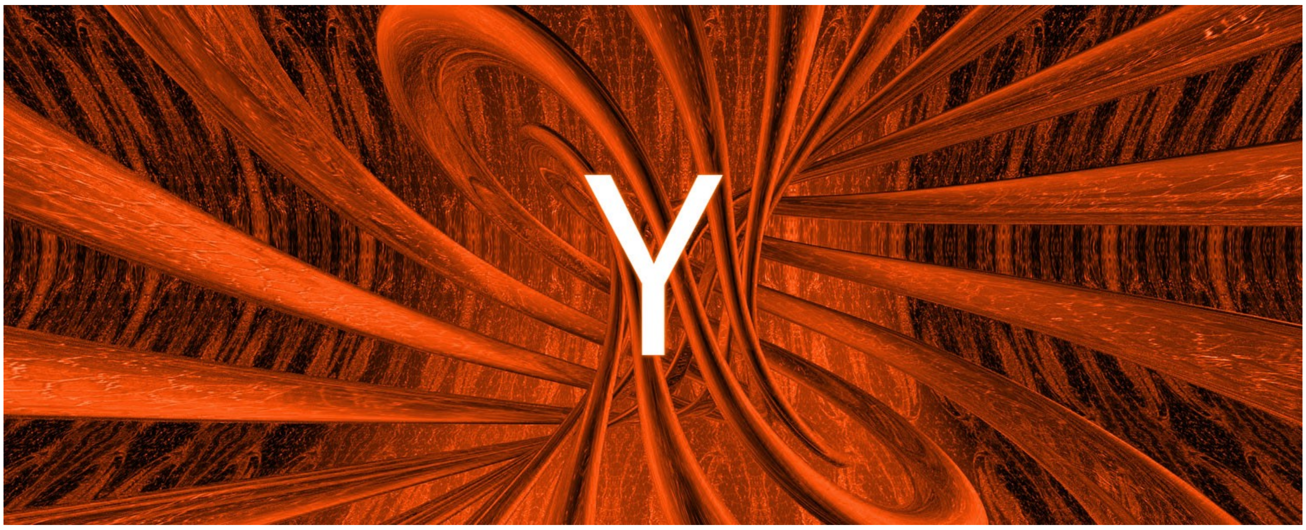


How Hacker News ranking algorithm works

In this post I'll try to explain how the Hacker News ranking algorithm works and how you can reuse it in your own applications. It's a very simple ranking algorithm and works surprising well when you want to highlight hot or new stuff.

Digging into news.arc code

Hacker News is implemented in Arc, a Lisp dialect coded by Paul Graham. Hacker News is open source and the code can be found at arclanguage.org. Digging through the news.arc code you can find the ranking algorithm which looks like this:

```
; Votes divided by the age in hours to the gravityth power.
; Would be interesting to scale gravity in a slider.

(= gravity* 1.8 timebase* 120 front-threshold* 1
   nourl-factor* .4 lightweight-factor* .3 )

(def frontpage-rank (s (o scorefn realscore) (o gravity gravity*))
  (* (/ (let base (- (scorefn s) 1)
          (if (> base 0) (expt base .8) base))
        (expt (/ (+ (item-age s) timebase*) 60) gravity))
     (if (no (in s!type 'story 'poll))  1
         (blank s!url)                  nourl-factor*
         (lightweight s)                (min lightweight-factor*
                                             (contro-factor s))
                                        (contro-factor s))))
```

In essence the ranking performed by Hacker News looks like this:

```
Score = (P-1) / (T+2)^G

where,
P = points of an item (and -1 is to negate submitters vote)
T = time since submission (in hours)
G = Gravity, defaults to 1.8 in news.arc
```

As you see the algorithm is rather trivial to implement. In the upcoming section we'll see how the algorithm behaves.

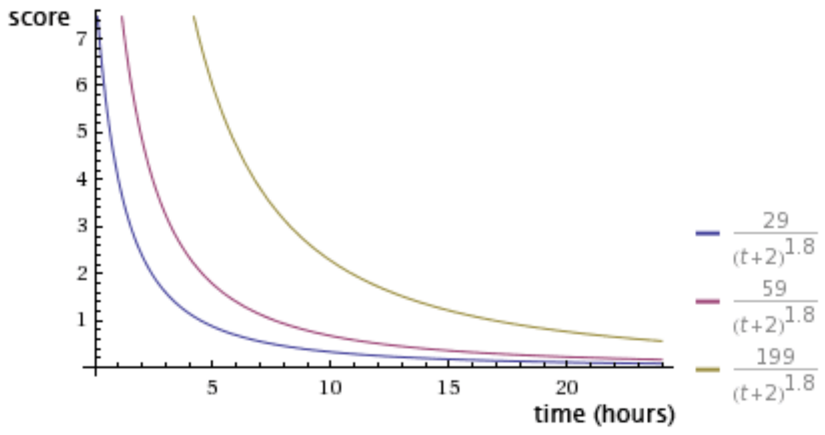
Effects of gravity (G) and time (T)

Gravity and time have a significant impact on the score of an item. Generally these things hold true:

> the score decreases as T increases, meaning that older items will get lower and lower scores
>
> the score decreases much faster for older items if gravity is increased

To see this visually we can plot the algorithm to Wolfram Alpha.

How score is behaving over time

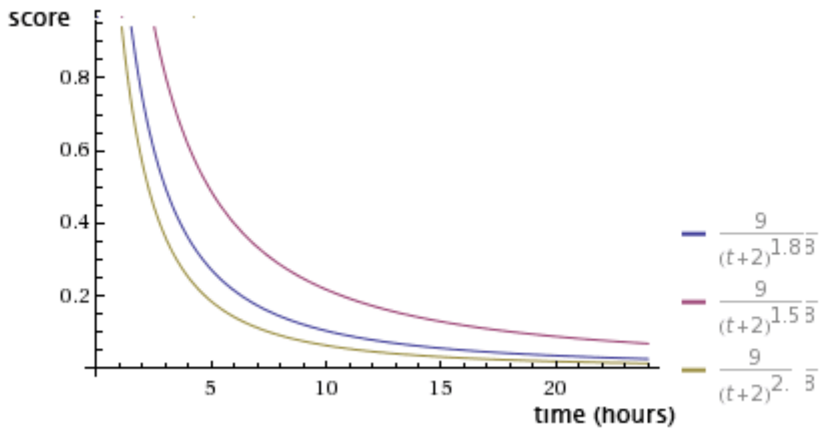


As you can see the score decreases a lot as time goes by, for example a 24 hour old item will have a very low score regardless of how many votes it got.

Plotting query:

```
plot(
    (30 - 1) / (t + 2)^1.8,
    (60 - 1) / (t + 2)^1.8,
    (200 - 1) / (t + 2)^1.8
) where t=0..24
```

How gravity parameter behaves



As you can see by the graph the score decreases a lot faster the larger the gravity is.

Plotting query:

```
plot(
    (p - 1) / (t + 2)^1.8,
    (p - 1) / (t + 2)^0.5,
    (p - 1) / (t + 2)^2.0
) where t=0..24, p=10
```

Python implementation

As already stated it's rather simple to implementing the score function. Here's a implementation in Python:

```
def calculate_score(votes, item_hour_age, gravity=1.8):
    return (votes - 1) / pow((item_hour_age+2), gravity)
```

The most crucial aspect is understanding how the algorithm behaves and how you can customize it for your application and I hope I have contributed that knowledge :-)

· · ·

You can view comments to this post and a lot more thoughts on HN's ranking here:

- http://news.ycombinator.com/item?id=1781013

Paul Graham has shared the updated HN ranking algorithm:

```
(= gravity* 1.8 timebase* 120 front-threshold* 1
       nourl-factor* .4 lightweight-factor* .17 gag-factor* .1)

    (def frontpage-rank (s (o scorefn realscore) (o gravity
gravity*))
      (* (/ (let base (- (scorefn s) 1)
              (if (> base 0) (expt base .8) base))
            (expt (/ (+ (item-age s) timebase*) 60) gravity))
         (if (no (in s!type 'story 'poll))  .8
             (blank s!url)                  nourl-factor*
```

```
                (mem 'bury s!keys)            .001
                                              (* (contro-factor s)
                                                 (if (mem 'gag
     s!keys)
                                                      gag-factor*
                                                     (lightweight s)
                                                      lightweight-
     factor*
                                                     1)))))
```

. . .

Originally published at amix.dk.