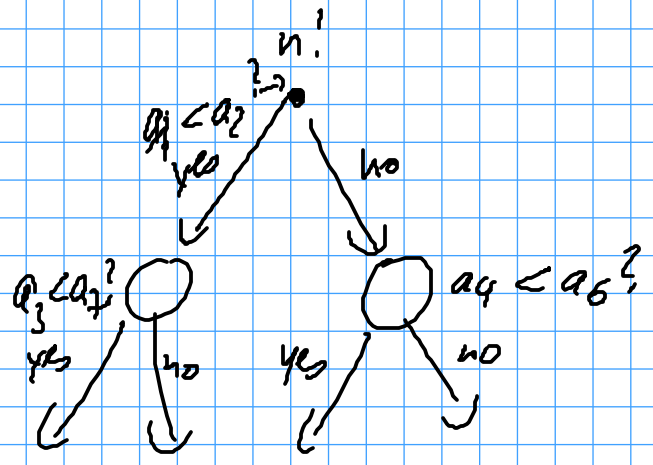
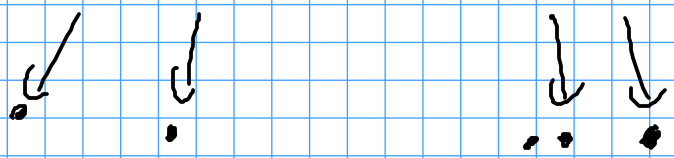
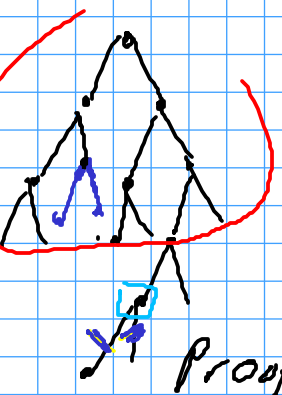
SORTING LOWER BOUNDS





Lemma: A deterministic comp.-based sorting alg. needs $\Omega(n \log n)$ comparisons on average.

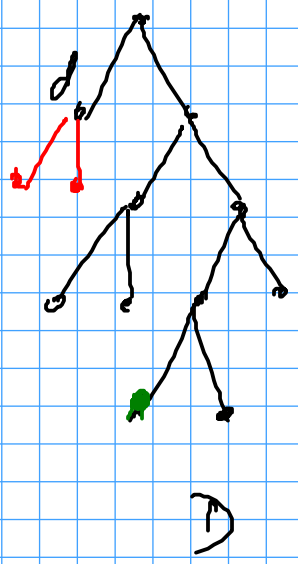
$\Leftrightarrow$ avg. depth of one input in the decision tree is $\lfloor \log_2(n) \rfloor$



Proof. Completely balanced tree: all leaves have a depth of $\lfloor \log_2(n) \rfloor$ or $\lceil \log_2(n) \rceil$

Now consider an unbalanced tree

- max depth D
- min. depth d of a leave
- average avg



Remove two siblings at maximum depth and relocate them to a leave at min. depth.

$$avg' = \frac{avg \cdot n! - 2D + 2(d+1) + D - 1 - d}{n!}$$

$$= \frac{avg \cdot n! - D + d + 1}{n!}$$

$$D - d \geq 2$$

$$\Rightarrow avg' \leq \frac{avg \cdot n! - 1}{n!} < avg$$

$\Rightarrow$ an unbalanced can always be modified such that the average depth of the leaves decrease

$\Rightarrow$ a balanced tree optimizes the average depth

$\Rightarrow$ on average we need for a det. comp.-based alg. $n \cdot \lceil \log_2(n) \rceil$ many questions to know the input permutation

$\Rightarrow$ $\Omega(n \log n)$ comp. necessary on average

Randomized Sorting Lower Bound

Lemma: For comp.-based randomized sorting algorithms we need to perform expectedly $\Omega(n \log n)$ comparisons to determine the correct result.

Proof: Think of a run of a rand. algorithm as u.e.r. choice of one deterministic alg. out of the set of all possible det.-algorithms

Expected runtime:

$$T(n) = \frac{1}{n!} \sum_{\text{inputs } I} \sum_{S \in \mathcal{S}} P(S) \, (\text{runtime of } A_S \text{ on } I)$$

$\mathcal{S}$: sequence of questions and decision tree

A_S: det. Alg. corresponding to S

$$= \sum_{S \in \mathcal{S}} P(S) \underbrace{\frac{1}{n!} \sum_{I} (\text{runtime of } A_S \text{ on } I)}_{\text{avg. runtime of a det. alg.} \geq n \lfloor \log_2(n) \rfloor \text{ (last lemma)}}$$

$$T(n) \geq \sum_{s \in S} p(s)\, n \lfloor \log_2(n) \rfloor$$

$$= n \lfloor \log_2(n) \rfloor \underbrace{\sum_{s \in S} p(s)}_{\uparrow}$$

$$= n \lfloor \log_2(n) \rfloor \quad \in \Omega(n \log n) \qquad //$$

Exercise: Implement a det. sorting alg. which runs in $O(n \log n)$ and QuickSort. Let both alg. run on a set of 10 Mio numbers, measure the runtime and the number of comparisons. Run QuickSort 100 times on the same input and have a look at the distributions of runtime / # comp.

Search

Given a set of numbers (e.g. represented as an array), we want to find the position of a certain element in that array or get the information that the element is not contained.

$A[n]$, look for q

naive: single scan over A, stop if q is found
→ costs us at most $O(n)$

→ even in a sorted linked list, complexity would not change

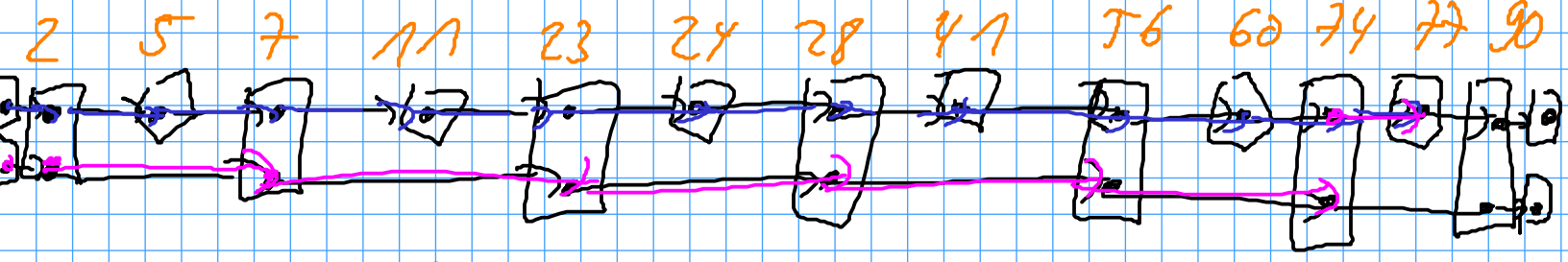
better: AVL-trees, red-black trees etc. → identify an element in $O(\log n)$

→ same time for insertion and deletion

even better: randomized Skip Lists

2.3.1 Def. Skip List

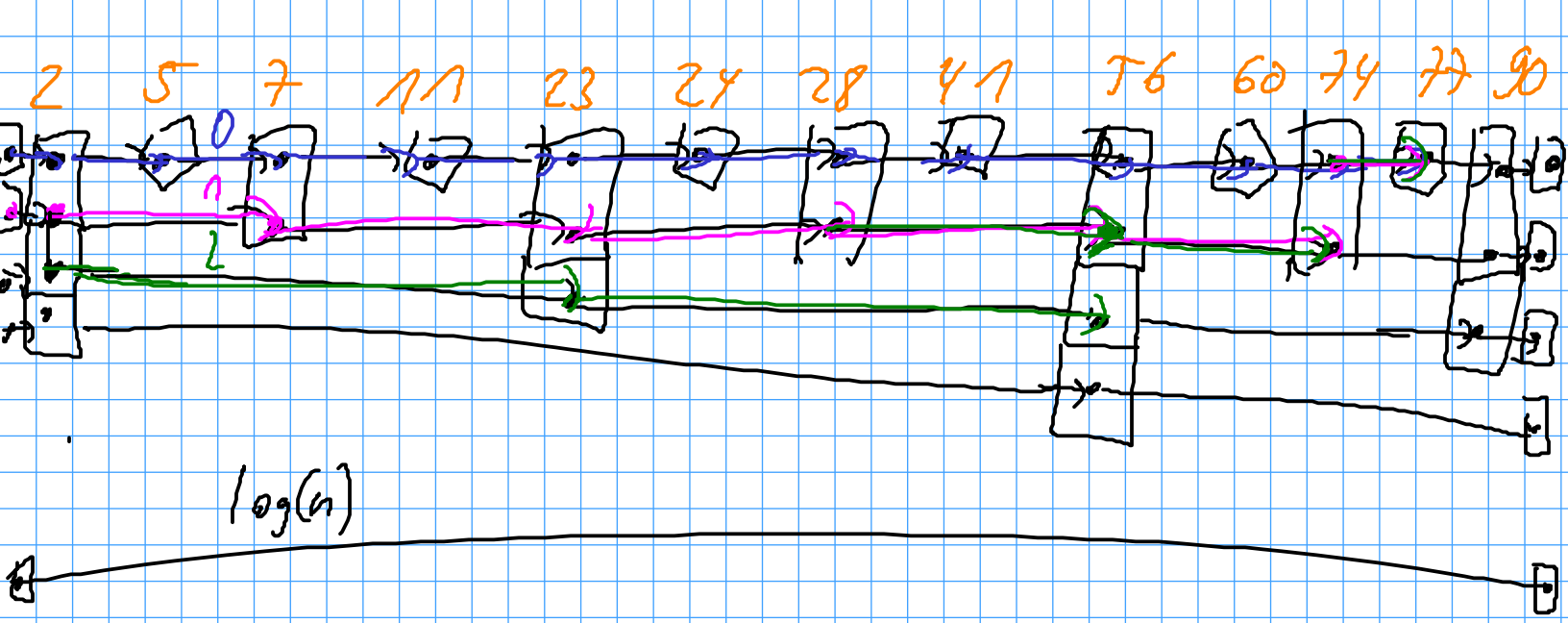
77



Simple linked list: at most n elements to consider

linked list +

Skip list for $i=2$: at most $\frac{n}{2}+1$



at most $\log(n)$ layers until the lowest skip list is empty if the number of elements is halved in each round

Lemma: Det. SkipLists exhibit a search time of $O(\log n)$.

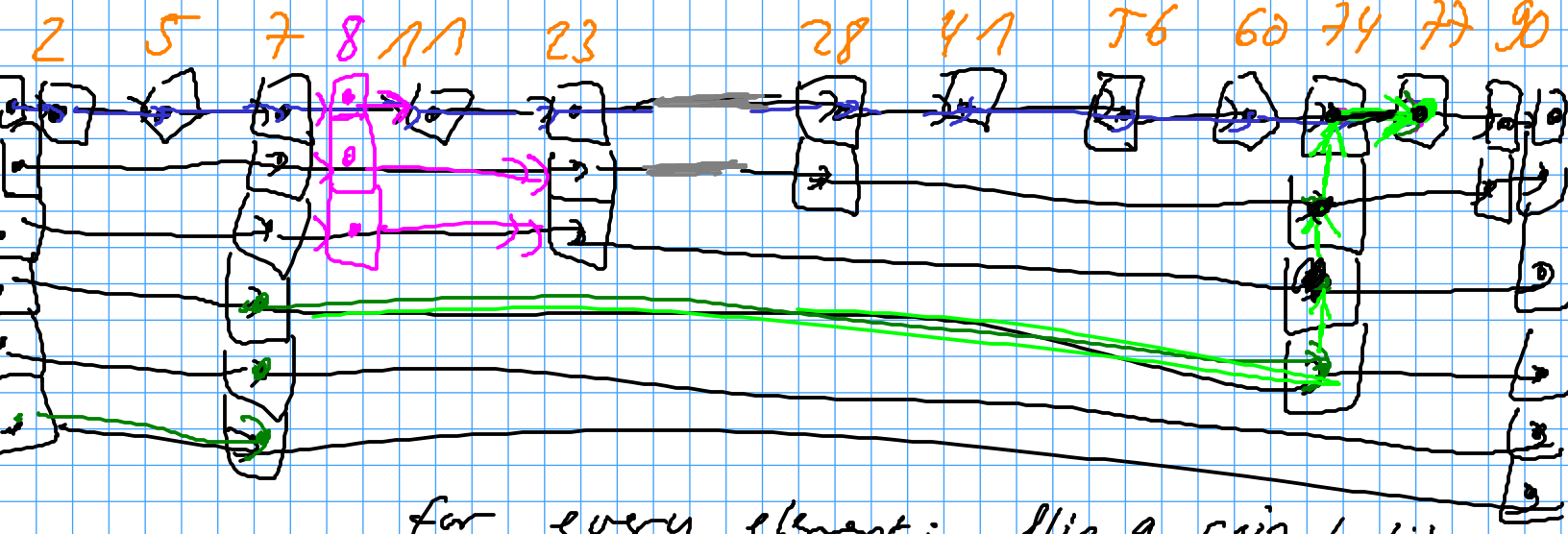
Proof: If we change from list i to list $i-1$, the next element in i is too large (or the tail). Hence in list $i-1$ we can only go one step to the right. $\Rightarrow$ constant time per list $\Rightarrow$ $O(\log n)$.

Search time is fine. But insertion and deletion could cost $O(n)$. Because

a new element can force almost the whole structure to be rebuild.

2.3.2. Randomized SkipLists

Construction: simple sorted linked list



for every element: flip a coin until it shows head

⇒ number of flips until head denotes the number of lists the element appears in

Lemma: Rand. Skip Lists exhibit an expected search time of $O(\log n)$.

Proof What is the expected number of lists we construct? In level 0 we have n elements. Every element survives

and is present in level 1 with a probability of $\frac{1}{2}$. Hence expectedly we have $\frac{n}{2}$ elements in layer 1. Accordingly $\frac{n}{4}$ elements in layer 2.

.....

Therefore in layer $> \log n$ we expect 0 elements to survive.

Backwards analysis: Start at the element we search for. and backtrack the search path. We want to count the number of lines backwards.

Observe, in the reverse path, we will always take a step up if we can (so switch to the next higher level). Otherwise we make a step to the left.

The probability for each step is $\frac{1}{2}$ as it is just determined by a single coin flip.

X_i denote the number of steps we need to walk through i levels.

$$E(X_i) = 1 + \frac{1}{2} E(X_{i-1}) + \frac{1}{2} E(X_i)$$

We expand the term to compute the value for $E(X_i)$ recursively.

$$E(X_i) = 2 + E(X_{i-1})$$

$$= 2 + 2 + E(X_{i-2}) = 4 + E(X_{i-2})$$

$$= 6 + E(X_{i-3}) = \dots$$

$$= 2j + E(X_{i-j})$$

$$= 2i$$

As i corresponds to the level, we have an expected search time of at most $E(X_{\log n}) = 2 \log n \in O(\log n)$ //

Exercise: Analyse the (expected) space consumption of det. and rand. Skip Lists.

So in contrast to det. Skip Lists insertions / deletions are expecting $O(\log n)$ operations as well.

Why?

Insertion:
- find the position — $O(\log n)$
- flip a coin until it is head is reached — exp. $O(\log n)$
- relocate links that cross that element — exp. $O(\log n)$
  $\Rightarrow$ at most $\log n$

$O(\log n)$

Deletion:
- find the element — $O(\log n)$
- delete it from all lists — $O(\log n)$
- bridge all interrupted links — $O(\log n)$

$O(\log n)$