Applications of Binary search

Today..

- Review: What is *binary search*?
- Binary search on monotonically increasing functions
- Basic methods of solving some problems by binary search
  - Using binary search directly
  - Converting optimization problems to decision problems

Today..

- **Review: What is *binary search*?**
- Binary search on monotonically increasing functions
- Basic methods of solving some problems by binary search
    - Using binary search directly
    - Converting optimization problems to decision problems

What was *binary search*?

- We want to find an element x from a sorted array $a[1..n]$ ($a[1] \leq a[2] \leq \cdots \leq a[n]$)
- If we search naively, it takes $O(n)$ time.
- However, we can do it in $O(\log n)$ time because..
  - If $x < a[m]$, $x < a[m+1], a[m+2], \cdots, a[n]$ also holds.
  - If $x > a[m]$, $x > a[m-1], a[m-2], \cdots, a[1]$ also holds.
  - So if we let m as a midpoint of $[1, n]$, the size of the interval becomes at most half.

What was *binary search*? (cont.)

1	2	3	4	5	6	7	8	9	10
1	2	5	7	9	11	14	16	21	25

If $x < 9$,
we can shrink the interval
into $[1, 4]$

If $x > 9$,
we can shrink the interval
into $[6, 10]$

The size of the interval was 10,
but after comparing it became at most 5.

What was *binary search*? (cont.)

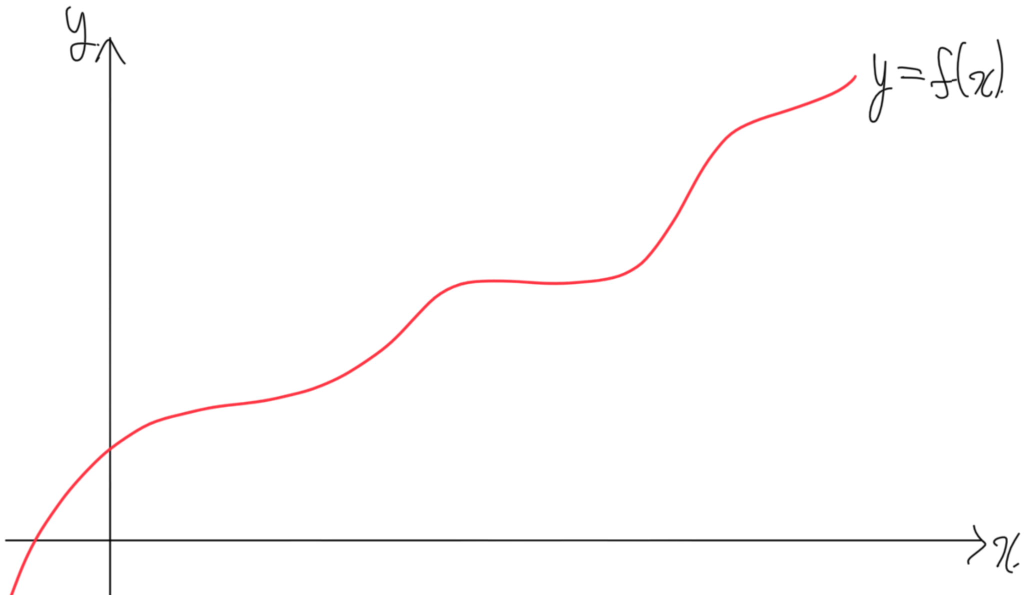
```c
int l = 0, r = n-1; // If x is in the array, it must be inside a[l..r]
while(l <= r) {
    int m = (l + r) / 2;
    if(x < a[m]) {
        r = m - 1;
    } else if(x > a[m]) {
        l = m + 1;
    }else { // we found it!
        printf("we found %d in position %d\n", x, m);
        break;
    }
}
```

Today..

- Review: What is *binary search*?
- **Binary search on monotonically increasing functions**
- Basic methods of solving some problems by binary search
  - Using binary search directly
  - Converting optimization problems to decision problems

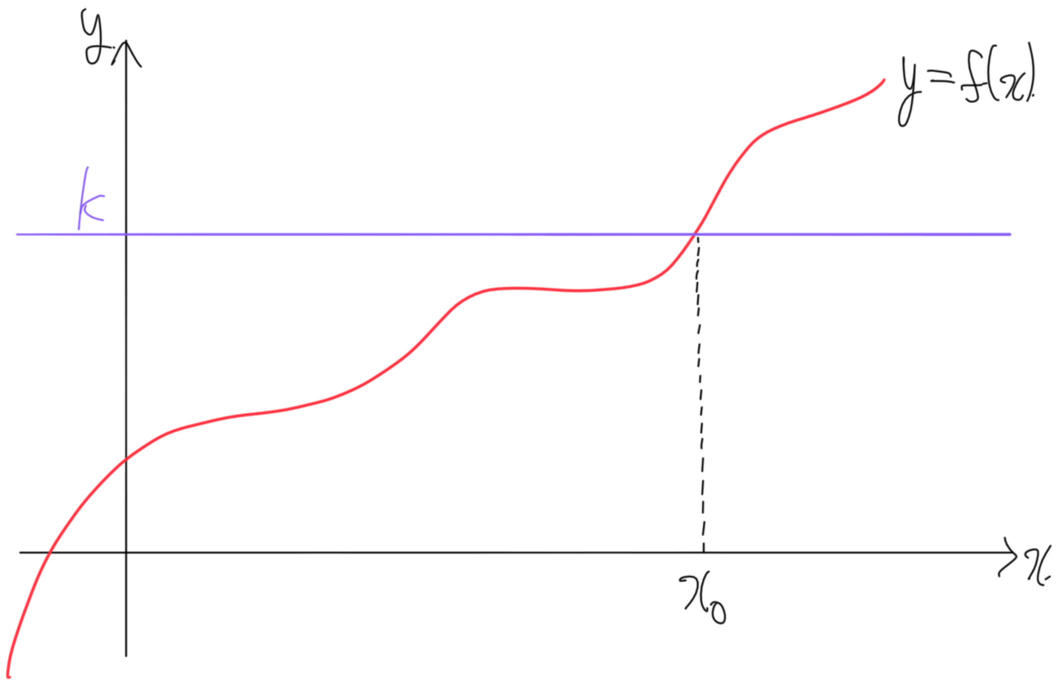
Binary search on functions

- To apply binary search on arrays, the array should be *monotonically increasing*.
- Like that, we can also apply a binary search on a *monotonically increasing* function $f\colon \mathbb{R} \to \mathbb{R}$. Assume it is continuous.



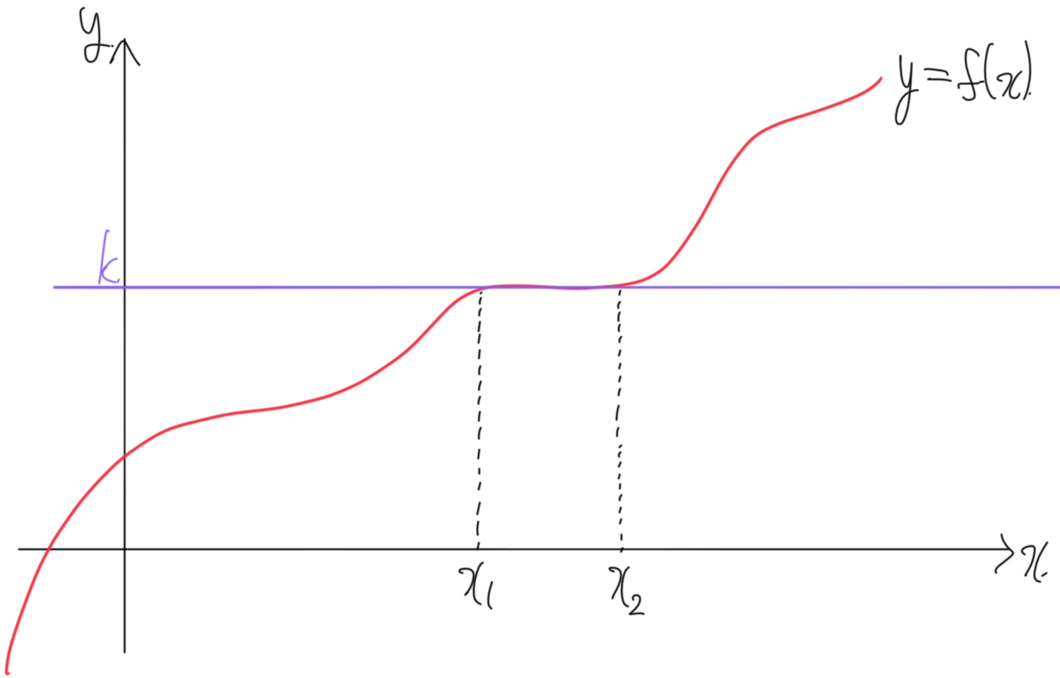
Binary search on functions (cont.)

- Given a real number k, we are trying to find some x_0 such that $f(x_0) = k$.



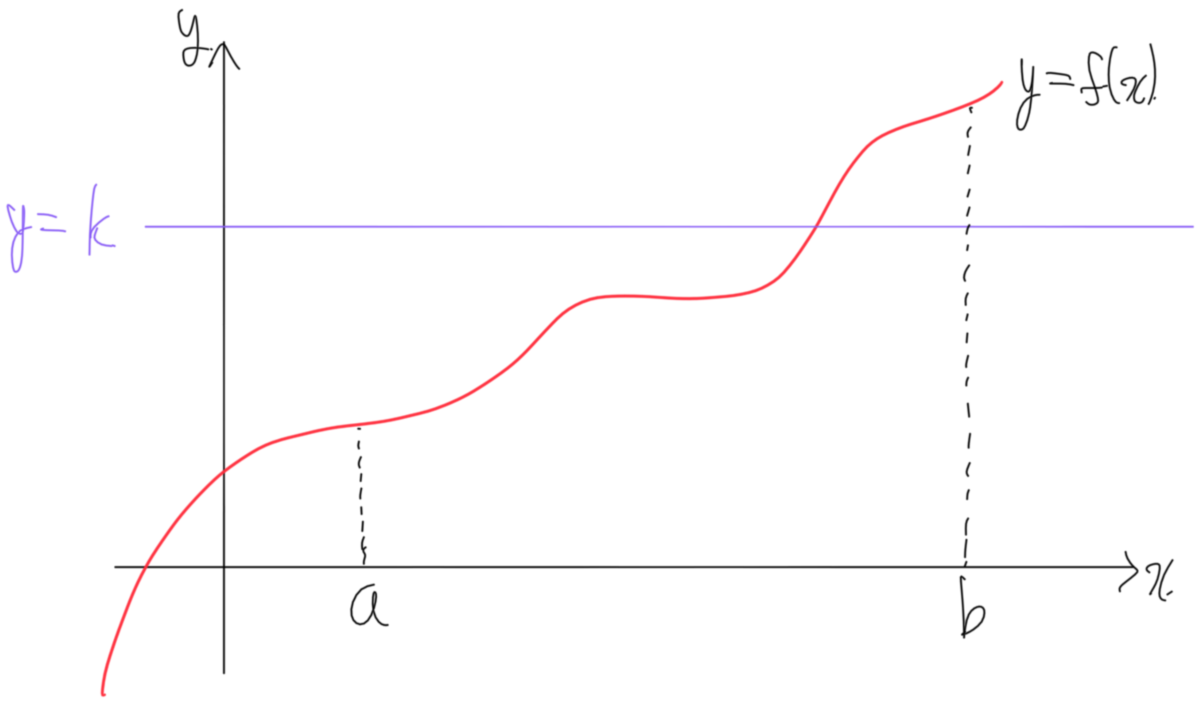
Binary search on functions (cont.)

- As this function is monotonically increasing, the range of the possible x_0 can form an interval. So we are interested in finding two endpoints x_1 and x_2.



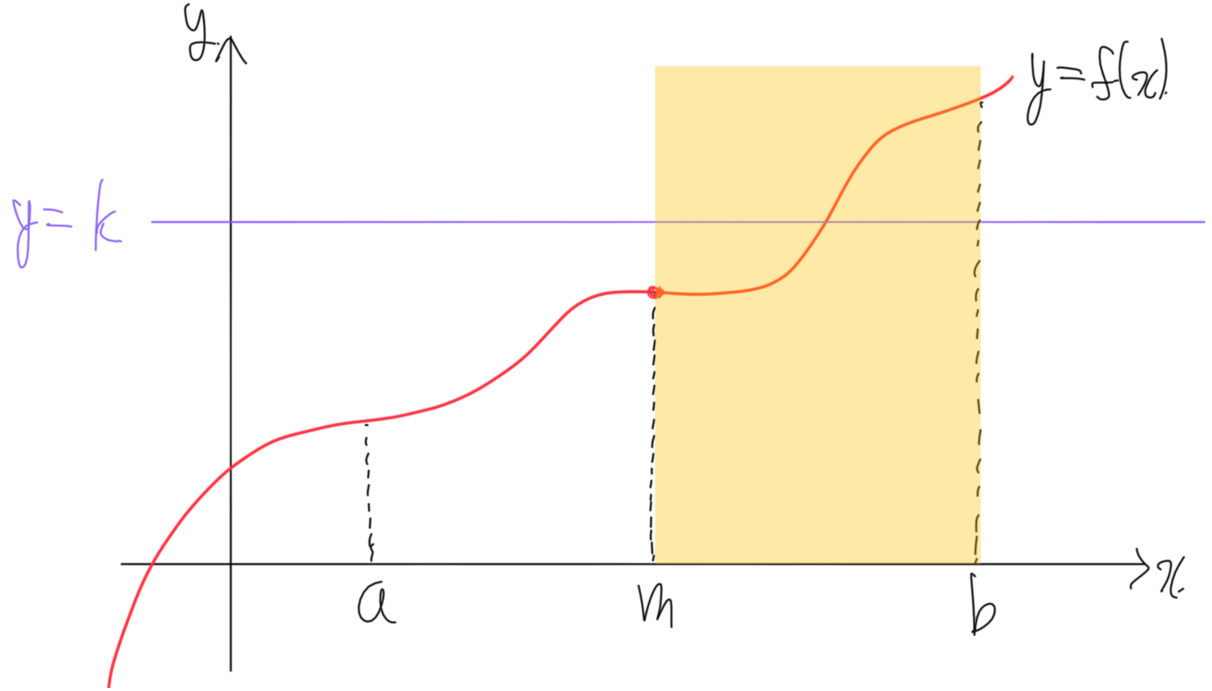
Binary search on functions (cont.)

- First, let's find a proper interval $[l, r]$ such that $f(l) < k$ and $f(r) > k$. As f is monotonically increasing, we are certain that the answer is inside the interval.



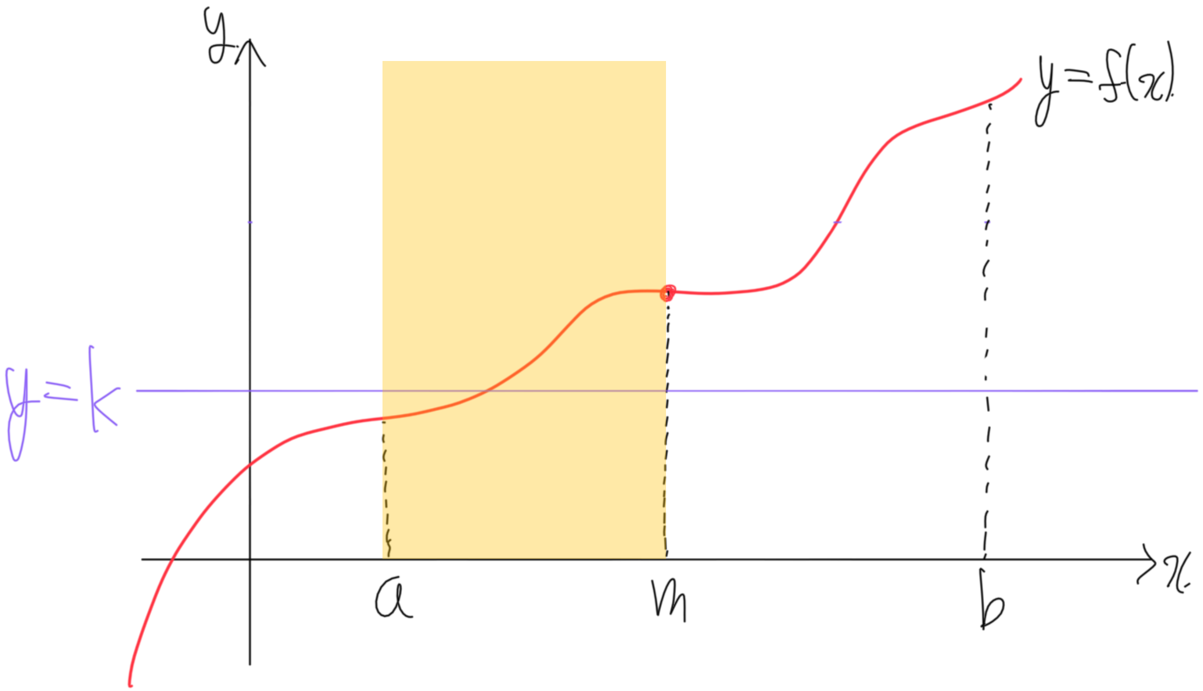
Binary search on functions (cont.)

- Find the midpoint $m = \frac{a+b}{2}$ and calculate $f(m)$.
  - If $f(m) < k$, $f(a..m) < k$ also holds. Therefore we can shrink the interval to $[m, b]$.



Binary search on functions (cont.)

- Find the midpoint $m = \frac{a+b}{2}$ and calculate $f(m)$.
  - If $f(m) > k$, $f(m..b) > k$ also holds. Therefore we can shrink the interval to $[a, m]$.

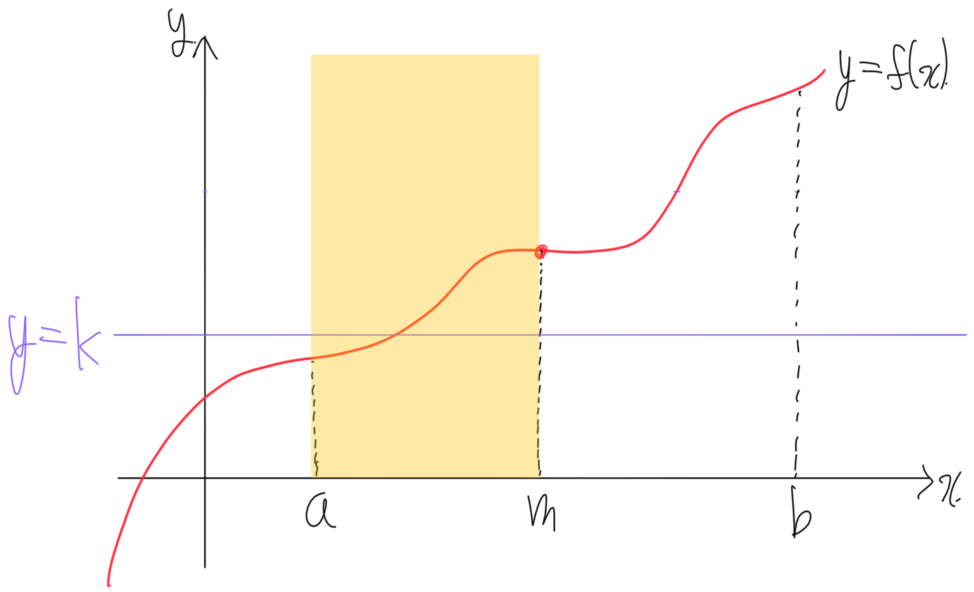


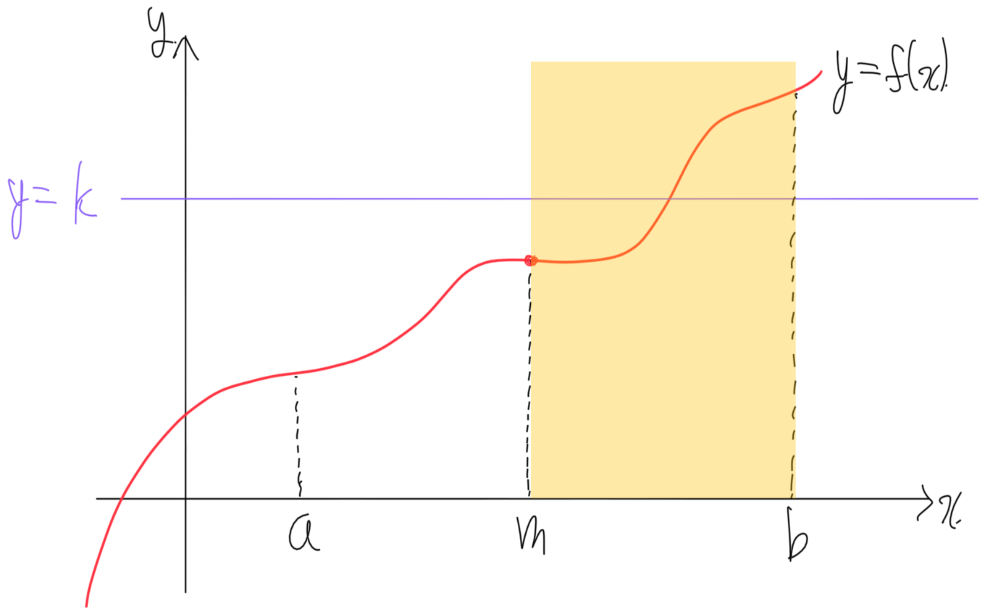
Binary search on functions (cont.)

- So, we can start from the interval $[l, r]$ and repeat these steps:
  - Set the midpoint $m = \frac{l+r}{2}$.
  - If $f(m) > k$, shrink the interval to $[l, m]$.
  - If $f(m) < k$, shrink the interval to $[m, r]$.
- If we repeat these steps s times, the length of the interval will become $\frac{(r-l)}{2^s}$.
- You should fix an appropriate s, by comparing $\frac{(r-l)}{2^s}$ and the desired precision that we want.

Binary search on functions (cont.)

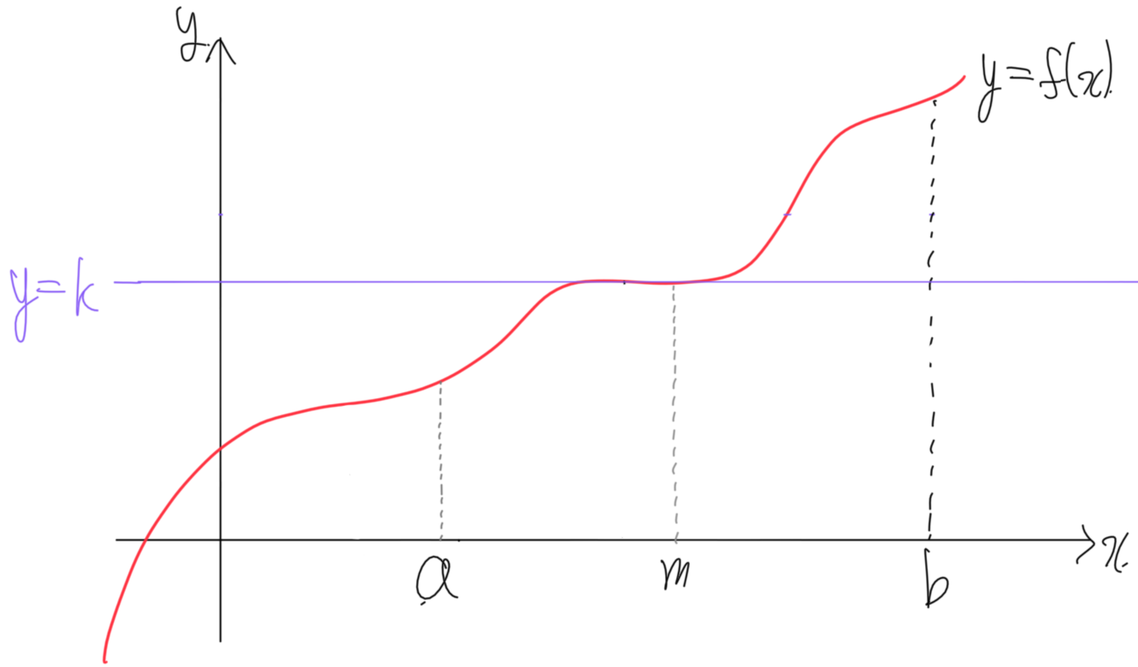
- But what if we need to find the endpoints?
- If $f(x) \neq k$, we shrink the interval in the same way we've discussed just before.





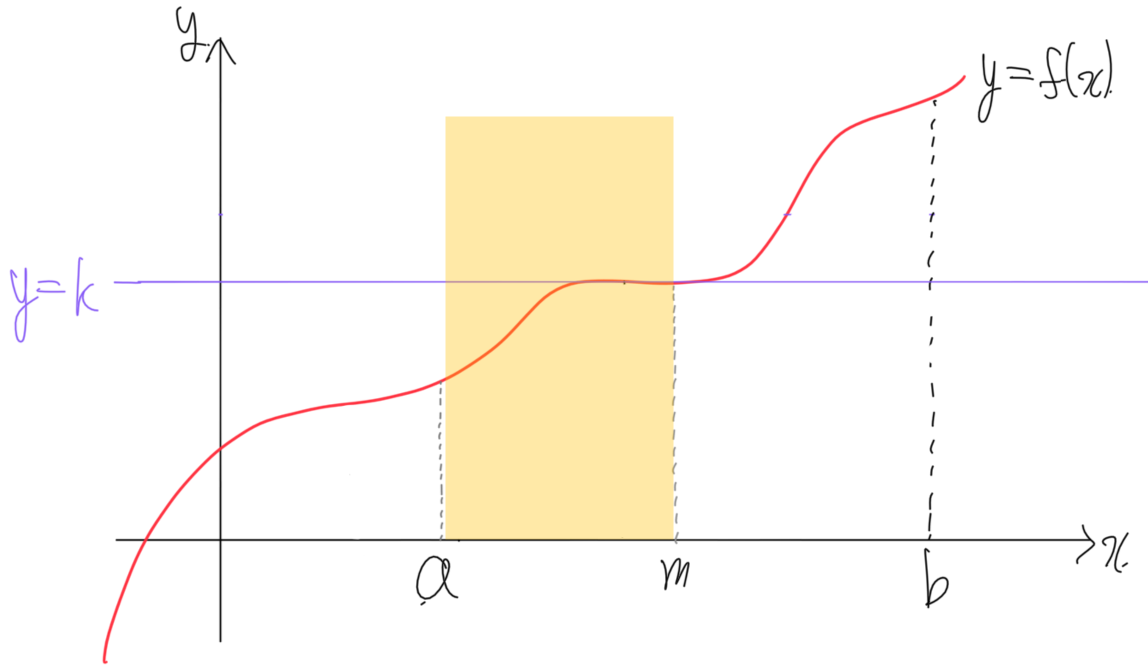
Binary search on functions (cont.)

- However, if $f(m) = k$, it is slightly different.



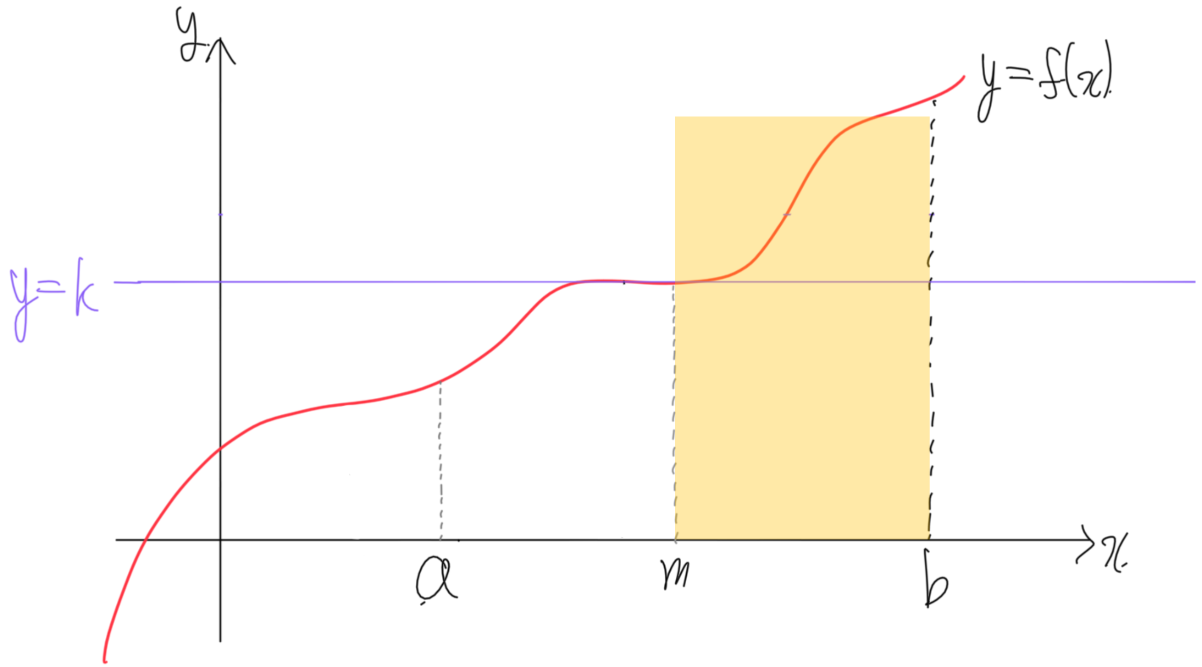
Binary search on functions (cont.)

- If we were to calculate the leftmost endpoint, we have to shrink the interval into $[a, m]$.



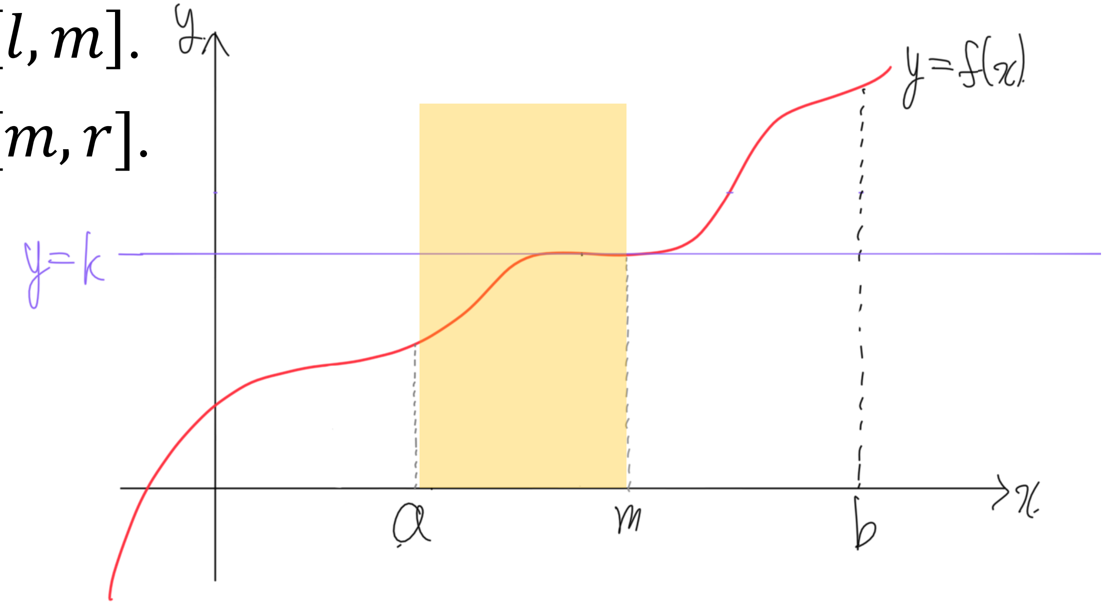
Binary search on functions (cont.)

- If we were to calculate the rightmost endpoint, we have to shrink the interval into $[m, b]$.



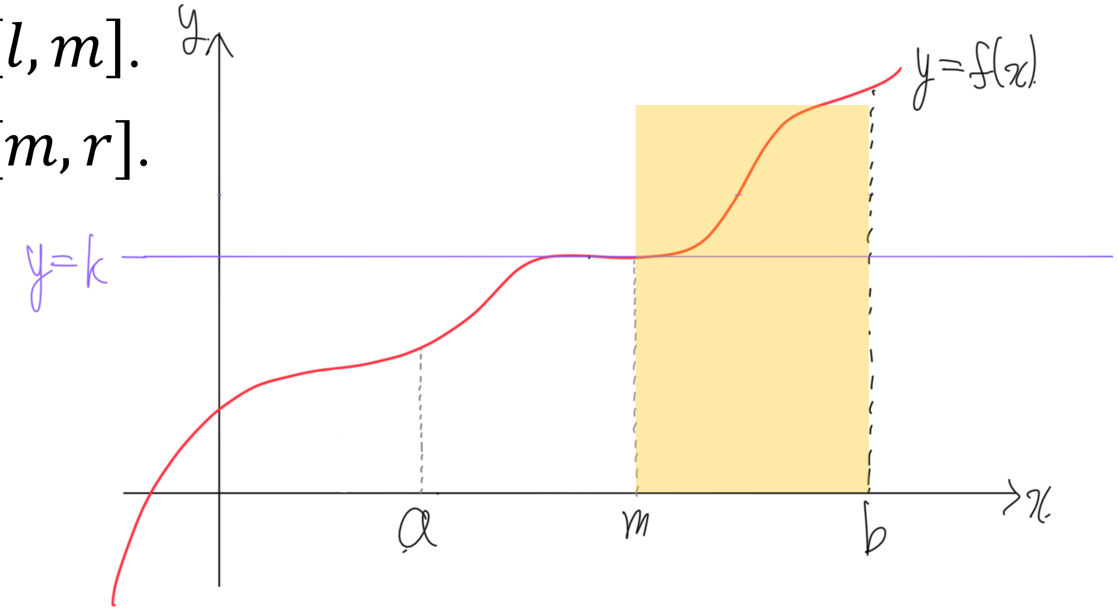
Binary search on functions (cont.)

- In short: To find the **leftmost** endpoint, we can start from the interval $[l, r]$ and repeat these steps:
  - Set the midpoint $m = \frac{l+r}{2}$.
  - If $f(m) \geq k$, shrink the interval to $[l, m]$.
  - If $f(m) < k$, shrink the interval to $[m, r]$.



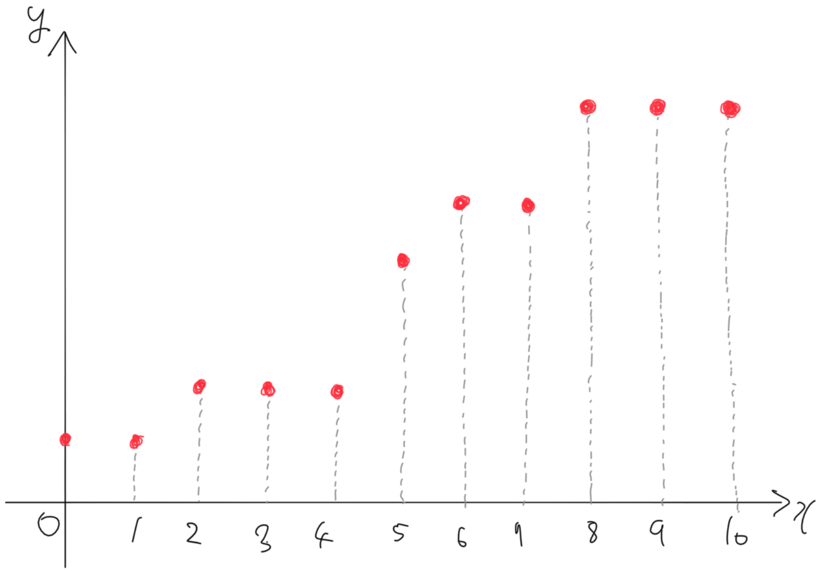
Binary search on functions (cont.)

- In short: To find the **rightmost** endpoint, we can start from the interval $[l, r]$ and repeat these steps:
  - Set the midpoint $m = \frac{l+r}{2}$.
  - If $f(m) > k$, shrink the interval to $[l, m]$.
  - If $f(m) \leq k$, shrink the interval to $[m, r]$.



Binary search on functions (cont.)

- We can apply this not only on real numbers, but also on **integers**.
- Actually, arrays (or sequences) can be just seen as a function $f\colon \mathbb{N} \to \mathbb{N}$ or $f\colon \mathbb{N} \to \mathbb{R}$, so you can apply the same method.

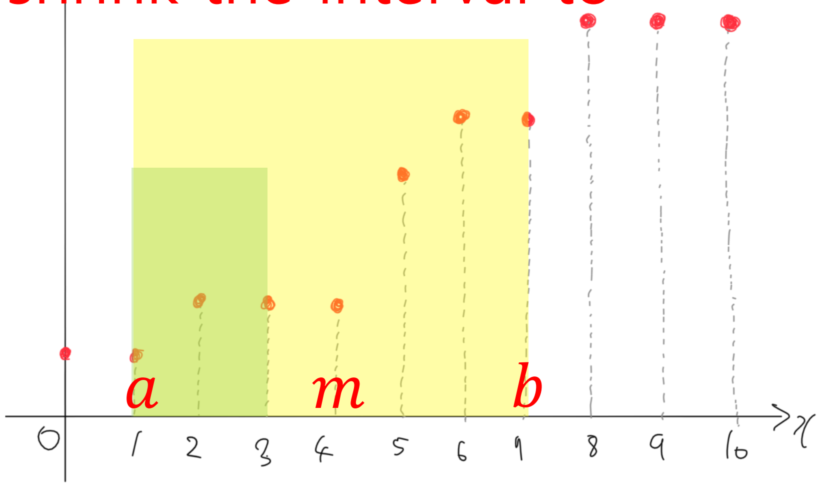


Binary search on functions (cont.)

- However, the situation is slightly different. For example, let's try to get the **leftmost** point that $f(x_l) = k$.
- We are going to store x_l in a variable `xl`.
- Let's make a large interval $[l, r]$ that must contains x_l.

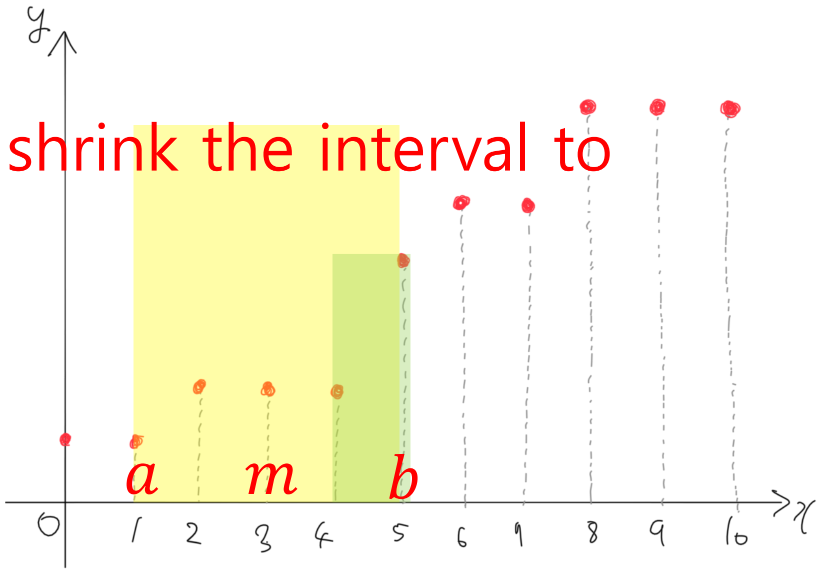
Binary search on functions (cont.)

- Then we can repeat the steps until $l \leq r$:
  - Set the midpoint $m = \left\lfloor \frac{l+r}{2} \right\rfloor$.
  - If $f(m) > k$, shrink the interval to $[l, m-1]$.
  - If $f(m) < k$, shrink the interval to $[m+1, r]$.
  - If $f(m) = k$, store the value of m into `xl`, and shrink the interval to $[l, m-1]$.
- After repeating, the variable `xl` stores the leftmost point.



Binary search on functions (cont.)

- We can also find the rightmost point x_r too, if we repeat these steps until $l \leq r$:
  - Set the midpoint $m = \left\lfloor \frac{l+r}{2} \right\rfloor$.
  - If $f(m) > k$, shrink the interval to $[l, m-1]$.
  - If $f(m) < k$, shrink the interval to $[m+1, r]$.
  - If $f(m) = k$, store the value of m into `xr`, and shrink the interval to $[m+1, r]$.
- After repeating, the variable `xr` stores the rightmost point.



Binary search on functions (cont.)

- By using these methods, we can also calculate
  - The leftmost x such that $f(x) \geq k$.
  - The rightmost x such that $f(x) \leq k$.
- ..but we leave these as your exercises. Try to fix a midpoint, and use the fact that f is monotonically increasing.

Today..

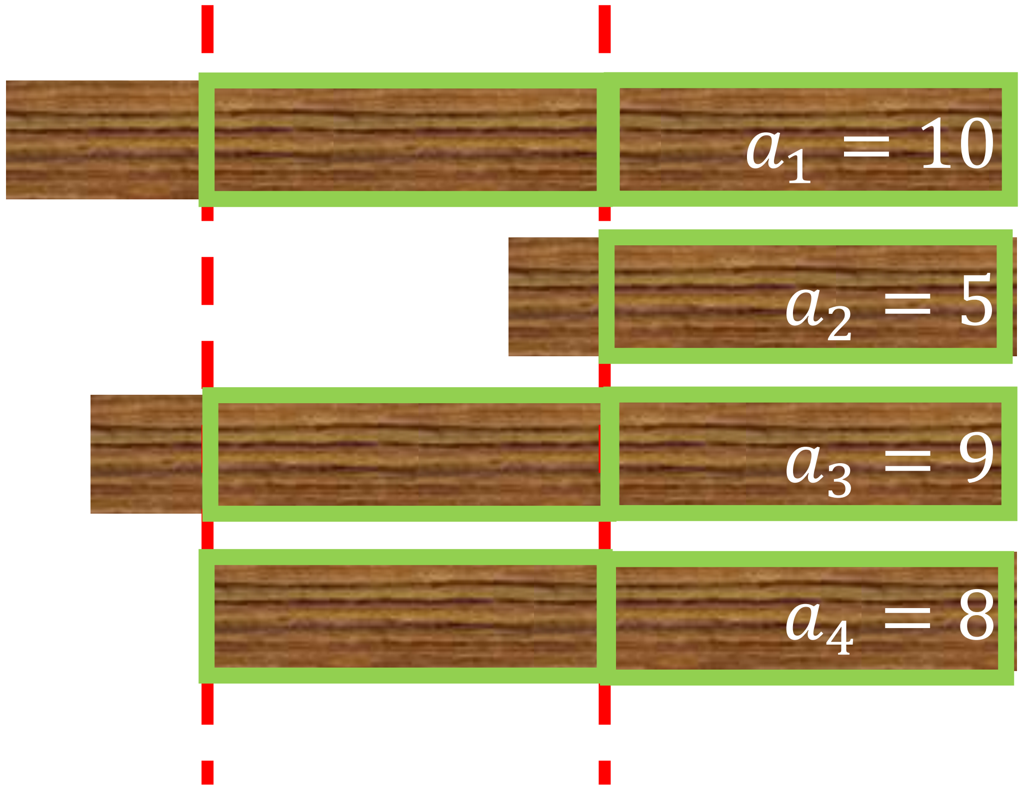
- Review: What is *binary search*?
- Binary search on monotonically increasing functions
- Basic methods of solving some problems by binary search
  - **Using binary search directly**
  - Converting optimization problems to decision problems

Some problems can be solved by binary search on functions

- Example problem:
- You are given n sticks. Each stick has length $a_1, a_2, \cdots, a_n$.
- We are going to make at least k sticks with the same length, by cutting the given sticks.
- We are *not allowed* to glue sticks, so we may have to throw away some sticks (if the length is not the same)
- What is the ***maximum possible length*** the resulting sticks?

Some problems can be solved by binary search on functions (cont.)

- Example for the example problem:



If $k = 7$, we can cut the sticks into length 4. Then, exactly 7 sticks are available, and it is the maximum possible length.

Some problems can be solved by binary search on functions (cont.)

- Solution: Suppose we fixed the length by x.
- Then, we can easily calculate the number of usable sticks by:

$$f(x) = \sum_{i=1}^{n} \lfloor \frac{a_i}{x} \rfloor$$

- And $f(x)$ is *monotonically decreasing*, since
  - $\lfloor \frac{a_i}{x} \rfloor$ is a monotonically decreasing function, so sum is monotonically decreasing too.
  - It is obvious that if we increase the length, the number of usable sticks will decrease.

Some problems can be solved by binary search on functions (cont.)

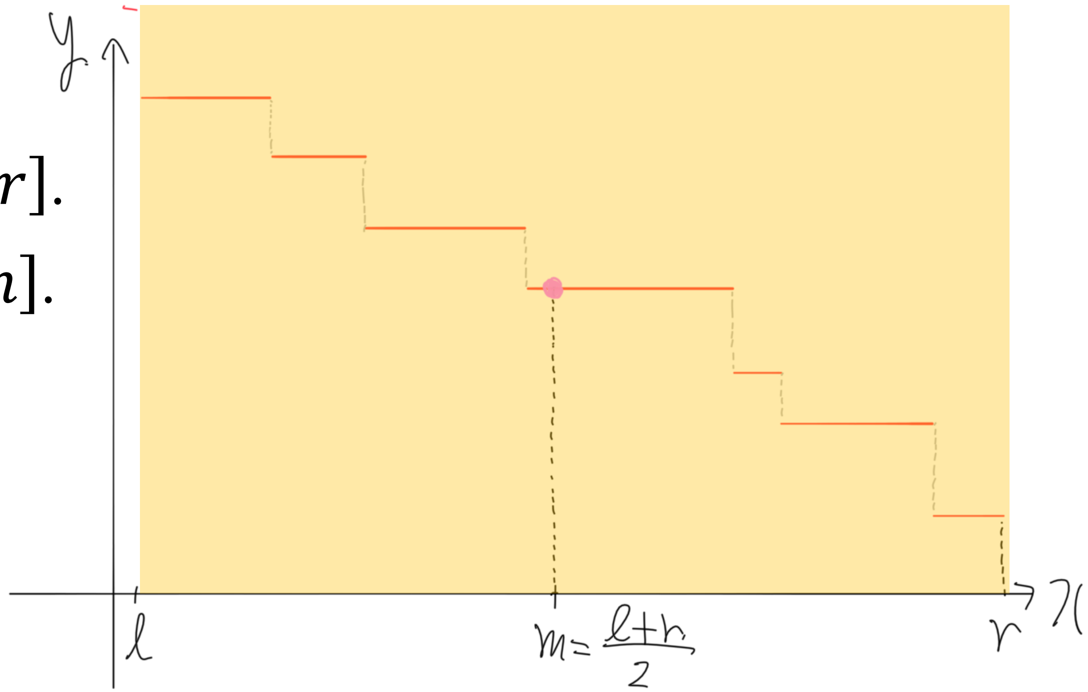
- So the problem became:

  Given a monotonically decreasing function $f(x)$, find the ***rightmost(largest)*** $\boldsymbol{x}$ $(x > 0)$ such that $\boldsymbol{f(x) \geq k}$ holds.

- We can easily find such x by binary search!
  - The function is *decreasing* (not increasing), but if we flip the function, the function can be seen as an increasing function.
  - So the binary search logic works, if we change the details slightly.

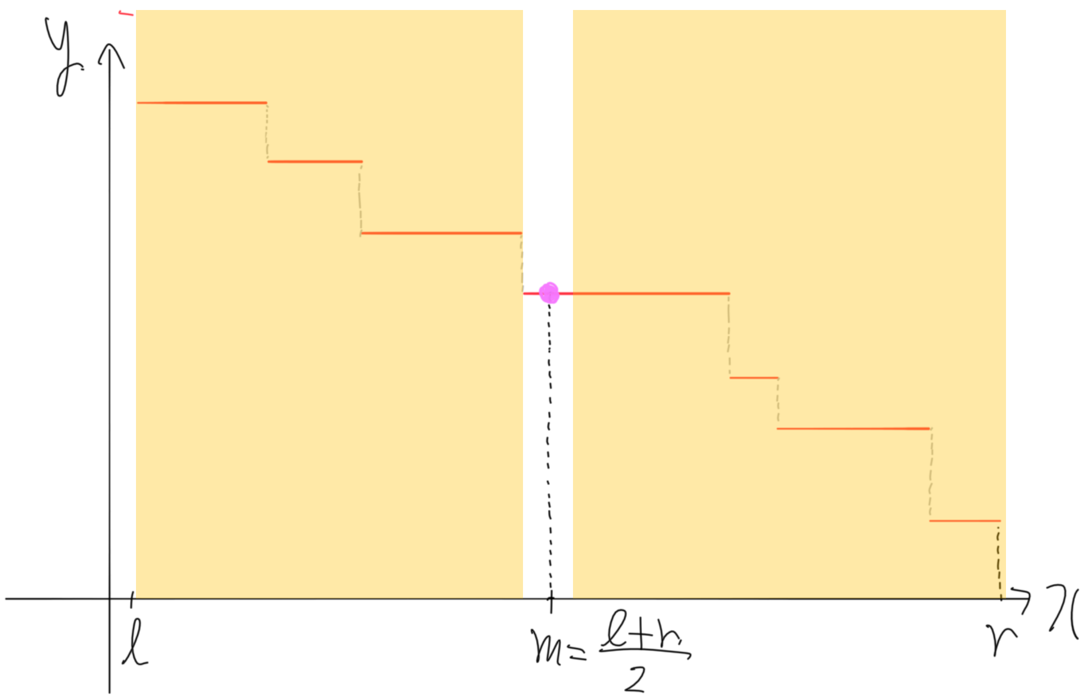
Some problems can be solved by binary search on functions (cont.)

- If we were to find a *real number* x, we could do like:
  - Start from the interval $[\epsilon, \max(a_i)]$.
  - Find the midpoint $m = \frac{l+r}{2}$.
  - If $f(m) \geq k$, shrink the interval to $[m, r]$.
  - If $f(m) < k$, shrink the interval to $[l, m]$.
  - Repeat this step about 100 times.



Some problems can be solved by binary search on functions (cont.)

- If we were to find a *integer* x, we could do like:
  - Start from the interval $[1, \max(a_i)]$.
  - Find the midpoint $m = \lfloor\frac{l+r}{2}\rfloor$.
  - If $f(m) \geq k$, update the variable $\mathtt{xr}$ to m, and shrink the interval to $[m+1, r]$.
  - If $f(m) < k$, shrink the interval to $[l, m-1]$.
  - Repeat this step until $l \leq r$.



Today..

- Review: What is *binary search*?
- Binary search on monotonically increasing functions
- Basic methods of solving some problems by binary search
  - Using binary search directly
  - **Converting optimization problems to decision problems**

Converting to a decision problem

- Suppose we have to solve a problem like:

  Find the **largest** $\boldsymbol{x}$ that satisfies these conditions: …

- In some cases, we can easily find whether a given x satisfies the given conditions.

$$f(x) = \begin{cases} 1, & x \text{ satisfies the given conditions} \\ 0, & \text{otherwise} \end{cases}$$

Converting to a decision problem (cont.)

- If f is **monotonically decreasing**, we can find the *largest* x using binary search.

Converting to a decision problem (cont.)

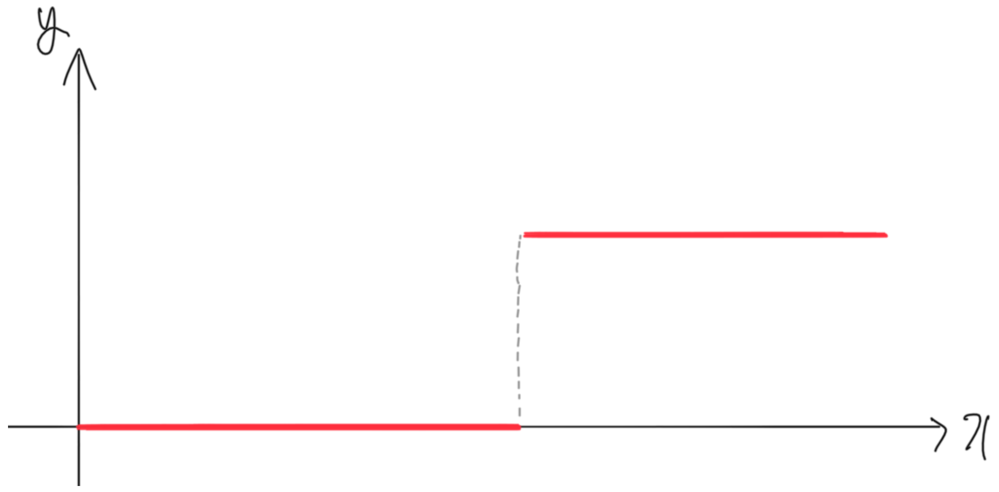
- In short, this idea is extremely useful if
  - It is really easy to find whether a **fixed $\boldsymbol{x}$ satisfies the conditions**.
  - If x satisfies the conditions, **any $\boldsymbol{y}$ smaller than $\boldsymbol{x}$** also satisfies the conditions.
- If the time complexity of calculating $f(x)$ is $O\big(T(n)\big)$, we can find the answer in $O(T(n)\log RANGE)$ time.

Example of converting to a decision problem

- There are n grocery stores and m schools in a 1D line.
- I am going to build a house on this line on coordinate x.
- I want to visit at least one store and one school while living, so I would like to ***minimize*** **max(*****the distance to the nearest store, the distance to the nearest school*****)**.
- What is the minimum?

Example of converting to a decision problem (cont.)

- Solution: Let $f(d)$ be 1 if we can build a house such that at least one store and one school within distance d, 0 otherwise.
- $f(d)$ is *monotonically increasing*, because if both one store and one school are within distance d, it is also within distance $d + \epsilon$.



Example of converting to a decision problem (cont.)

- So it is sufficient to know how to calculate $f(d)$. We want to know whether a coordinate x exists such there is a store and a school within distance d.
- A store in coordinate a_i is within distance d if:

$$|x - a_i| \leq d$$

- If we write this in terms of x,

$$a_i - d \leq x \leq a_i + d \iff x \in [a_i - d, a_i + d]$$

Example of converting to a decision problem (cont.)

- We can do the same thing with schools.
- A school in coordinate b_j is within distance d if:

$$\left|x - b_j\right| \leq d$$

- If we write this in terms of x,

$$b_j - d \leq x \leq b_j + d \iff x \in \left[b_j - d, b_j + d\right]$$

Example of converting to a decision problem (cont.)

- So, for each school and for each store, we can make an interval that the house should be in.
- The condition is *at least one store and one school should be in distance* d.
- So we just need to check whether there exists a point such that is inside a red interval and a blue interval. This is easy to check.