# Binary and Ternary Search lecture02

Raveesh Gupta

March 2019

# Chapter 1

# Binary Search

**Binary Search is a search algorithm that find the position of a target value within a sorted data.** Invariant: Given $a_{1...n}$. Maintain $l$ and $r$, that $a_{l...r}$ contains the value

Step: Update $l$ and $r$ by $mid := l + (r - l)/2$

## 1.1 Implementation

```
\\first occurrence of b
l := 1, r: = n
while r - l > 1
    mid := l + (r-l)/2
    if a[mid] < b
        l = mid
    else
        r = mid
    for i:= l..r
    if a[i] == b
        return i
 return -1

\\last occurrence of b
l := 1, r: = n
while r - l > 1
    mid := l + (r-l)/2
    if a[mid] <= b \\CHANGE no. 1
        l = mid
    else
        r = mid
for i:= r..l\\CHANGE no. 2
    if a[i] == b
```

```
        return i
return -1

\\Finding first greater or equal occurrence (std:lower_bound)
l := 1, r: = n
while r - l > 1
    mid := l + (r-l)/2
    if a[mid] < b
        l = mid
    else
        r = mid
for i:= l..r
    if a[i] >= b \\CHANGE NO. 1
    return i
return -1

\\Finding strictly greater occurrence (std:upper_bound)
l := 1, r: = n
while r - l > 1
    mid := l + (r-l)/2
    if a[mid] < b
        l = mid
    else
        r = mid
for i:= l..r
    if a[i] > b \\CHANGE NO. 1
    return i
return -1
```

Standard C++ Library functions: *lower$_b$ound*, *binary$_s$earch*, *upper$_b$ound*, *equal$_r$ange* must be used whenever the solution is trivial.

## 1.2 Re-enumerate the Sequence

Transform an array $a_i$ such that its order remain same but its minimum value starts from 0.

```
vector<int> b(a);
sort(b.begin(), b.end());
b.erase(unique(b.begin(),b.end()), b.end())
for(int i =0; i<a.size();i++)
    a[i] = lower_bound(b.begin(), b.end(), a[i]) - b.begin();
```

**It is used to transform the large data set into small data set (without changing the order) or to convert a set of integers into non-negative integers.**

## 1.3 Number Of Occurrences

Given two arrays $a$ and $b$ of size $n$ and $m$, $\forall \mathrm{b}_i$, i $\epsilon$ $[1..m]$ find its number of occurrences in $a$.

```
sort(a.begin(), a.end());
for(int b_i: b){
    auto range = equal_range(a.begin(), a.end(), b_i);
    cout << range.second - range.first<< endl;
}
```

# Chapter 2

# Smallest $n * n$ square to fit $k$ $a * b$ rectangles

**Find the smallest size n of square that it can fit at least $k$ non-overlapping $a * b$ rectangles (they can't be rotated).**

## 2.1 Analysis

To find the solution we invert the problem, instead of finding $n$, try to find the maximum number of $a * b$ rectangles that can be inserted in an $n * n$ square. Let $f(n)$ be this number.

$$f(n) = \left\lfloor \frac{n}{a} \right\rfloor * \left\lfloor \frac{n}{b} \right\rfloor$$

**Notice this function is a monotonically increasing function.** Find $MIN$ $n_0$ such that $f(n_0) >= k$.

## 2.2 Solution by Binary Search

```
l := 1, r: = 1e9+1
while r - l > 1
    mid := l + (r-l)/2
    if (mid/a)*(mid/b) < k
        l = mid
    else
        r = mid
    for i := l..r
        if (i/a) * (i/b) >= k
        return i
```

**Notice that a solution always exist as a very big square of** $n*n, n >>>> a, b$ **can fit all the rectangles.**

# Chapter 3

# Maximum equal k Pieces

Given n spool of thread. The *ith* contains $a_i$ of thread. You can cut thread to cutoff k pieces of thread of equal (same) length. What is the maximal possible length of k pieces?

## 3.1 Analysis

To find the solution we invert the problem, instead of finding maximum length $l$ , we assume that we know the length $l$ and check if its possible to get $k$ equal pieces.

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

Note $f(l)$ is monotonically decreasing in nature on increasing $n$.

## 3.2 Solution by Binary-Search

```
def f(l)
    sum := 0
    for i = 1... n:
        sum := sum + a[i]/l
    return sum

l := 1, r: = 1e9+1
while r - l > 1
    mid := l + (r-l)/2
    if f(mid) < k
        l = mid
    else
        r = mid
```

```
for i := l..r
    if (i/a) * (i/b) >= k
    return i
```

**Note that there is always an answer since $l = 1$ and $\sum_{i=1}^{n} \mathbf{a}_i$ is always greater than k.**

# Chapter 4

# Solve $f(x) = C$, $f(x)$ is an monotonic function

Consider $f(x$ is monotonically increases. You are to find the root of the equation $f(x) = C$ on the segment $[l, r]$.

## 4.1 Implementation

```
dbl binary_search(dbl c){
    dbl l = 0.001, r = 100.000;
    FOR(_, 0, 100){
        dbl mid = (l + r)/2;
        if(f(mid) < c) l = mid;
        else r = mid;
    }
    return min(l, r);
}
```

# Chapter 5

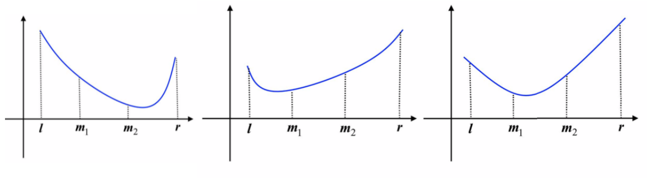
# Ternary Search

You are given a uni-modal function (convex or concave function) $f(x)$. Find the minimum or maximum value of $f(x), x\epsilon[l, r]$.

Calculate $m_1$ and $m_2$ by

$$m_1 := l + (r - l)/3, m_2 := r - (r - l)/3$$

The idea is if $f(m_1) > f(m_2)$ then you can be sure that the minimum will not be existing in the interval $[l, m1)$ and vice-versa. This kind of reduction in the search space is useful as it reduces the length of search space by a multiplicative



factor of 0.66 on every iteration.

## 5.1 Implementation

```
for 200 times:
    m1 := (l + (r−l)/3)\\ should be a double
    m2 := (r − (r−l)/3)\\ should be a double
    if(f(m1) > f(m2)), then
        l := m1
    else
        r := m2
```

**Note that 200 iterations are enough for having a precise solution.**

## 5.2 Important Notes

1. Inverse the sign to find the maximum of the convex function.
2. Sum of convex function is also a convex function.

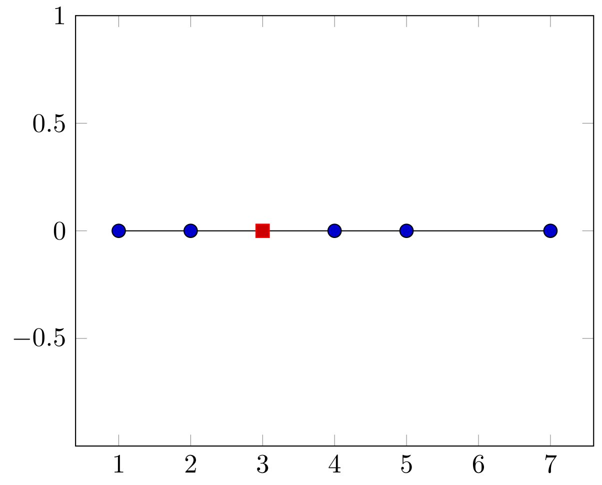
Notice that if $m_1$ and $m_2$ are made close enough without effecting our property that our minimum exists between $[l, r]$ our search space reduction will reduce much faster.

```
for 200 times:
    m1 := (l + (r-l)*0.45)\\ should be a double
    m2 := (r - (r-l)*0.45)\\ should be a double
    if(f(m1) > f(m2)), then
        l := m1
    else
        r := m2
```

# Chapter 6

# Supermarket Location Problem

Given $n$ weighted points on the axis $Ox$, find the point to minimize weighted sum of distances.

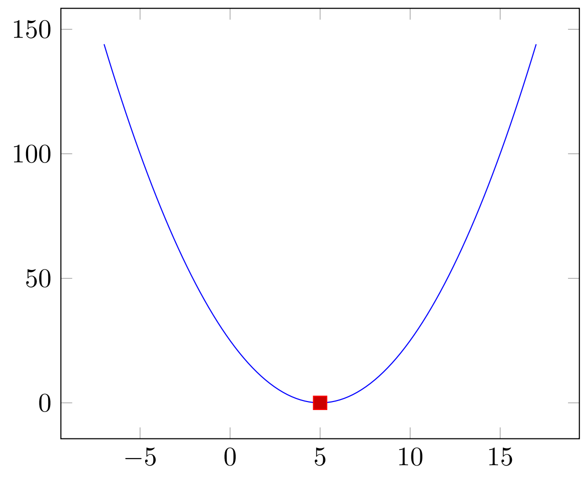


Mathematically, we are required to find k such that:

$$f(k) = \sum_{i=1}^{n} w_i * (|x_i - k|)$$

and solution is

$$MIN[f(k)]$$



## 6.1 Analysis

**Its difficult to determine the solution for n points but its simple for a single ith point.**

$$g(k) = w_i * |k - x_i|$$

As you can see below the position of the target point is overlapping on the $x_i$ to get the $g(x_i) = 0$ minimum.

Another important observation is that the $g(k) = w_i * |k - x_i|$ is a convex function. Since $f(x) = \sum g(x)$, $f(x)$ is also a convex function. Since sum of convex functions is also a convex function, $f(x)$ is also a convex function. Simply ternary search over all value of $k \epsilon [x_1, x_n]$ and find the $MIN[f(k)]$.

## 6.2 Implementation

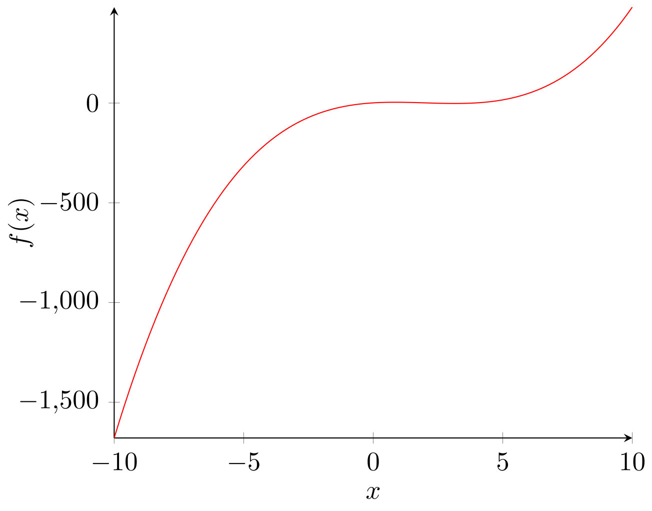
```
def f(k):
    sum := 0
    for i:=1..n:
        sum = sum + w[i]*abs(x[i] - k);
    return sum

for 200 times:
    m1 := (l + (r-l)*0.45)\\ should be a double
    m2 := (r - (r-l)*0.45)\\ should be a double
    if(f(m1) > f(m2)), then
        l := m1
    else
        r := m2
```

# Chapter 7

# Ternary Search: Minimum of a Smooth Function

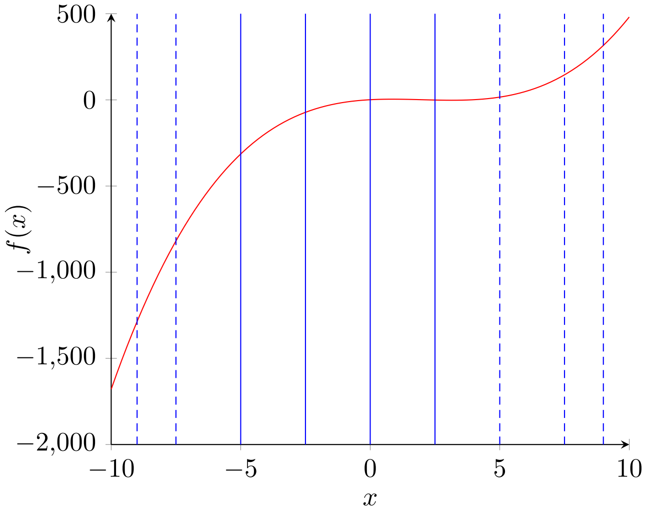
Given a smooth function $f(x)$ on the interval $[l, r]$. Find

$$MIN[f(x)], x \epsilon [l, r]$$



## 7.1 Analysis

Since the $f(x)$ cannot be categorized in a convex or concave function. we decide to split the graph in $m$ small enough sections such that $ith$ section resembles a convex function.



Each local minimum in *ith* section can be found in logarithmic time and since there are m sections , we iterate over each local minimum to find the global minimum in segment $[l, r]$.

# Chapter 8

# Platform Parkour (FHC Round 1)

**You're about to put on an exciting show at your local circus — a parkour demonstration! $N$ platforms with adjustable heights have been set up in a row, and are numbered from $1$ to $N$ in order from left to right. The initial height of platform $i$ is $H_i$ metres.**

**When the show starts, $M$ parkourists will take the stage. The $i^{th}$ parkourist will start at platform $A_i$, with the goal of reaching a different platform $B_i$. If $B_i$ ¿ $A_i$, they'll repeatedly jump to the next platform to their right until they reach $B_i$. If $B_i$ ¡ $A_i$, they'll instead repeatedly jump to the next platform to their left until they reach $B_i$. All of the parkourists will complete their routes simultaneously (but don't worry, they've been trained well to not impede one another).**

**Not all parkourists are equally talented, and there are limits on how far up or down they can jump between successive platforms. The $i^{th}$ parkourist's maximum upwards and downwards jump heights are $U_i$ and $D_i$, respectively. This means that they're only able to move directly from platform $x$ to some adjacent platform $y$ if $H_x - D_i <= H_y <= H_x + U_i$, where $H_x$ and $H_y$ are the current heights of platforms $x$ and $y$, respectively.**

**With the show about to begin, a disastrous flaw has just occurred to you — it may not be possible for all of the parkourists to actually complete their routes with the existing arrangement of platforms! If so, you will need to quickly adjust some of the platforms' heights first. The height of each platform may be adjusted upwards or downwards at a rate of 1 metre per second, to any non-negative real-valued height of your choice, and multiple platforms may be adjusted simul-**

**taneously. As such, if the initial height of platform $i$ is $H_i$ and its final height is $Pi$, then the total time required to make your chosen height adjustments will be $max|Hi - Pi|$ over $i = 1..N$.**

**Determine the minimum amount of time required to set up the platforms such that all $M$ parkourists will then be able to complete their required routes. Note that you may not perform further height adjustments once the show starts. The platform heights must all remain constant while all $M$ parkourists complete their routes.**

**In order to reduce the size of the input data, you're given $H_1$ and $H_2$. $H_3..N$ may then be generated as follows using given constants $W$, $X$, $Y$, and $Z$ (please watch out for integer overflow during this process):**

$$H_i = (W * H_{i-2} + X * H_{i-1} + Y)\%Z\textbf{(for i=3..N)}$$

## 8.1 Analysis

First rule of solving any problems is to express the problem in mathematical expressions and observe if it fits any computational models.

Find $\forall i \in [0, N) H_i^{'} \mid H_i^{'} + MIN(U_k) >= H_{i+1}^{'} \bigwedge H_i^{'} - MIN(D_k) <= H_{i+1}^{'}$ such that $MAX(|H_{i+1} - H_i|)$ is minimum, so we are minimizing some maximum. (Greedy or binary search comes to mind).

Observation: Let x be maximum amount of change any platform can have then $H_i^{'} = [H_i - x, H_i + x] \bigwedge H_{i+1}^{'} = [H_{i+1} - x, H_{i+1} + x]$ then our expression converts to:

Find $x \ \forall i \in [0, N) \mid H_i + x + MIN(U_k) >= H_{i+1} - x \bigwedge H_i - x - MIN(D_k) <= H_{i+1} + x$. Equivalently,

Find $x \mid [H_i - x - MIN(D_k), H_i + x + MIN(U_k)] \cap [H_{i+1} - x, H_{i+1} + x] \neq \emptyset$ (Intersection of segments).

Observation: Increasing $x$ (maximum allowed change) will not decrease the chance of above equation to be true. **Increasing $x$ will never make the equation false.**

The above equation can be interpreted as a binary function $f(x) := [H_i - x - MIN(D_k), H_i + x + MIN(U_k)] \cap [H_{i+1} - x, H_{i+1} + x] \neq \emptyset$ which is monotonic in $x$. Hence we can binary search $x$ and find the minimum $x$ such that the function $f$ is not false.

Some other important note: maximum allowed change($x$) can be 0.

## 8.2 Algorithm

```
int H[INF], U[INF], D[INF]; // U[] and D[] are U min and D min respectively for

bool inter(double l1, double r1, double l2, double r2){
        if(max(l1, l2) <= min(r1, r2)) return 1;
        return 0;
}

bool check(double x){
        FOR(i, 0, N-1)
                if(! inter((double)(H[i] - x - D),(double) (H[i] + x + U), (doub
                        return 0;
        return 1;
}

double binary(){
        double lo = 0, hi = 5e6;
        FOR(_, 0, 100){
                double mi= (lo + hi)>>1;
                if(!check(mi))
                        lo = mi + 1;
                else
                        hi = mi - 1;
        }
        if(!check(lo) && check(hi)) return hi;
        return lo;
}
```

**This is the major idea and rest of Implementation is left to readers.**