

# Skip Lists

Kristian Guillaumier

# Skip lists

- We want to keep a collection of items, and want to efficiently search, insert and delete in it.
- We could use:
  - Arrays: but they are not dynamic.
  - Self balancing trees (e.g., B-Trees, RBTs, AVL Trees).
- In 1989 the skip list data structure was proposed by William Pugh.
- The skip list is simpler to understand and implement than self-balancing trees, and it is very efficient.
- It is also a good example of a ‘probabilistic’ data structure.

# ...continued

- A skip list is based on a linked list.
- A basic linked list has a problem:
  - We don't have random access – in other words even though it might be sorted we cannot search in logarithmic time like an array.
- A skip list is a randomised (we'll explain later) data structure that solves the above problem.

# Efficiency

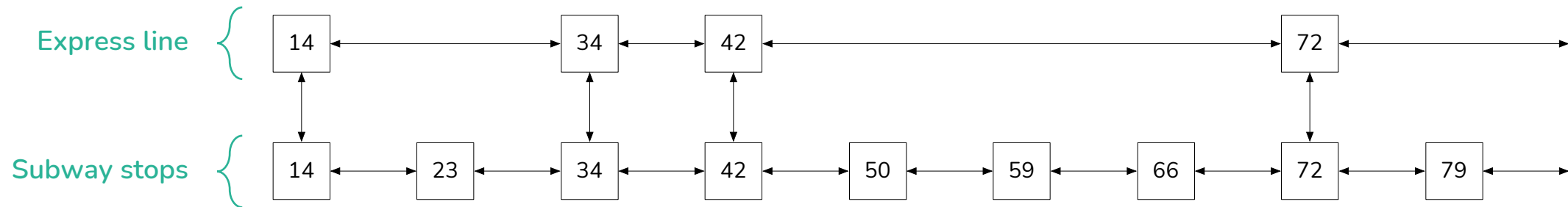
- In the previous data structures, we have seen, we have mentioned bounds (e.g.,  $O(\log_2 n)$  worst case for RBTs) on the performance.
- The performance of Skip Lists is **in expectation**:
  - i.e., we expect  $O(\log_2 n)$  performance – note that this observation is not as powerful as an actual bound.
- But...
  - We will prove that it is  $O(\log_2 n)$  **with a very high probability**.
  - This will be true for every operation.

# Skip lists

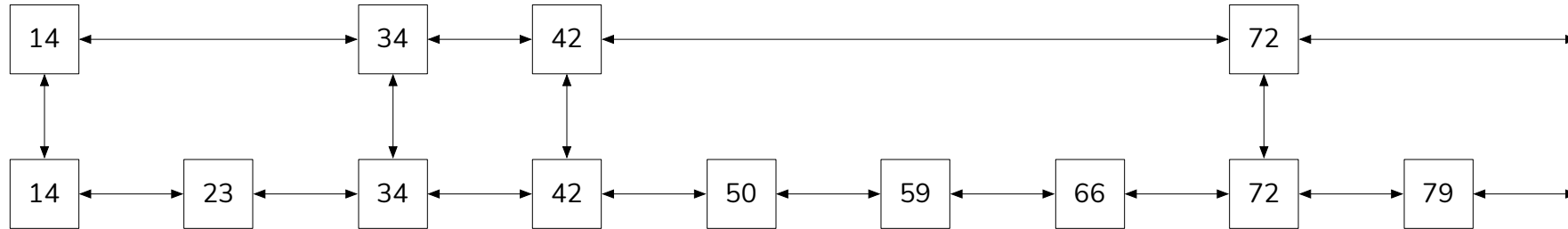
- The basis of the algorithm is a linked list.
- We assume that the list is sorted – we'll make sure of this in the insert operation.
- Remember that we still cannot do a binary search on a linked list even if it is sorted.

# Illustration

- Overview of the method...
- These are street numbers with the following the subway stops (7<sup>th</sup> avenue line).
- The subway has an express line (this makes it a skip list).



# A simple example



Note how the 'local' line has all the elements.

The express line has a subset.

If we want to go to 59:

1. We take the express.
2. At 42, we are told that the next station is 72 (i.e., greater than the 59 we are looking for).
3. We get off the express.
4. We get on the local line and continue.

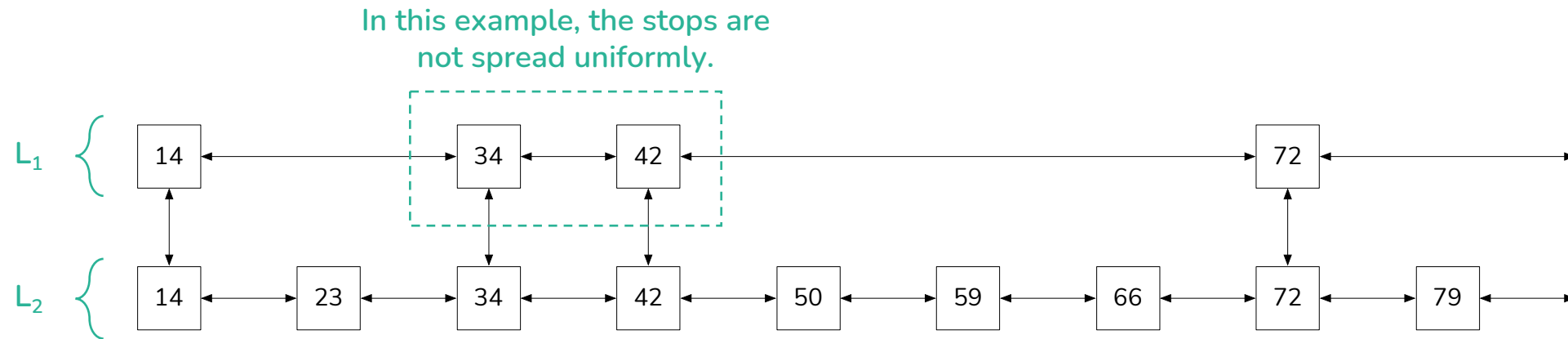
# Overview of a search

- Init: Let  $L$  = topmost list.
- Move right in  $L$  until we either find what we are looking for and return it, or the next value is greater than we are looking for.
- If the next value is greater than we are looking for:
  1. If this is the bottom list, then the item is not found.
  2. Otherwise, move to the next lower list.
- Restart.



# Skip lists

- If the lists are drawn from top to bottom (where bottom is the full list), what do we put in  $L_1$ ?
- If this is a subway, we would put the most popular stops in it, but, in general, we do not have this kind of 'domain' knowledge.
- Ideally, we would **spread the stops out uniformly**.



# Uniformly spread

Assuming  $|L_1|$  is half  $|L_2|$ :

L1: 1 ↔ 3 ↔ 5 ↔ 7 ↔ 9  
↕            ↕            ↕            ↕            ↕

L2: 1 ↔ 2 ↔ 3 ↔ 4 ↔ 5 ↔ 6 ↔ 7 ↔ 8 ↔ 9 ↔ 10

Another uniform spread:

L1: 1            ↔            5  
↕                            ↕

L2: 1 ↔ 2 ↔ 3 ↔ 4 ↔ 5 ↔ 6 ↔ 7 ↔ 8 ↔ 9 ↔ 10

Both are uniform spreads. Is one better than another?

# Some analysis

- What is the maximum number of steps if we have two lists  $L_1$  and  $L_2$  in my skip list?
- If we spread out uniformly, in the worst-case scenario (last item) we would have to:
  - Go through  $|L_1|$  steps.
  - Go down to  $|L_2|$  and go through  $\frac{|L_2|}{|L_1|}$  steps.
  - In total we have  $|L_1| + \frac{|L_2|}{|L_1|}$  steps.
  - We want to minimise this.

# ...continued

- $|L_2|$  (the bottom list) is  $n$ , where  $n$  is the number of elements in the data structure.
- To minimize, we can only choose the size of  $L_1$  since  $n$  is not a choice:

$$|L_1| + \frac{n}{|L_1|}$$

- This formula will be smaller when:

$$|L_1| + \frac{n}{|L_1|}$$

  
 $L_1$  is smaller     $L_1$  is larger

  
We must find a balance

# ...continued

- So, we must balance:

$$\underbrace{|L_1|}_{\text{This half}} + \underbrace{\frac{n}{|L_1|}}_{\text{And this half}}$$

- The formula is minimised when the 2 'halves' are the same:
  - $|L_1| = \frac{n}{|L_1|}$
  - Rearranged...
  - $|L_1|^2 = n$
  - $|L_1| = \sqrt{n}$

# ...continued

- Let's work out some examples for  $n = 150$  items:

- Let's try  $|L_1| = 2$ :

- The formula  $|L_1| + \frac{n}{|L_1|} = 2 + \frac{150}{2} = 77$

- Let's try  $|L_1| = 50$ :

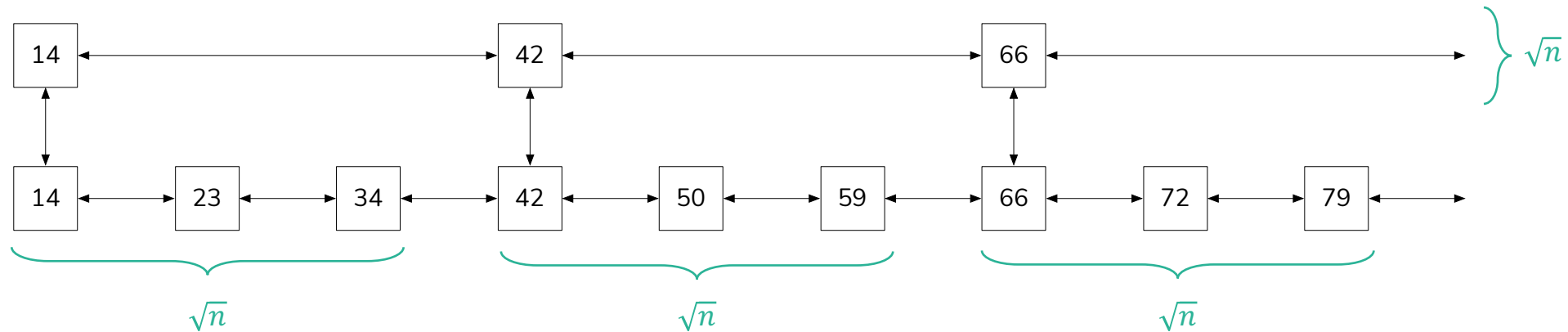
- The formula  $|L_1| + \frac{n}{|L_1|} = 50 + \frac{150}{50} = 53$

- Let's try  $|L_1| = \sqrt{150} = 12.25$ :

- The formula  $|L_1| + \frac{n}{|L_1|} = 12.25 + \frac{150}{12.25} = 24.5$

# ...continued

- Since the two 'halves' of  $|L_1| + \frac{n}{|L_1|}$  are the same and each half is  $\sqrt{n}$  then the search cost is  $\sqrt{n} + \sqrt{n} = 2\sqrt{n}$
- So, a normal linked list has a  $n$  'time performance' and skip list with two lists gives me  $2\sqrt{n}$  'time performance' – better but not as good as it can get.



## ...continued

- $O(\sqrt{n})$  is better than  $O(n)$  but still not nearly as good as  $O(\log_2 n)$  time.
- The solution is to add a third 'express line' and a fourth, and fifth, and...
- So...
  - For two lists:  $2 \times \sqrt{n}$
  - For three lists:  $3 \times \sqrt[3]{n}$
  - For  $k$  lists:  $k \times \sqrt[k]{n}$



$$k \times \sqrt[k]{n}$$

- Consider two lists. What is the search cost?

$$\text{Search cost} = L1 + \frac{n}{L1}$$

$$\text{a} \quad L1 = \frac{n}{L1} \Rightarrow L1 = \sqrt{n}$$

$$\sqrt{n} + \sqrt{n} = 2 \times \sqrt{n}$$

But all 'parts' are equal

- Consider three lists. What is the search cost?

$$\text{Search cost} = L2 + \frac{L1}{L2} + \frac{n}{L1}$$

$$\text{a} \quad L2 = \frac{L1}{L2} \Rightarrow L2^2 = L1$$

$$\text{b} \quad L2 = \frac{n}{L1} \Rightarrow L2 = \frac{n}{L2^2} \Rightarrow L2^3 = n \Rightarrow L2 = \sqrt[3]{n}$$

$$\sqrt[3]{n} + \sqrt[3]{n} + \sqrt[3]{n} = 3 \times \sqrt[3]{n}$$

But all 'parts' are equal

$$k \times \sqrt[k]{n}$$

- Consider four lists. What is the search cost?

$$\text{Search cost} = L3 + \frac{L2}{L3} + \frac{L1}{L2} + \frac{n}{L1}$$

a  $L3 = \frac{L2}{L3} \Rightarrow L3^2 = L2$

b  $L3 = \frac{L1}{L2} \Rightarrow L3 = \frac{L1}{L3^2} \Rightarrow L3^3 = L1$

c  $L3 = \frac{n}{L1} \Rightarrow L3 = \frac{n}{L3^3} \Rightarrow L3^4 = n \Rightarrow L3 = \sqrt[4]{n}$

$$\sqrt[4]{n} + \sqrt[4]{n} + \sqrt[4]{n} + \sqrt[4]{n} = 4 \times \sqrt[4]{n}$$

But all 'parts' are equal

- We can go on making this argument for  $k$ .

## ...continued

- The question now is, how many lists  $k$  do we need?
- We **don't want  $k$  to be  $n$**  otherwise we will have  $n$  levels and we will need to 'fall down' to a lower level  $n$  times and end up with  $O(n)$ .
- We **don't want  $k$  to be too small** otherwise we will have 'root' time which is not as good as  $\text{Log}_2$  time.
- We want  $k$  to be logarithmic to have **logarithmic depth**.
- So, for  $\log_2 n$  lists, we get a search performance of  $\log_2(n) \times^{\log_2(n)} \sqrt[n]{n}$ .

# ...continued

- Breaking down  $\log_2(n) \times \sqrt[\log_2(n)]{n}$

- We know the  $y^{\text{th}}$  root of a number  $n$  is  $\sqrt[y]{n} = n^{\frac{1}{y}}$

- So...

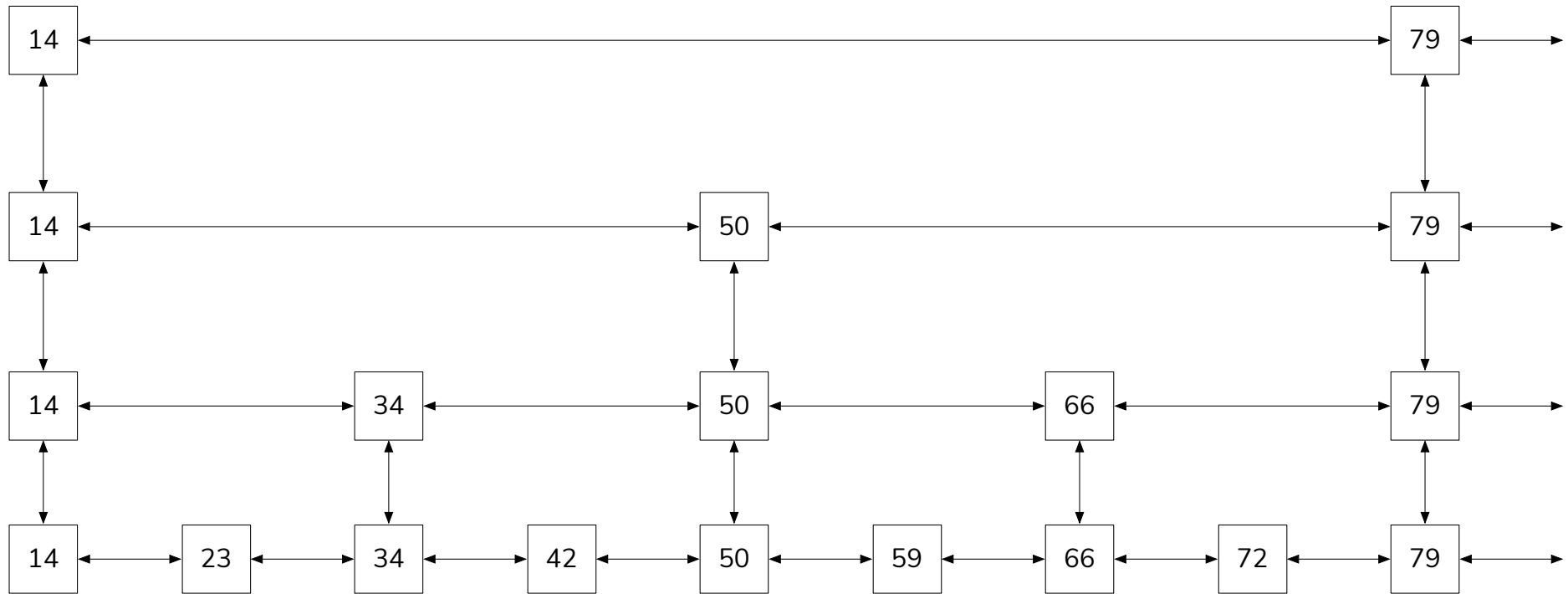
- $\log_2(n) \times \sqrt[\log_2(n)]{n} = \log_2(n) \times n^{\frac{1}{\log_2(n)}}$

Remember that  $a^b = 2^{b \times \log_2(a)}$

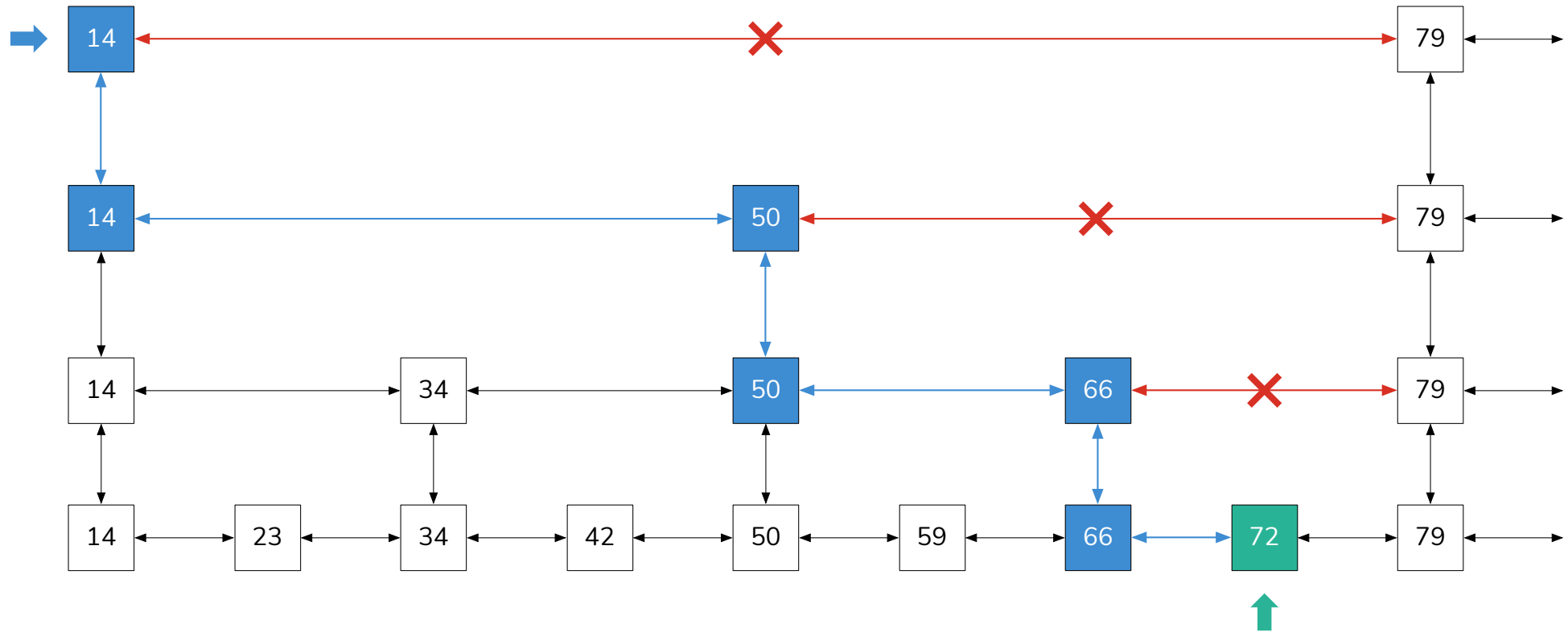
$$n^{\frac{1}{\log_2(n)}} = 2^{\frac{1}{\log_2(n)} \times \log_2(n)} = 2^1 = 2$$

- Giving  $2 \times \log_2(n)$
- So, for  $\log_2(n)$  lists, we need to perform  $2 \times \log_2(n)$  search steps – logarithmic!

# A simple search example – find 72

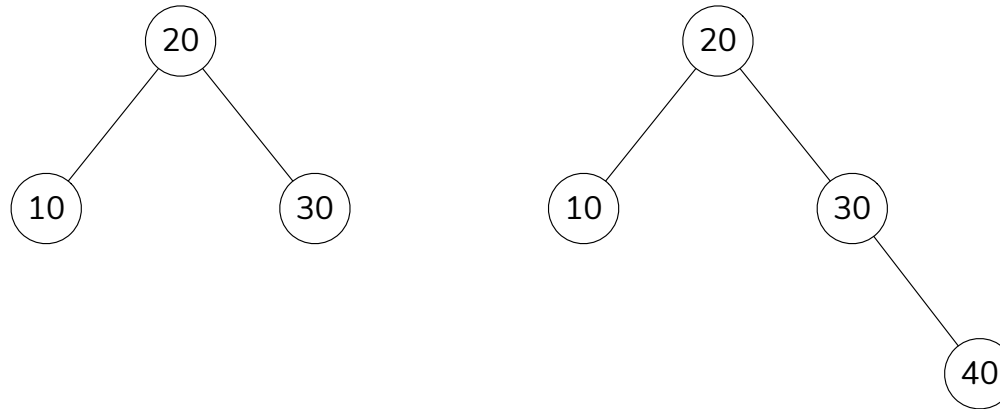


# A simple search example – find 72



# Skip lists

- The skip list in the previous illustration we have seen is an **ideal skip list**. This is like the concept of an ideal BST – insertions and deletions will ‘corrupt’ this structure.
- Ideal BST and insertion example:

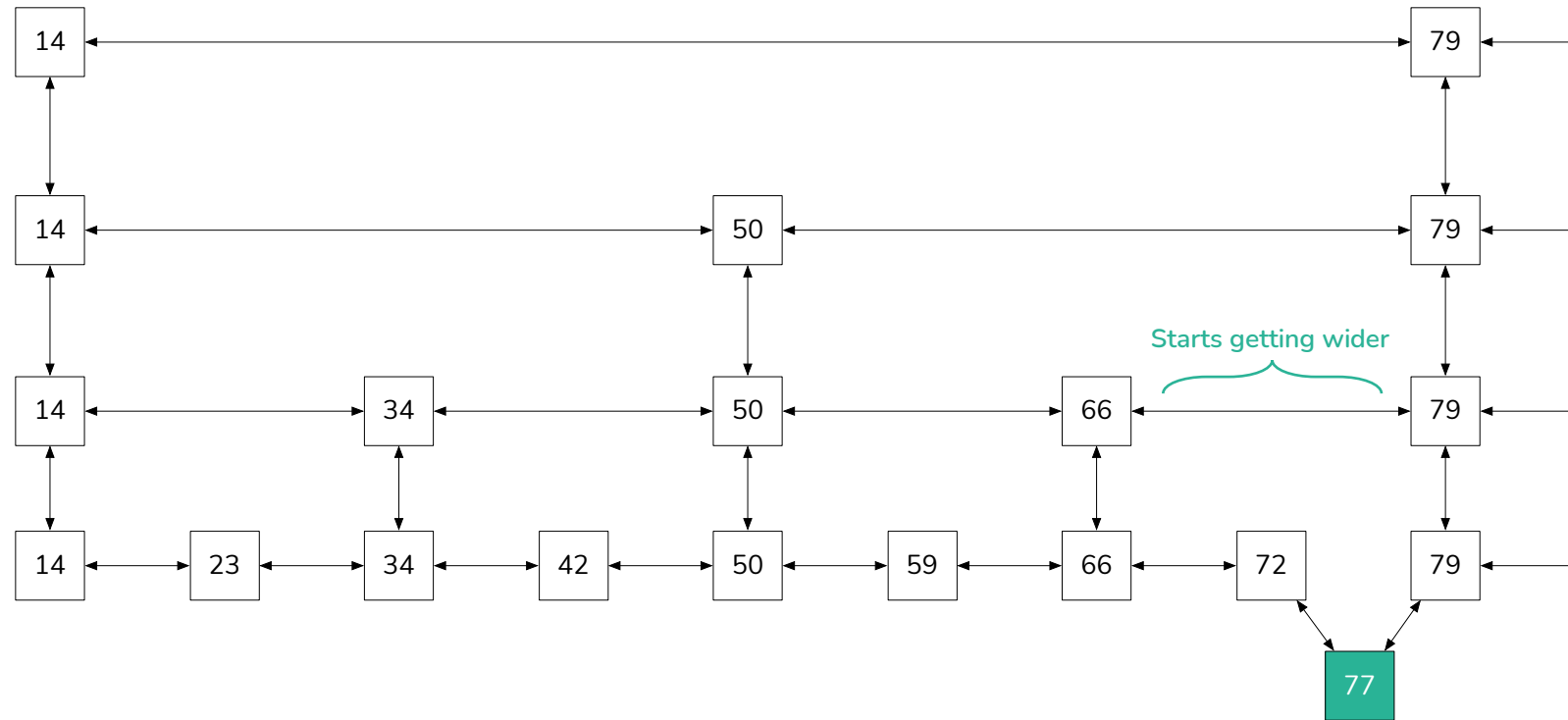


# Insertion

- As usual, we run a search to find where the item should fit in the **bottom list**.
- Remember that, by definition, the bottom list contains all the elements.
- Once we find the place in the bottom list, we insert it (in the bottom list).  
Note that we have now 'de-idealised' our skip list.



# Inserting the value 77

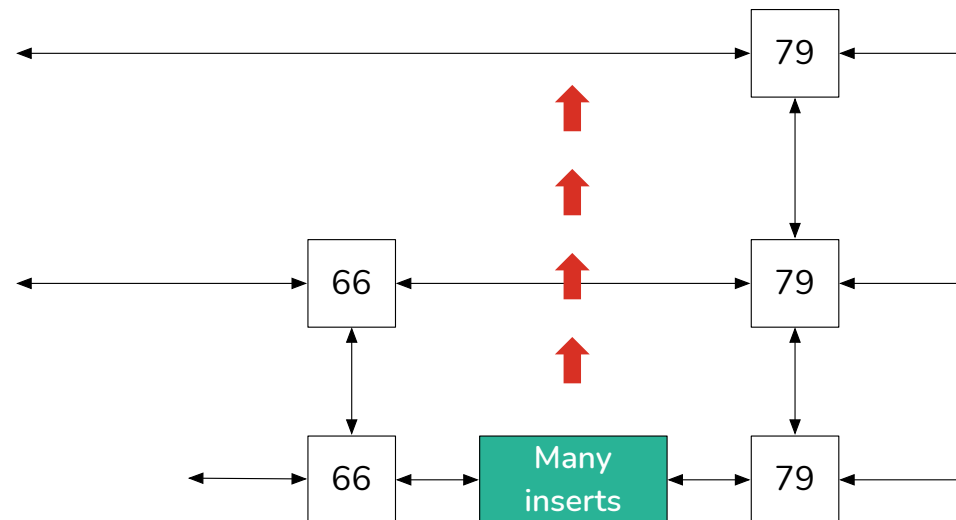


Note how the length of this segment is longer, starting to make things inefficient.

# ...continued

- We can see that after a number of insertions, the 'segment' starts becoming longer and longer compared to the others (the skip list property is damaged).
- Clearly, **some items must move to upper lists** to maintain the skip list 'balance'. **How?**

What condition?



Some of the items we insert need to be promoted to higher levels. Which ones? How?

## ...continued

- Intuitively, some of the elements we insert must be promoted up a level (for each level) to maintain ratio of elements between lists.
- In skip lists, the answer to whether an inserted element gets promoted to an upper level is determined by **flipping a coin** (50/50 chance – half go up).
- Heads = promote up.
- Tails = don't promote.

*We will show that, using this strategy, we can expect our operations to still be  $O(\log_2 n)$ .*

# A small problem when the list is empty

Insert 9. Do we promote? **Toss** → **No**. Good. So far:

[9]

Insert 20:

[9]↔[20]

Do we promote? **Toss** → **yes**:

[20]

↑

[9]↔[20]

Oops... I don't have a top-left from where to start searching:

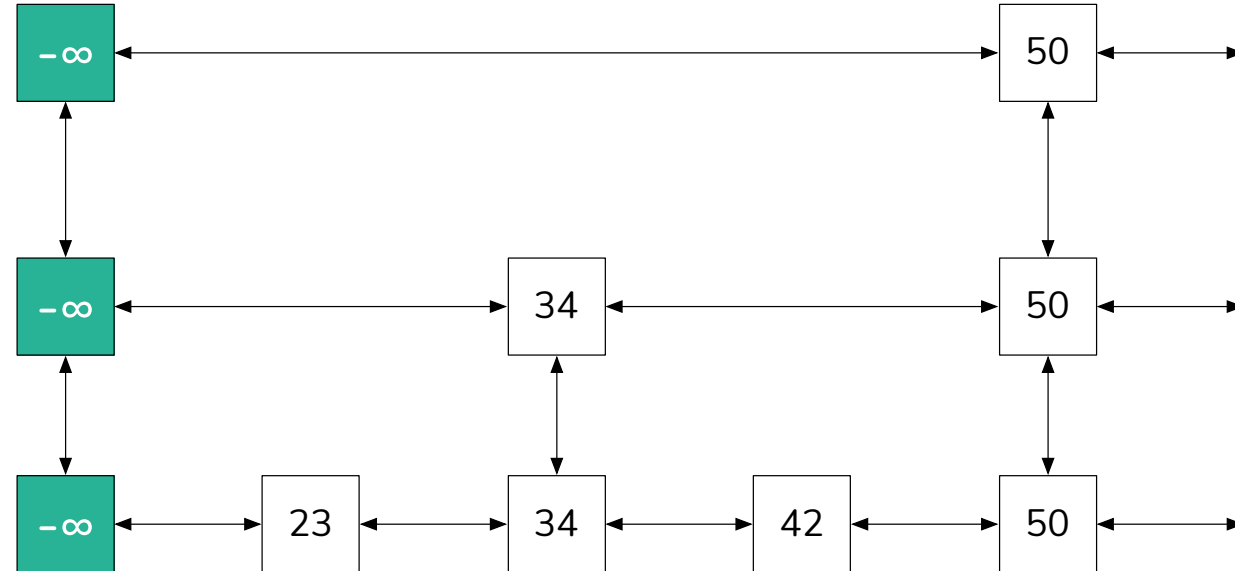
✗ [20]

↑

[9]↔[20]

# Solution

- Add a special value of  $-\infty$  to **each** list:



# Deletion

- Note that deletion is trivial.
- If I want to delete X, **just delete its occurrence from every list.**
- Done.

# Theorem

*With high probability, every search in an  $n$  element skip list costs  $O(\log_2 n)$*

# Preparation: what does WHP mean?

- Remember,  $O(\log_2 n)$  is saying “order  $\log_2(n)$ ”.
- So...
  - $1 \times \log_2(n)$  is order  $\log_2(n)$ .
  - $1.3 \times \log_2(n)$  is order  $\log_2(n)$  as well.
  - $10 \times \log_2(n)$  is order  $\log_2(n)$  as well.
- In other words, if the cost is  $c \times \log_2(n)$  where  $c$  is some constant, we still have “order  $\log_2(n)$ ” performance, which is what we want.



# Preparation: what does WHP mean?

- An event occurs **WHP** if for any number  $\alpha \geq 1$ , the event occurs with probability at least (greater than):

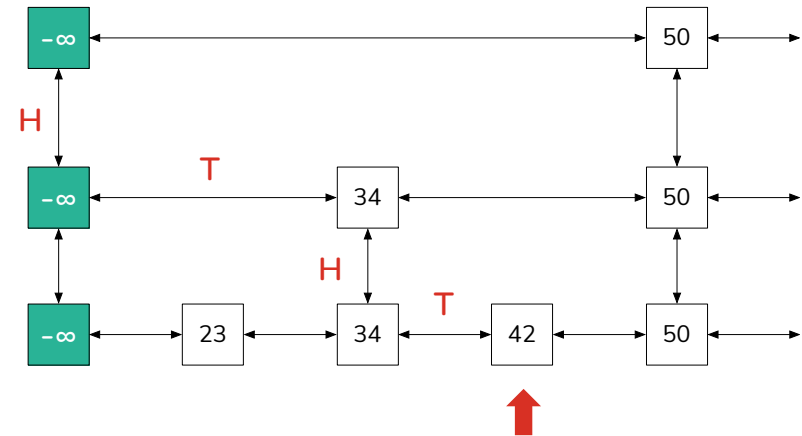
$$1 - o\left(\frac{1}{n^\alpha}\right)$$

- So, if the event E occurs with a probability of at least  $1 - o\left(\frac{1}{n^\alpha}\right)$ , this means that the event **will not** occur with a probability of at most (less than):

$$o\left(\frac{1}{n^\alpha}\right)$$

# Preparation: reasoning about a search

- Suppose that **we analyse a search backward**.
- In a **normal search** we start at top-left  $-\infty$  then move right or down.
- In a **backward search** we...
  - Start at the result.
  - If we can move up, we got heads. Otherwise, if we can move left, we got tails.
  - Finish top-left  $-\infty$ .
  - Think of any search as a string such as “HHHTTHTHTTTTTTH”.



# Preparation: some stats

- Recall that the notation  $\binom{n}{r}$  denotes the number of ways that we can choose  $r$  objects from a set containing  $n$  distinct elements.
- Also,  $\binom{n}{r} = \frac{n!}{r!(n-r)!}$
- So, if a flip a coin 4 times, in how many ways can we get 1 heads?

$$\frac{4!}{1!(4-1)!} = \frac{24}{6} = 4$$

HTTT, THTT, TTHT, TTTH

# Preparation: some stats

- If I flip a coin 4 times, in how many ways I can get 2 heads:

$$\binom{4}{2} = \frac{4!}{2! (4 - 2)!} = \frac{24}{4} = 6$$

HHTT, HTHT, HTTH, THHT, THTH, TTHH

# Preparation: some stats

- Let's say we flip a coin 5 times. What is the probability of getting **exactly** 2 heads?

$$\underbrace{\binom{5}{2}}_{\text{Number of different ways I can get this.}} \times \underbrace{\left(\frac{1}{2}\right)^2}_{\text{Probability of getting 2 heads.}} \times \underbrace{\left(\frac{1}{2}\right)^3}_{\text{Probability of getting 3 tails.}}$$

This is obviously the total number of flips less what we are looking for.

# Preparation: some stats

- Let's say I flip a coin 5 times. What is the probability of getting **at most** 2 heads?

$$\binom{5}{2} \times \cancel{\binom{1}{2}}^2 \times \left(\frac{1}{2}\right)^3$$

$$= \binom{5}{2} \times \left(\frac{1}{2}\right)^3 \quad \left. \vphantom{\binom{5}{2} \times \left(\frac{1}{2}\right)^3} \right\} \text{Another way of saying "getting at least 3 tails".}$$

# Preparation: some stats

- Also note that there is a known bound (we won't prove this here) that says:

$$\binom{n}{r} \leq \left(e \cdot \frac{n}{r}\right)^r$$

$e$  is Euler's number, the base of the natural log, 2.718...

There is a nice explanation here:

[https://en.wikipedia.org/wiki/Binomial\\_coefficient#Bounds\\_and\\_asymptotic\\_formulas](https://en.wikipedia.org/wiki/Binomial_coefficient#Bounds_and_asymptotic_formulas)

# Back to our proof, our strategy will be...

- First, we will show that the number of levels in a skip list is bound by  $O(\log_2 n)$ .
- This means that in any search there will not be any more than  $O(\log_2 n)$  down moves (or up moves if we're using the backward search idea).
- Which also means that in any sequence “HHTTTHTHH...” there will not be more than  $O(\log_2 n)$  H moves.
- Additionally, by the time we have performed all our H moves, the search is complete.
- We finally need to show that by the time we perform  $O(\log_2 n)$  H moves, the total cost of the search (the length of the H-T sequence) is also  $O(\log_2 n)$ .



# Step 1: proving a bound on the height

- Remember we are trying to prove that all searches in a skip list with  $n$  elements take  $O(\log_2 n)$  steps.
- In BSTs we know that if we can show that the number of levels in the tree is  $O(\log_2 n)$  then we get  $O(\log_2 n)$  search performance.
- Unfortunately, this is not true for skip lists. We could have  $O(\log_2 n)$  levels but it could still be 'unbalanced'.
- An extreme example:

□

□ ↔ □

□ ↔ □ ↔ □

□ ↔ □ ↔ ...a lot of items... ↔ □

Height is bound but search  
is still very inefficient.

# Step 1: proving a bound on the height

- Although the height does not help us establish performance of the search, proving a bound is useful to prove our result.
- So...

*Lemma: WHP a skip list with  $n$  elements will not have more than  $O(\log_2 n)$  levels.*

# Step 1: proving a bound on the height

- Probability of an element going up 1 level is  $\frac{1}{2}$
- Probability of an element going up 2 levels is  $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$
- Probability of an element going up 3 levels is  $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$
- Probability of an element going up  $k$  levels is  $\frac{1}{2} \times \frac{1}{2} \times \cdots \times \frac{1}{2} = \frac{1}{2^k}$
- Probability of an element going up  $c \cdot \log_2 n$  levels is  $\frac{1}{2^{c \cdot \log_2 n}}$

# Step 1: proving a bound on the height

- So, the probability of an element going up **more** than  $c \cdot \log_2 n$  levels is  $< \frac{1}{2^{c \cdot \log_2 n}}$ .
- But we are not dealing with only one element – we are inserting  $n$  elements.
- So, the probability **of any one** of the  $n$  elements going up more than  $c \times \log_2 n$  levels is:

$$< n \times \frac{1}{2^{c \cdot \log_2 n}}$$

$$< n \times \frac{1}{n^c}$$

$$< \frac{n}{n^c}$$

$$< \frac{1}{n^{c-1}}$$

So, the probability is polynomially small.  
The larger we make  $c$  the less probability  
there is it will go beyond  $c \cdot \log_2 n$

$$c - 1 = \alpha$$

## Step 2: final proof

*With high probability, every search in an  $n$  element skip list costs  $O(\log_2 n)$*

To show this, we will use our ‘backwards’ search idea...

## Step 2: final proof

- From our lemma we know that we cannot make more than  $c \cdot \log_2 n$  ‘up’ moves (WHP).
- So, we have a bound on the number of ‘up’ moves.
- Remember that in all the flips we make, we can never (WHP) generate more than  $c \cdot \log_2 n$  heads – we can never go beyond the highest level.

*So, we want to show that WHP the total number of flips until we get  $c \cdot \log_2 n$  heads is  $O(\log_2 n)$*

## Step 2: final proof

- Let's say that we make  $10 \cdot c \cdot \log_2 n$  total flips. What is the probability that we get exactly  $c \cdot \log_2 n$  heads?

$$\underbrace{\binom{10 \cdot c \cdot \log_2 n}{c \cdot \log_2 n}}_{\text{Number of different ways I can get this.}} \times \underbrace{\left(\frac{1}{2}\right)^{c \cdot \log_2 n}}_{\text{Probability of getting heads.}} \times \underbrace{\left(\frac{1}{2}\right)^{9 \cdot c \cdot \log_2 n}}_{\text{Probability of getting tails.}}$$

## Step 2: final proof

- So, after making  $10 \cdot c \cdot \log_2 n$  total flips ...

$$\Pr(\text{exactly } c \cdot \log_2 n \text{ heads}) = \binom{10 \cdot c \cdot \log_2 n}{c \cdot \log_2 n} \times \left(\frac{1}{2}\right)^{c \cdot \log_2 n} \times \left(\frac{1}{2}\right)^{9 \cdot c \cdot \log_2 n}$$

- Then...

$$\Pr(\text{at most } c \cdot \log_2 n \text{ heads}) = \binom{10 \cdot c \cdot \log_2 n}{c \cdot \log_2 n} \times \cancel{\left(\frac{1}{2}\right)^{c \cdot \log_2 n}} \times \left(\frac{1}{2}\right)^{9 \cdot c \cdot \log_2 n}$$



## Step 2: final proof

- Now, we rearrange a bit using  $\binom{n}{r} \leq \left(e \cdot \frac{n}{r}\right)^r$  } We discussed this earlier in our preparation

- We fix:

$$\Pr(\text{at most } c \cdot \log_2 n \text{ heads}) = \binom{10 \cdot c \cdot \log_2 n}{c \cdot \log_2 n} \times \left(\frac{1}{2}\right)^{9 \cdot c \cdot \log_2 n}$$

- To get:

$$\Pr(\text{at most } c \cdot \log_2 n \text{ heads}) \leq \left(e \times \frac{10 \cdot c \cdot \log_2 n}{c \cdot \log_2 n}\right)^{c \cdot \log_2 n} \times \left(\frac{1}{2}\right)^{9 \cdot c \cdot \log_2 n}$$

## Step 2: final proof

$$\leq \left( e \times \frac{10.c.\log_2 n}{c.\log_2 n} \right)^{c.\log_2 n} \times \left( \frac{1}{2} \right)^{9.c.\log_2 n}$$

$\left( \frac{1}{2} \right)^{9.c.\log_2 n} = \frac{1^{9.c.\log_2 n}}{2^{9.c.\log_2 n}} = \frac{1}{2^{9.c.\log_2 n}}$

$$\leq (10.e)^{c.\log_2 n} \times \frac{1}{2^{9.c.\log_2 n}}$$

$$\leq \frac{(10.e)^{c.\log_2(n)}}{2^{9.c.\log_2 n}}$$


Remember that  $a^b = 2^{b \times \log_2(a)}$

$$(10.e)^{c.\log_2 n} = 2^{\log_2(10.e) c.\log_2(n)}$$


$$\leq \frac{2^{\log_2(10.e).c.\log_2(n)}}{2^{9.c.\log_2(n)}}$$

# Step 2: final proof

$$\leq \frac{2^{\log_2(10e) \cdot c \cdot \log_2(n)}}{2^{9 \cdot c \cdot \log_2 n}}$$


$$\leq 2^{(\log_2(10e) - 9) \cdot c \cdot \log_2(n)}$$

$$\leq 2^{-\alpha \cdot \log_2 n}$$

$$\leq \frac{1}{2^{\alpha \cdot \log_2 n}}$$


$$\leq \frac{1}{n^\alpha}$$

Note that as the 10 approaches  $\infty$ , then the 9 increases **linearly** but the  $\log_2(10e)$  increases **logarithmically** (more slowly), so it is:

- Negative
- $(\log_2(10e) - 9)$  goes to  $-\infty$

Remember that  $a^b = 2^{b \times \log_2(a)}$

$$2^{\alpha \cdot \log_2 n} = n^\alpha$$

## Step 2: final proof

- So, we have shown that if we perform  $Q \cdot c \cdot \log_2 n$  flips in total, the probability of getting  $\leq c \cdot \log_2 n$  heads is incredibly small...

$$\Pr(\text{at most } c \cdot \log_2 n \text{ heads}) \leq \frac{1}{n^\alpha}$$

- Which means that by the time we perform  $Q \cdot c \cdot \log_2 n = O(\log_2 n)$  search steps to find a result, we would have performed  $c \cdot \log_2 n = O(\log_2 n)$  up moves, and by lemma 1, we cannot perform more than that.

# Recap

- An event  $E$  occurs WHP if probability is  $\geq 1 - o\left(\frac{1}{n^\alpha}\right)$ .
- An event  $E$  occurs WLP if probability is  $< o\left(\frac{1}{n^\alpha}\right)$ .

## ...continued

- The probability of an element reaching  $c \cdot \log_2 n$  levels is  $\frac{1}{2^{c \cdot \log_2 n}}$
- The probability of any of my  $n$  elements reaching  $c \cdot \log_2 n$  levels is  $\leq \frac{1}{2^{c \cdot \log_2 n}} + \dots + \frac{1}{2^{c \cdot \log_2 n}}$  for  $n$  times (by Boole's inequality).
- Rearranged:
  - $\leq \frac{n}{2^{c \cdot \log_2 n}}$
  - $\leq \frac{1}{n^{c-1}}$

Where  $\alpha = c - 1$ . So, the probability of any one of our  $n$  elements reaching  $c \cdot \log_2 n$  levels is very small (within our definition of WLP).

## ...continued

- A search is performed from top-left down to bottom-right.
- We visualise the **search in reverse**.
- In reverse, we can:
  - **Move to the left**: i.e., we had gotten a tails for that element.
  - **Move up**: i.e., we had gotten a heads for that element.
- So, a search is a sequence HHTTTHHTH...

## ...continued

- We have shown that WHP we cannot make more than  $c \cdot \log_2 n$  up moves.
- Remember that the probability of any of our  $n$  elements going up more than  $c \cdot \log_2 n$  levels is  $< \frac{1}{n^\alpha}$
- So, in any search HHTTTTHHTH... we will not have more than  $c \cdot \log_2 n$  “H’s”.
- Clearly, the cost of a search is the length of a ‘string’ of HHTTTTHHTH... which is equivalent to the total number of flips made to insert that item.
- Clearly,  $\#Flips = \#Heads + \#Tails$  (and so far, we have a bound on  $\#heads$ ).



## ...continued

- Suppose that I start flipping coins.
- At some point, I will have flipped  $c \cdot \log_2 n$  heads.
- At this point, I cannot flip any more heads.
- That is, by now I have performed the total number of flips.
- I need to show that this total number of flips (cost of the search) is  $O(\log_2 n)$ .

## ...continued

- Let's say that we performed  $O(\log_2 n)$  flips.
- What is the probability of getting at most  $(\leq) c \cdot \log_2 n$  heads?
- We have shown that this probability is  $\leq \frac{1}{n^\alpha}$
- In other words,  $\Pr(\text{getting} \leq c \cdot \log_2 n \text{ heads}) \leq \frac{1}{n^\alpha}$
- The implication is: if we flip  $O(\log_2 n)$  times, the probability of getting less than  $O(\log_2 n)$  heads is negligible.

# Finally...

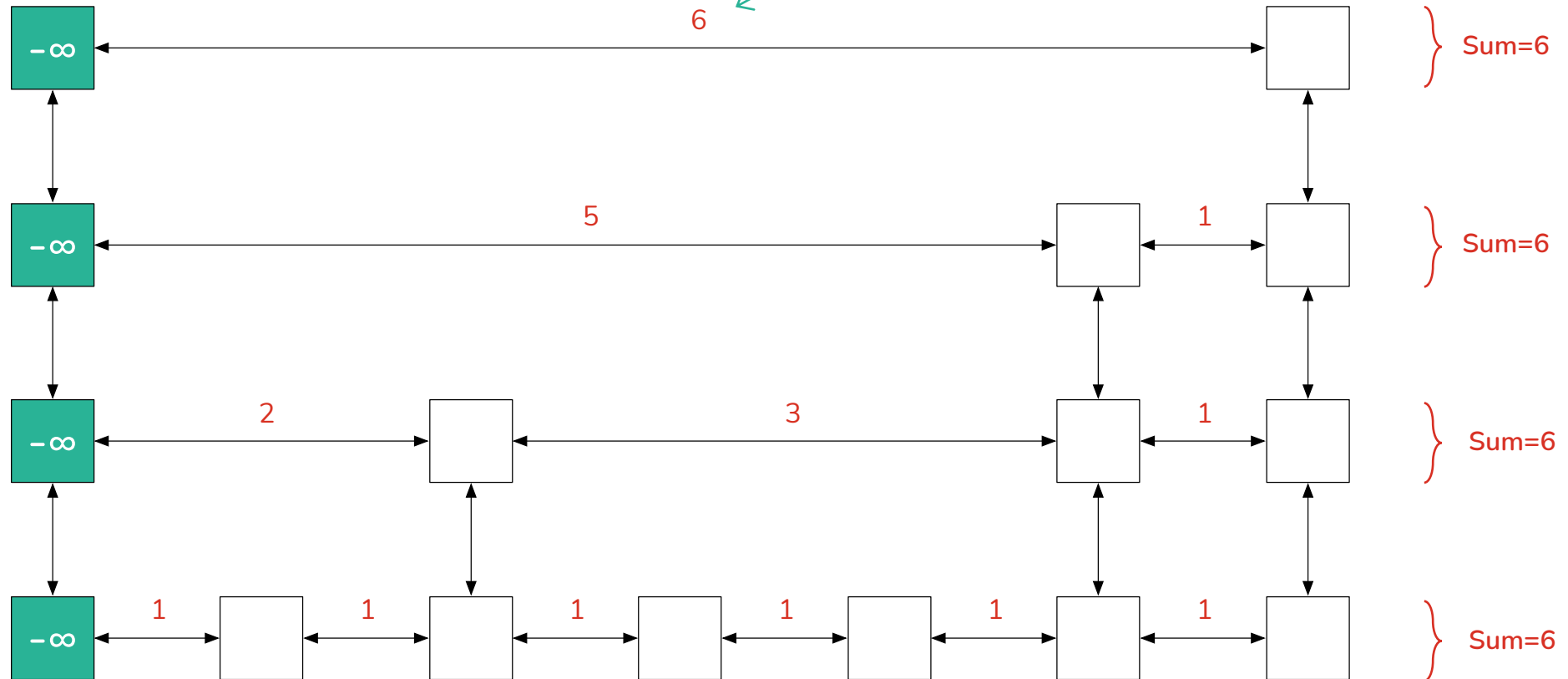
- By the time we flip  $O(\log_2 n)$  heads the search is complete.
- By the time we flip  $O(\log_2 n)$  heads, we have performed  $O(\log_2 n)$  total flips WHP.
- We cannot perform more than  $O(\log_2 n)$  heads WHP.
- So, we cannot perform more than  $O(\log_2 n)$  total flips.

# Indexable skip lists

- So far, our skip lists can do insertions and deletions in  $O(\log_2 n)$ .
- However, **lookups at an index** (accessing the  $i^{\text{th}}$  element) still requires  $O(n)$  time.
- This can be easily remedied by storing the ‘**width**’ of a link for every link. This way we can get  $O(\log_2 n)$  performance to look up at an index.
- Consider the next example...

# ...continued

The width of a link is the sum of the link widths beneath it.



Note how every link has a width

# Accessing the $i^{\text{th}}$ element

- To access the  $i^{\text{th}}$  element, we simply traverse the list summing the widths of each traversed ‘horizontal’ link but **go down a level** if the adding a link width results in a value greater than  $i$ .
- When a new item is inserted, the widths can be efficiently updated as the skip list is traversed during the insert.
- See the technical document called “**A skip list cookbook**” by William Pugh for implementation details.

# Further reading

- These notes should be supplemented by:
  - Introduction to Algorithms (Clifford Stein, Thomas H Cormen, Ronald L Rivest, Charles E Leiserson – MIT Press)
- There are very good notes on skip lists:
  - [http://videolectures.net/mit6046jf05\\_demaine\\_lec12/](http://videolectures.net/mit6046jf05_demaine_lec12/)
- Original skip list paper:
  - <ftp://ftp.cs.umd.edu/pub/skipLists/skiplists.pdf>