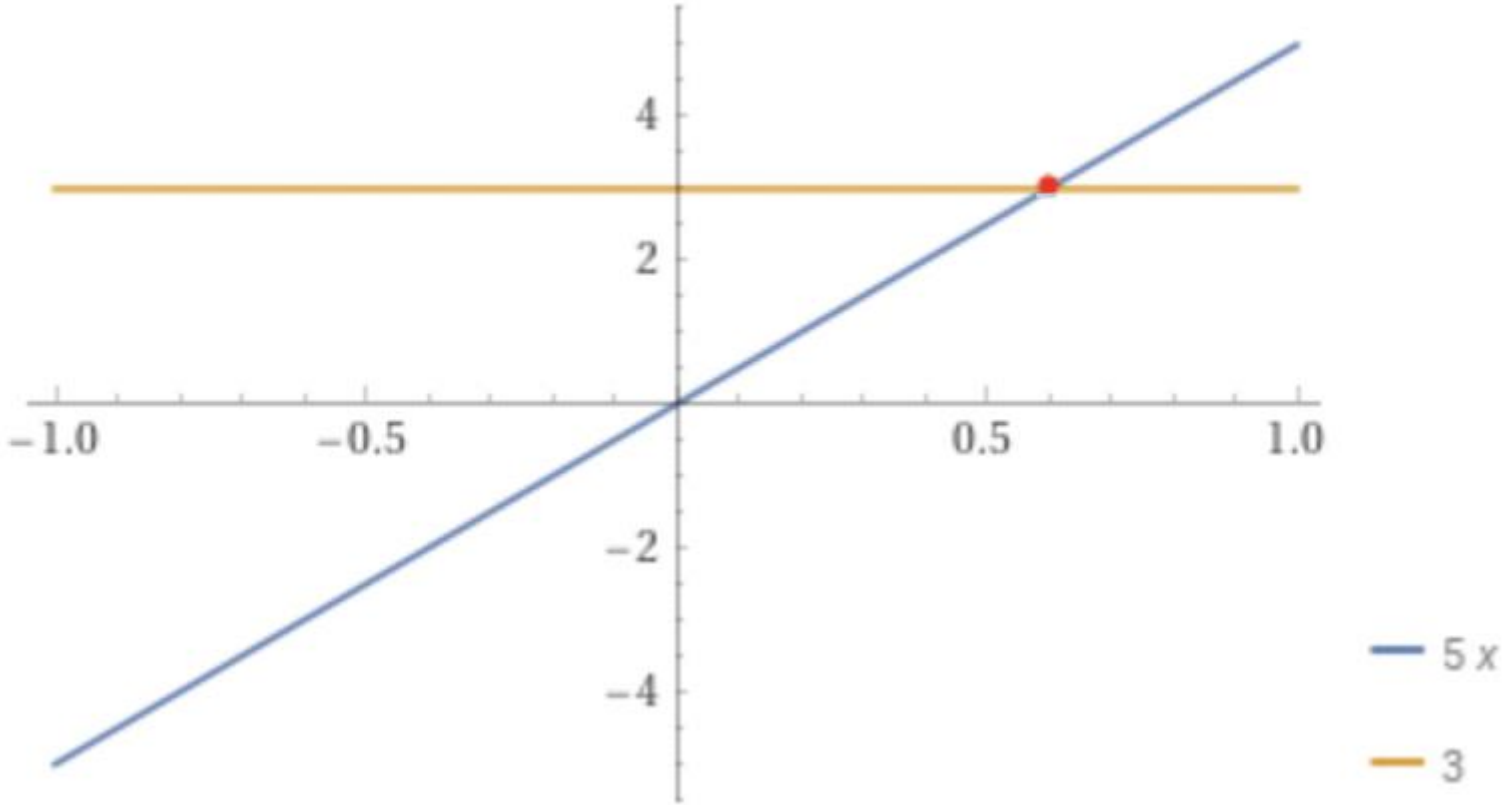
stdin

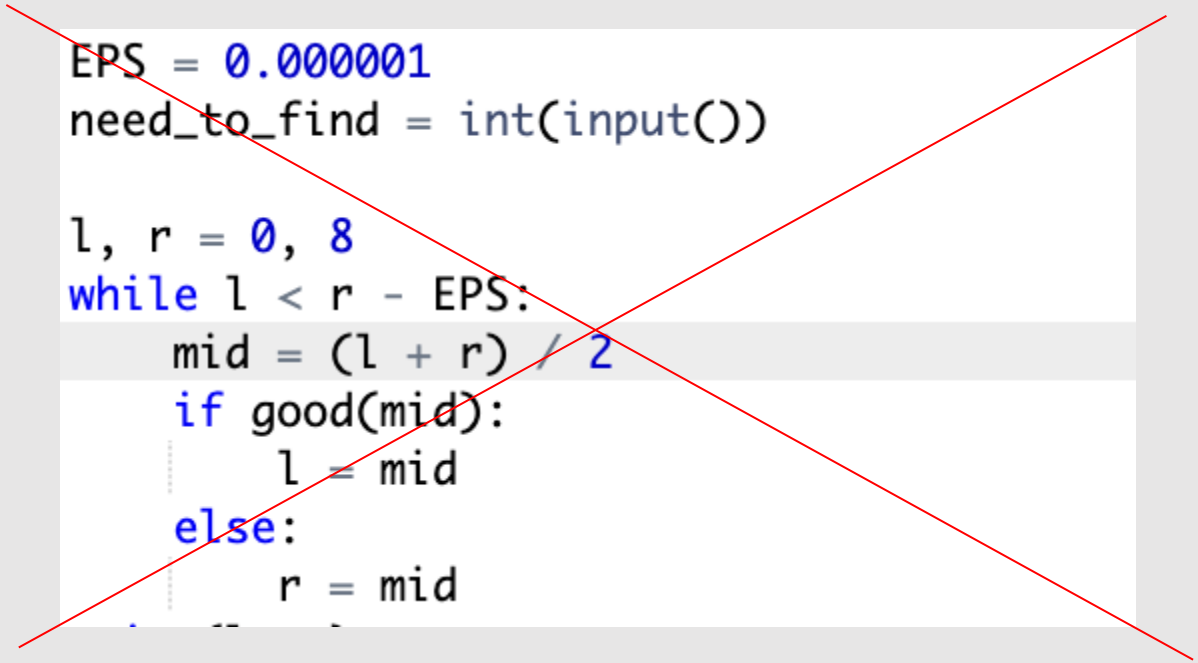
```
Standard input is empty
```

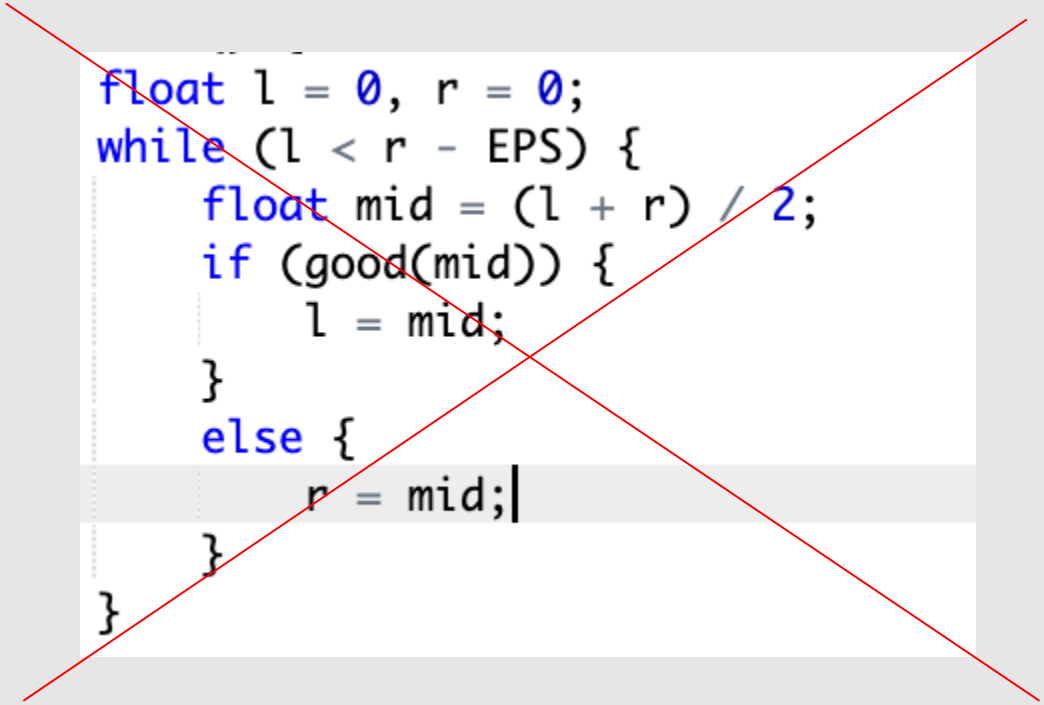
stdout

```
0 1
```

Plot




EPS = 0.000001
need_to_find = int(input())
l, r = 0, 8
while l < r - EPS:
mid = (l + r) / 2
if good(mid):
l = mid
else:
r = mid


float l = 0, r = 0;
while (l < r - EPS) {
float mid = (l + r) / 2;
if (good(mid)) {
l = mid;
}
else {
r = mid;
}
}

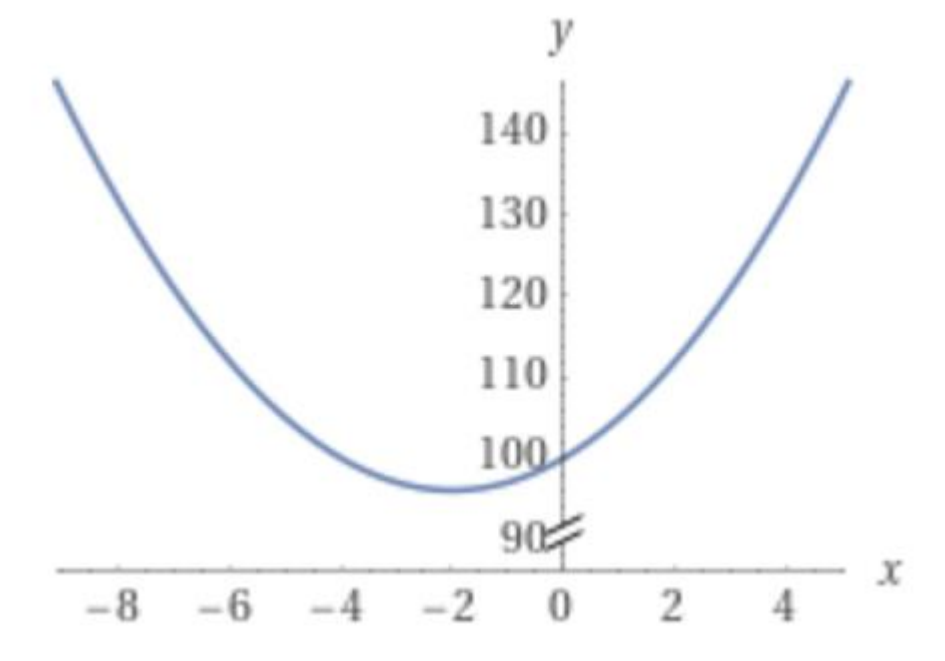
```
l, r = 0, 8
op = 100 # log(r - l) + -log(eps) + C
while op > 0:
    mid = (l + r) / 2
    if good(mid):
        l = mid
    else:
        r = mid
```

```
float l = 0, r = 0;
int op = 100; //log(r - l) + -log(eps) + C
while (op) {
    op--;
    float mid = (l + r) / 2;
    if (good(mid)) {
        l = mid;
    }
    else {
        r = mid;
    }
}
```

derivative

$$\lim_{h \to 0} \frac{f\left(x+h\right) - f\left(x\right)}{h}$$

derivative



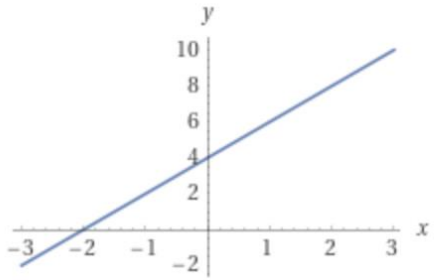
derivative of x^2 + 4x + 100

NATURAL LANGUAGE MATH INPUT

Derivative

$$\frac{d}{dx}\left(x^2 + 4x + 100\right) = 2(x + 2)$$

Plot



(x from −3 to 3)

derivative

- if f’(x) = 0, then f(x) changed direction(from going bigger to going lower)
- if f(x) = a * x^2 + b * x + c it changes direction only once
- if f’(x) = 0 we have a root for f(x) = a * x^2 + b * x + c

derivative

- h -> 0?
- We have the lowest number unlike in math
- EPS
- f’(x) = (f(x + EPS) - f(x)) / EPS
- f’(x) > 0 ⇔ f(x + EPS) - f(x) > 0

min(2x^2 - 3x + 5)

NATURAL LANGUAGE | MATH INPUT | EXTENDED KEYBOARD | EXAMPLES | UPLOAD | RANDOM

Input interpretation

minimize	$2x^2 - 3x + 5$

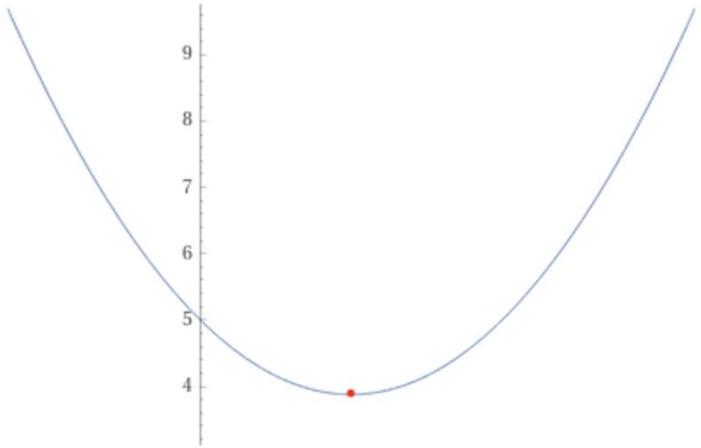
Global minimum

Decimal form | Step-by-step solution

$$\min\{2x^2 - 3x + 5\} = \frac{31}{8} \text{ at } x = \frac{3}{4}$$

Plot

Enlarge | Data | Customize | Plain Text



(x from −1 to 2.5)

редактировать fork скачать

```python
EPS = 10 ** (-9)

def f(x):
    return 2 * x * x - 3 * x + 5

def good(x):
    return f(x) - f(x - EPS) <= 0

l, r = -10, 10
ops = 100
while ops > 0:
    mid = (l + r) / 2
    if good(mid):
        l = mid
    else:
        r = mid
    ops -= 1
print(l, r)
```

#stdin #stdout 0.03s 9584KB

stdin

```
Standard input is empty
```

stdout

```
0.7500000770694047 0.7500000770694049
```

редактировать fork скачать

```cpp
#include <iostream>

using namespace std;
const double EPS = 1e-9;

double f(double x) {
    return 2 * x * x - 3 * x + 5;
}

bool good(double x) {
    return f(x) - f(x - EPS) <= 0;
}

int main() {
    double l = -10, r = 10;
    int ops = 100;
    while (ops > 0) {
        double mid = (l + r) / 2;
        if (good(mid)) {
            l = mid;
        }
        else {
            r = mid;
        }
        ops -= 1;
    }
    cout << l << " " << r;
}
```

Успешно #stdin #stdout 0.01s 5336KB

stdin

```
Standard input is empty
```

stdout

```
0.75 0.75
```

Hints(binsearch)

- if x is good, then y > x or y < x is also good
- We can somehow check if x is good
- We can guess left and right bound(later we will need only one of them)

Binary search(printers)

- We have two printers
- The first one prints x pages per minute
- The second one prints y pages per minute
- You need to print n pages, how much time will it take?

Example, x = 3, y = 4, n = 12, answer = 2 minutes

Example, x = 3, y = 4, n = 25, answer = 4 minutes

Math

$$\lceil n / (x + y) \rceil$$

Hints(printers)

- If we can print n pages in t_1 seconds, then we can print them in any t_2 >= t_1 second
- if t * x + t * y >= n, then t minutes is enough
- 0 minutes is always not enough, ⌈n / x⌉ is always enough

```python
x = 3
y = 4
n = 12

def good(t):
    return t * x + t * y < n

l = 0
r = (n - 1) // x + 1
while l < r - 1:
    mid = (l + r) // 2
    if good(mid):
        l = mid
    else:
        r = mid
print(l, r)
```

#stdin #stdout 0.04s 9428KB

stdin

```
Standard input is empty
```

stdout

```
1 2
```

редактировать fork скачать

```cpp
#include <iostream>

using namespace std;

int x = 3, y = 4, n = 12;

bool good(int t) {
    return t * x + t * y < n;
}

int main() {
    int l = 0, r = (n - 1) / x + 1;
    while (l < r - 1) {
        int mid = (l + r) / 2;
        if (good(mid)) {
            l = mid;
        }
        else {
            r = mid;
        }
    }
    cout << l << " " << r;
}
```

#stdin #stdout 0.01s 5520KB

stdin

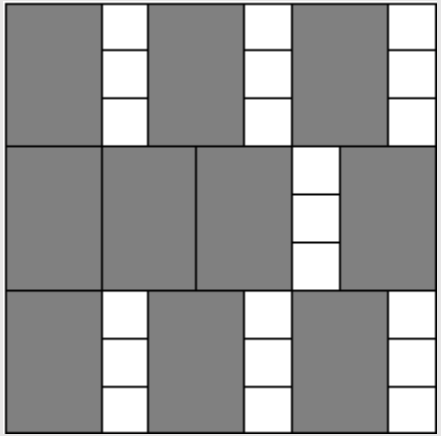
```
Standard input is empty
```

stdout

```
1 2
```

Binary search(diplomas)

- You have n pictures with size (a, b)
- You need to buy a board with size (size, size)
- What is the minimum size you need to fit all the pictures in.
- Example a = 2, b = 3, n = 10, answer = 9



Diplomas(hints)

- If we can buy a board (size_1, size_1), then we can buy (size_2, size_2) for any size_2 >= size_1
- if ⌊size / a⌋ * ⌊size / b⌋ >= n, then size is enough
- size = 0 is always not enough, max(a * n, b) is always enough

```python
a = 2
b = 3
n = 10

def good(size):
    return (size // a) * (size // b) < n

l = 0
r = n * a
while l < r - 1:
    mid = (l + r) // 2
    if good(mid):
        l = mid
    else:
        r = mid
print(l, r)
```

#stdin #stdout 0.03s 9492KB

stdin

```
Standard input is empty
```

stdout

```
8 9
```

```cpp
#include <iostream>

using namespace std;

int a = 2, b = 3, n = 10;

bool good(int size) {
    return (size / a) * (size / b) < n;
}

int main() {
    int l = 0, r = a * n;
    while (l < r - 1) {
        int mid = (l + r) / 2;
        if (good(mid)) {
            l = mid;
        }
        else {
            r = mid;
        }
    }
    cout << l << " " << r;
}
```

#stdin #stdout 0.01s 5516KB

stdin

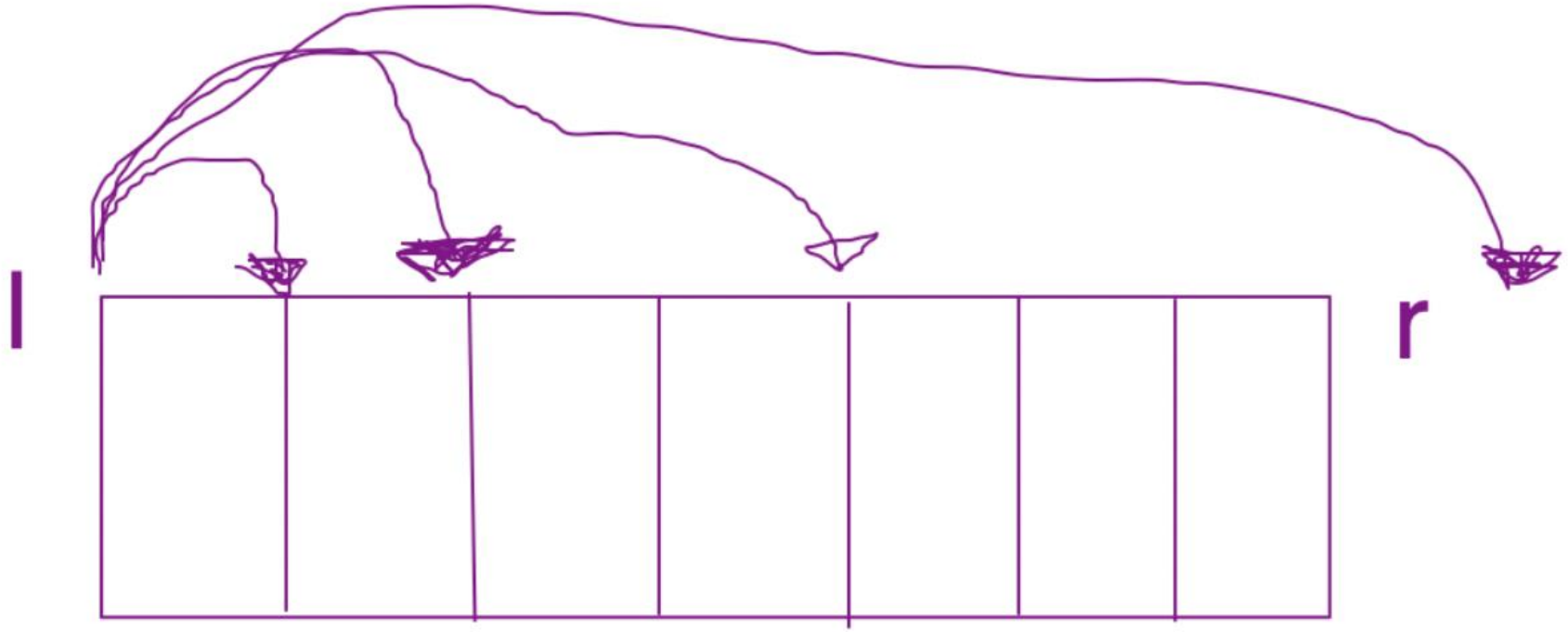
```
Standard input is empty
```

stdout

```
8 9
```

Exponential search

- What if we don’t know right bound?
- Let’s guess it with reverse binary search!!!
- For example if l = 0, r = 8, our binary search will do l += 4, l += 2, l += 1
- So this method is exponential search


l
r

```python
x = 3
y = 4
n = 100

def good(t):
    return t * x + t * y < n

l = 0
r = 1
step = 1

while good(r):
    l = r
    r += step
    step *= 2
print(l, r)

while l < r - 1:
    mid = (l + r) // 2
    if good(mid):
        l = mid
    else:
        r = mid

print(l, r)
```

#stdin #stdout 0.03s 9524KB

stdin

```
Standard input is empty
```

stdout

```
8 16
14 15
```

```cpp
#include <iostream>

using namespace std;

int x = 3, y = 4, n = 100;

bool good(int t) {
    return t * x + t * y < n;
}

int main() {
    int l = 0, r = 1, step = 1;
    while (good(r)) {
        l = r;
        r += step;
        step *= 2;
    }
    cout << l << " " << r << "\n";
    while (l < r - 1) {
        int mid = (l + r) / 2;
        if (good(mid)) {
            l = mid;
        }
        else {
            r = mid;
        }
    }
    cout << l << " " << r;
}
```

#stdin #stdout 0.01s 5516KB

cor

stdin

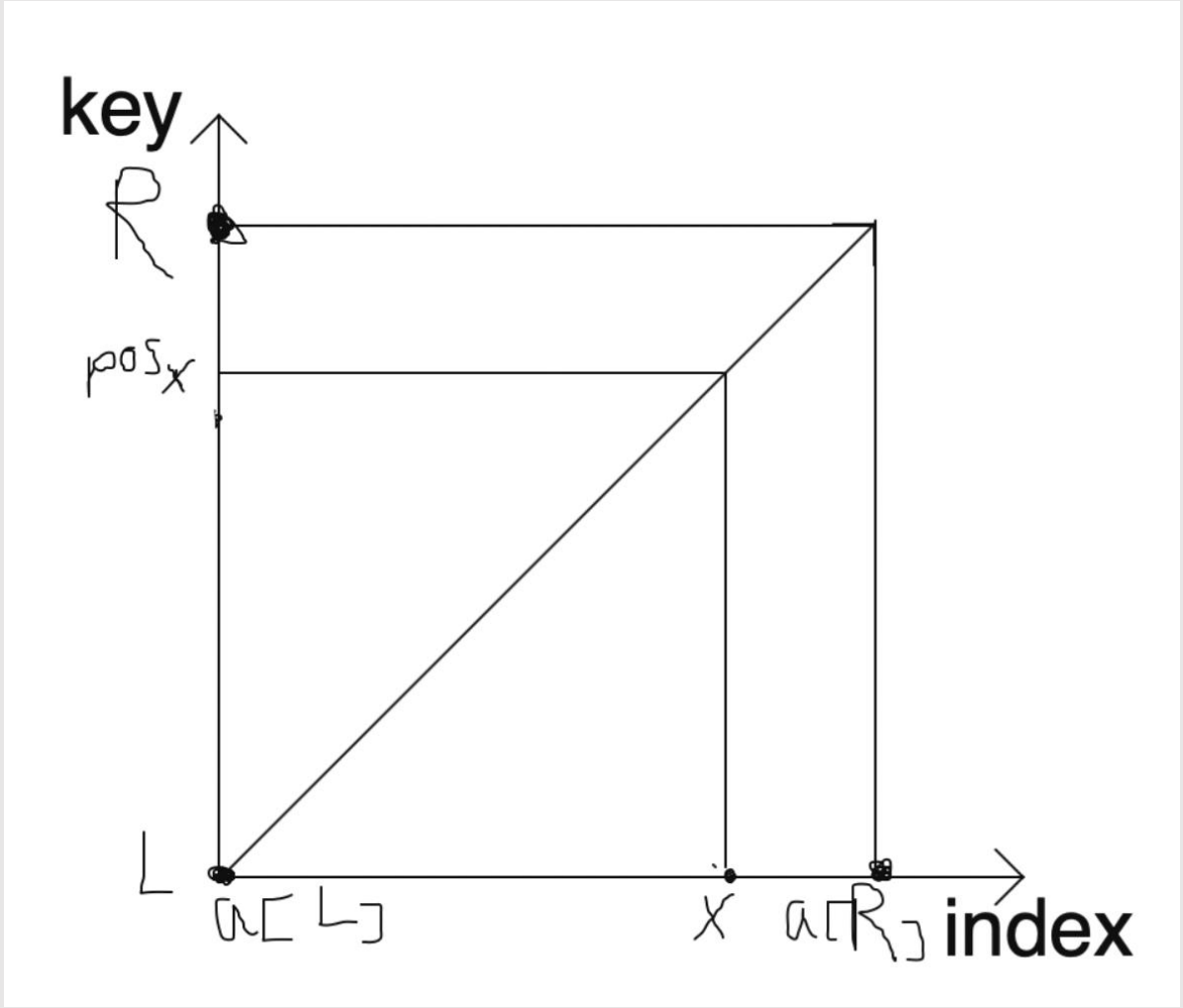
```
Standard input is empty
```

stdout

```
8 16
14 15
```

Interpolation search

- What if we can make some extra assumptions????
- Let’s imagine we search in random array or for example we search in dictionary
- For example, the names "Anny" or "Zlatan" are not good to look for somewhere in the middle of the dictionary.


key
R
pos_x
L
a[L]
x
a[R]
index

Random array

- Imagine we have random array.
- a[0..9] = 0..20, a[pos] = 5, pos = ?
- a[0..9] = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20], a[pos] = 5, pos = 2 or 3

Random array

- The x number should be approximately at the position m = (r - l) * (x - a_l) / (a_r - a_l) because (r - l) is the length, 1/(a_r-a_l) is the growth factor, and (x - a_l) is the growth required
- a[0..9] = 0..20, a[pos] = 5, pos = (9 - 0) * (5 - 0) / (20 - 0) = 2 or 3

Random array

- We have array [0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
- We have x = 12
- m = (13 - 0) * (12 - 0) / (14 - 0) = 11
- We have element = 12, so the search is finished.

Random array

- We have array [1, 5, 8, 9, 10, 11, 12, 13, 13, 13, 13, 13, 13, 14]
- We have x = 8
- m = (13 - 0) * (8 - 0) / (14 - 0) = 7
- We have array [1, 5, 8, 9, 10, 11, 12, 13]
- m = (7 - 0) * (8 - 0) / (13 - 0) = 4
- We have array [1, 5, 8, 9, 10]
- m = (4 - 0) * (8 - 0) / (10 - 0) = 3
- We have array [1, 5, 8, 9]
- m = (3 - 0) * (8 - 0) / (9 - 0) = 2
- We have array [8, 9]

```python
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
key = 12


l = 0
r = len(a) - 1
while l < r - 1:
    mid = l + (key - a[l]) * (r - l) // (a[r] - a[l])
    print(l, r, mid)
    if a[mid] < key:
        l = mid
    elif a[mid] == key:
        l = mid
        r = mid
        break
    else:
        r = mid
print(l, r)
```

#stdin #stdout 0.03s 9520KB

stdin

```
Standard input is empty
```

stdout

```
0 13 11
11 11
```

```cpp
#include <iostream>

using namespace std;

int main() {
    int a[14] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};
    int key = 12;

    int l = 0, r = 13;
    while (l < r - 1) {
        int mid = l + (key - a[l]) * (r - l) / (a[r] - a[l]);
        cout << l << " " << r << " " << mid << "\n";
        if (a[mid] < key) {
            l = mid;
        }
        else if (a[mid] == key) {
            l = mid;
            r = mid;
            break;
        }
        else {
            r = mid;
        }
    }
    cout << l << " " << r << "\n";
}
```

#stdin #stdout 0.01s 5428KB

stdin

```
Standard input is empty
```

stdout

```
0 13 11
11 11
```

Dictionary

- In the dictionary we can’t understand what is a[r] and a[l], so we need to decode it somehow
- The first option is to do an interpolation search first by the first character, then by the second character and so on
- The second option is a_0 * 26^(max_length) + a_1 * 26^(max_length - 1) + .. + a_(length_of_a) * 26^(max_length - length_of_a)

Example

aba = 1 * 26^2 + 2 * 26^1 + 1 * 26^0 = 729

baa = 2 * 26^2 + 1 * 26^1 + 1 * 26^0 = 1379

ca = 3 * 26^2 + 1 * 26^1 = 2054

sorted - [aba, baa, ca]

Worst case

- [0, (10^18) * (n - 1)]
- We need to find (10^18 - 1)
- m = (n - 0) * ((10^18 - 1) - 0) / ((10^18) - 0), so we'll move the right pointer one to the left each time.
- O(n)

Important Facts

- log_a(b) = log_c(b) / log_c(a)
- n^log_n(a) = a

Asymptotic

- if the data is random enough, we get a good enough cutoff(we can prove that then we cut off not to n/2 elements but to sqrt(n) each time)

Asymptotic

- If we reduce the number of elements from n to sqrt(n) each time, then after step 1 we get - n^(½) elements, and after step k we get - n^(½^k)
- So to get 1 element we need to do (½^k) = log_n(2) = log_2(2) / log_2(n) = 1 / log_2(n)
- So 2^k = log_2(n)
- So k = log(log(n))

Interpolation search

Visualization of the interpolation search algorithm in which 24 is the target value.

Class	Search algorithm
Data structure	Array
Worst-case performance	$O(n)$
Best-case performance	$O(1)$
Average performance	$O(\log(\log(n)))$[2]
Worst-case space complexity	$O(1)$

Ternary search

- What else can we do, if we don’t like derivatives?

min(2x^2 - 3x + 5)

NATURAL LANGUAGE | MATH INPUT | EXTENDED KEYBOARD | EXAMP

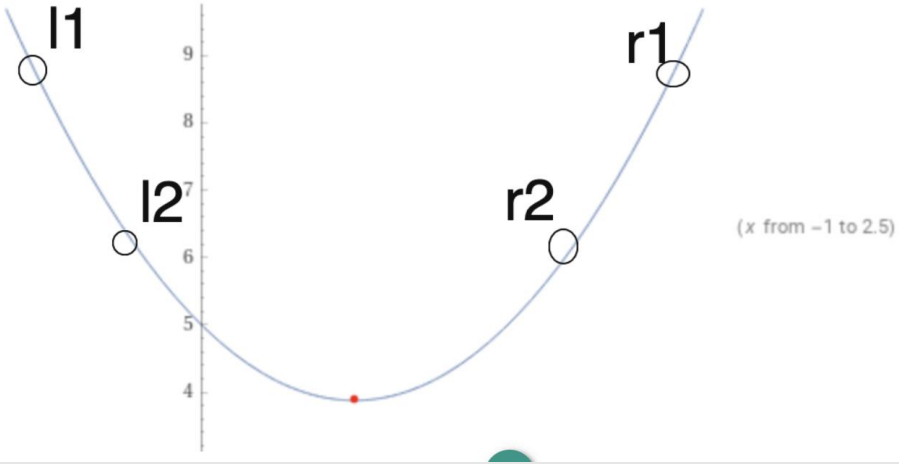
Input interpretation

minimize	$2x^2 - 3x + 5$

Global minimum | Decimal form

$$\min\left\{2x^2 - 3x + 5\right\} = \frac{31}{8} \text{ at } x = \frac{3}{4}$$

Plot | Enlarge | Data



Ternary search

- l2 = l1 + (r1 - l1) / 3, r2 = l1 + 2 * (r1 - l1) / 3
- if f(l2) < f(r2) -> answer is DEFINITELY in [l1, r2]
- if f(l2) >= f(r2) -> answer is DEFINITELY in [l2, r1]
- WHY?

Ternary search

- x is the point of minimum of the function
- f'[i] is decreasing on the segment [l, x]
- f'[i] is increasing on the segment [x, r]
- ∀a, b : l <= a <= b <= x -> f(l) >= f(a) >= f(b) >= f(x)
- ∀a, b : x <= a <= b <= r -> f(x) <= f(a) <= f(b) <= f(r)

f(a) < f(b) <-> a in [x, b] or x in [a, b]

```python
EPS = 10 ** (-9)

def f(x):
    return 2 * x * x - 3 * x + 5

l, r = -10, 10
ops = 100
while ops > 0:
    l_1 = l + (r - l) / 3
    r_1 = l + 2 * (r - l) / 3
    if f(l_1) < f(r_1):
        r = r_1
    else:
        l = l_1
    ops -= 1
print(l, r)
```

#stdin #stdout 0.03s 9520KB

stdin

```
Standard input is empty
```

stdout

```
0.7500000117556708 0.7500000117556709
```

```cpp
#include <iostream>

using namespace std;
const double EPS = 1e-9;

double f(double x) {
    return 2 * x * x - 3 * x + 5;
}

int main() {
    double l = -10, r = 10;
    int ops = 100;
    while (ops > 0) {
        double l_1 = l + (r - l) / 3;
        double r_1 = l + 2 * (r - l) / 3;
        if (f(l_1) < f(r_1)) {
            r = r_1;
        }
        else {
            l = l_1;
        }
        ops -= 1;
    }
    cout << l << " " << r;
}
```

#stdin #stdout 0.01s 5512KB

stdin

```
Standard input is empty
```

stdout

```
0.75 0.75
```

Asymptotic

- Each time we go from a segment of length n to length 2/3 n
- Just like in the binary search we need to do x operations until we get 1 element
- It is log_{3/2}(n), so we have O(log(n))

Asymptotic

We have n cows and m stalls, we need to find the maximum distance, that we can achieve between cows.

Cows and stalls

We have 3 cows and 5 stalls - [1, 2, 3, 100, 1000]

The answer = 99

first cow - 1

second cow - 100

third cow - 1000

Cows and stalls

- If distance x is reachable, then distance y < x is also reachable
- We can somehow check if x by trying to reach such distance
- l = min distance between stalls, r = max distance between stalls + 1

Cows and stalls

5 3

1 2 3 100 1000

you need to check 50

1 - ok

2 - not ok, because dist < 50

3 - not ok, because dist < 50

100 - ok, because dist > 50

1000 - ok, because dist > 50

All codes

linear search(c++) - https://ideone.com/IwmVsN

linear search(python) - https://ideone.com/JeXdN4

binary search - array(python) - https://ideone.com/YUyauM

binary search - array(c++) - https://ideone.com/RUZWlY

bs - devirative(python) - https://ideone.com/2RKHfZ

bs - devirative(c++) - https://ideone.com/kAPZBT

printers(c++) - https://ideone.com/4vqHK4

printers(python) - https://ideone.com/nLmWCh

exp search(python) - https://ideone.com/SmEouh

exp search(c++) - https://ideone.com/IJqlI8

diplomas(c++) - https://ideone.com/XB5jWk

diplomas(python) - https://ideone.com/pdZX8h

intersearch(python) - https://ideone.com/yEY8hf

intersearch(c++) - https://ideone.com/GGOCo6

ternary_search(python) - https://ideone.com/U9QFUS

ternary_search(c++) - https://ideone.com/h0rqo6

That's All Folks!