CSSE 230 Day 25

Sorting Lower Bound
Radix Sort
Skip Lists

Reminders/Announcements

- **Exam** is Thursday evening
- Complete the **EditorTrees partner evaluation** today
- Before the exam, **copy your team's EditorTreesMilestone2 project** to your individual CSSE 230 repository
  - Team > Update
  - Team > Disconnect
    - Before you press the Yes button, choose "Also Delete SVN metadata"
  - Team > Share Project > SVN > Next, choose your repo
  - Team>Commit
    - Just to be sure everything is there.

Tuesday – Thursday classes

- WA 8 due at 8 AM Tuesday
- I'll take up to one class period to answer your questions related to the exam
  - Same format as the Wednesday Q&A session I did before the first exam.
- The last programming project will be introduced, along with some background material needed to do it.
- Because of the exam Thursday evening, **no class meeting Thursday afternoon.**

Questions?

A Lower-Bound on Sorting Time

- We can't do much better than what we already know how to do.

What’s the best best case?

- Lower bound for best case?
- A particular algorithm that achieves this?

What's the best worst case?

- Want a function f(N) such that the **worst case running time** for all sorting algorithms is Ω(f(N))

- How do we get a handle on "all sorting algorithms"?

Tricky!

What are “all sorting algorithms”?

- We can’t list all sorting algorithms and analyze all of them
  - Why not?

- But we can find a uniform representation of any sorting algorithm that is based on **comparing** elements of the array to each other

This "uniform representation" idea is exploited in a big way in Theory of Computation, e.g., to demonstrate the unsolvability of the "Halting Problem"

First of all…

- The problem of sorting N elements is at least as hard as determining their ordering
  - e.g., determining that $a_3 < a_4 < a_1 < a_5 < a_2$
- So any lower bound on all "order-determination" algorithms is also a lower bound on "all sorting algorithms"

Q1

Sort Decision Trees

- Let A be any **comparison-based algorithm** for sorting an array of distinct elements
- Note: sorting is asymptotically equivalent to determining the correct order of the originals
- We can draw an EBT that corresponds to the comparisons that will be used by A to sort an array of N elements
  - This is called a **sort decision tree**
  - Just a pen-and-paper concept, not actually a data structure
  - Different algorithms will have different trees

So what?

- Minimum number of external nodes in a sort decision tree? (As a function of N)
- Is this number dependent on the algorithm?
- What's the height of the shortest EBT with that many external nodes?

$$\lceil \log N! \rceil \approx N \log N - 1.44N = \Omega(N \log N)$$

No comparison-based sorting algorithm, known or not yet discovered, can **ever** do better than this!

Can we do better than *N* log *N*?

- $\Omega(N \log N)$ is the best we can do if we compare items
- Can we sort without comparing items?

Yes, we can! We can sort if we avoid comparing items

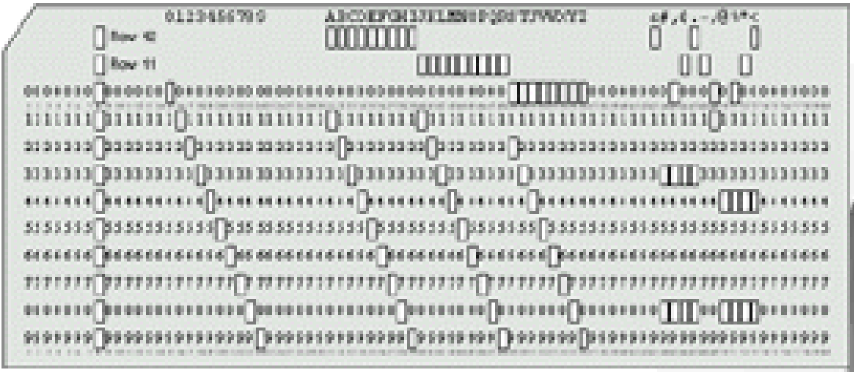
Q5

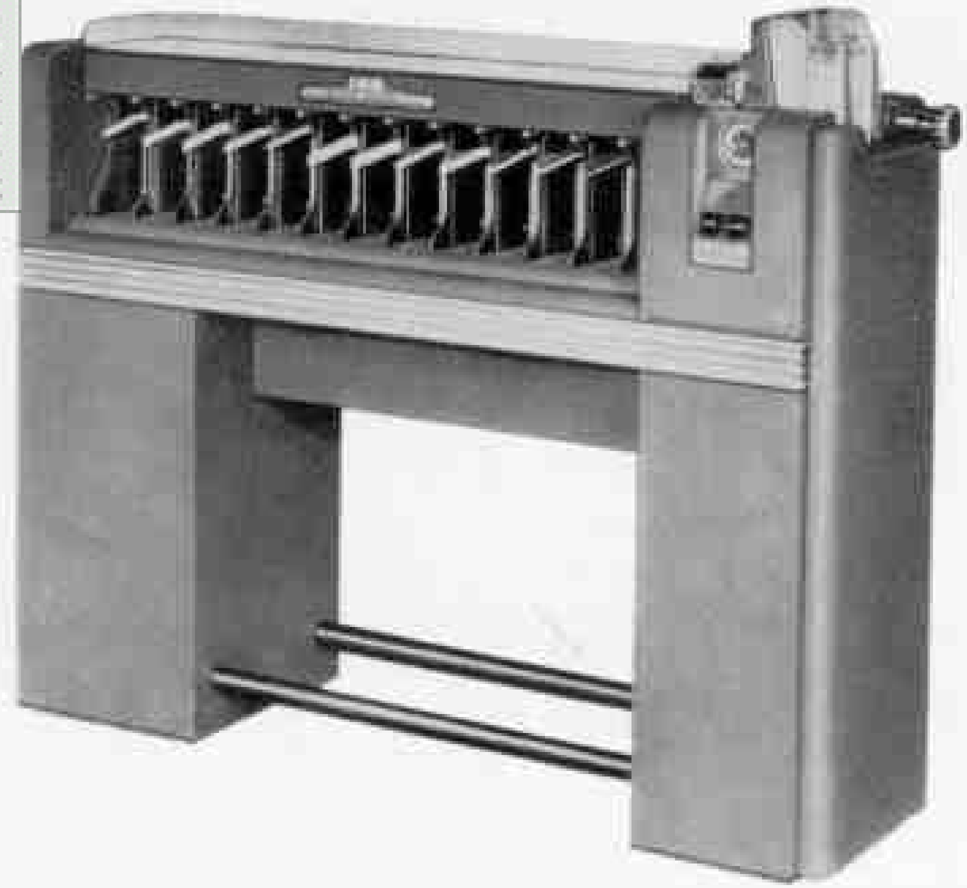
- O(N) sort: Bucket sort
  - Works if possible values come from limited range
  - Example: Exam grades histogram
- A variation: Radix sort

Radix sort

- A picture is worth 10^3 words, but an animation is worth 2^{10} pictures, so we will look at one.
- http://www.cs.auckland.ac.nz/software/AlgAnim/radixsort.html

Radix sort example: card sorter





Type 82 Electric Punched Card Sorting Machine

Used an appropriate combo of mechanical, digital, and human effort to get the job done.

Skip Lists

- An alternative to balanced trees

Q9-11

An alternative to AVL trees

- Indexed lists.
- One-level index.
- 2nd-level index.
- 3rd-level index
- log-n-level index.
- Problem: insertion and deletion.
- Solution: Randomized node height: Skip lists.
  - Pugh, 1990 CACM.
- http://iamwww.unibe.ch/~wenger/DA/SkipList/
- Notice that skip lists do not share with binary trees the problem that threads are designed to solve.

A slightly different skip list representation

- Uses a bit more space, makes the code simpler.
- Michael Goodrich and Roberto Tamassia.

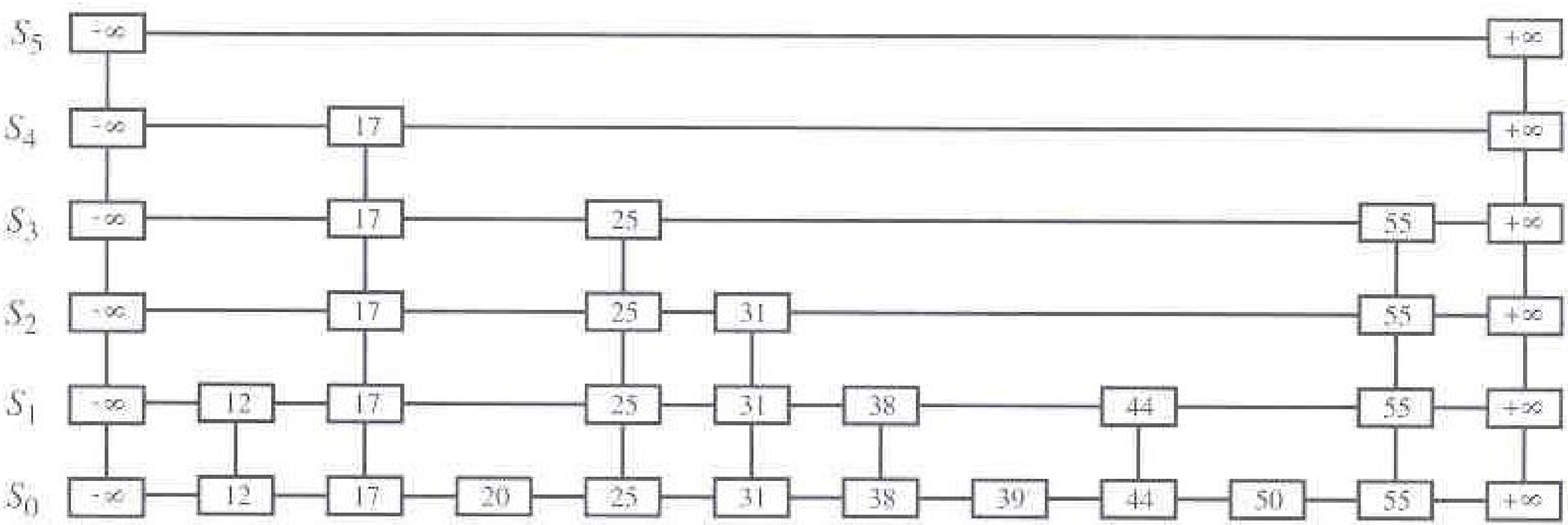


Figure 8.9: Example of a skip list.

Methods in SkipListNode class

after(p): Return the position following p on the same level.

before(p): Return the position preceding p on the same level.

below(p): Return the position below p in the same tower.

above(p): Return the position above p in the same tower.

Search algorithm

1. If S.below(p) is null, then the search terminates—we are ***at the bottom*** and have located the largest item in S with key less than or equal to the search key k. Otherwise, we ***drop down*** to the next lower level in the present tower by setting $p \leftarrow S$.below(p).
2. Starting at position p, we move p forward until it is at the right-most position on the present level such that key(p) $\leq k$. We call this the ***scan forward*** step. Note that such a position always exists, since each level contains the special keys $+\infty$ and $-\infty$. In fact, after we perform the scan forward for this level, p may remain where it started. In any case, we then repeat the previous step.

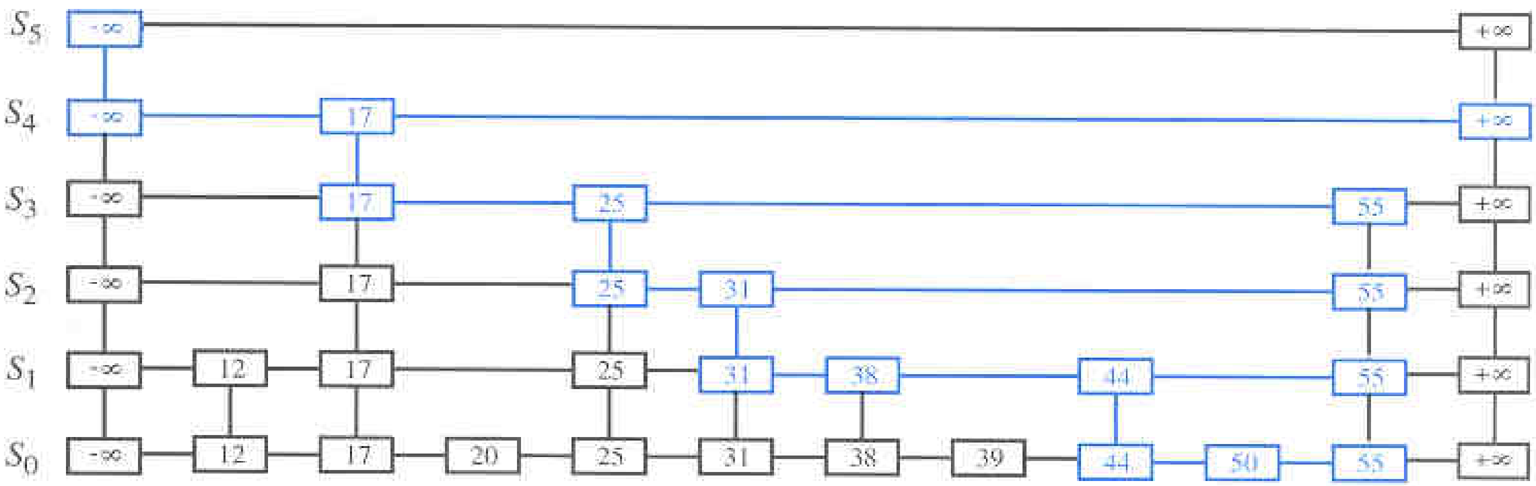
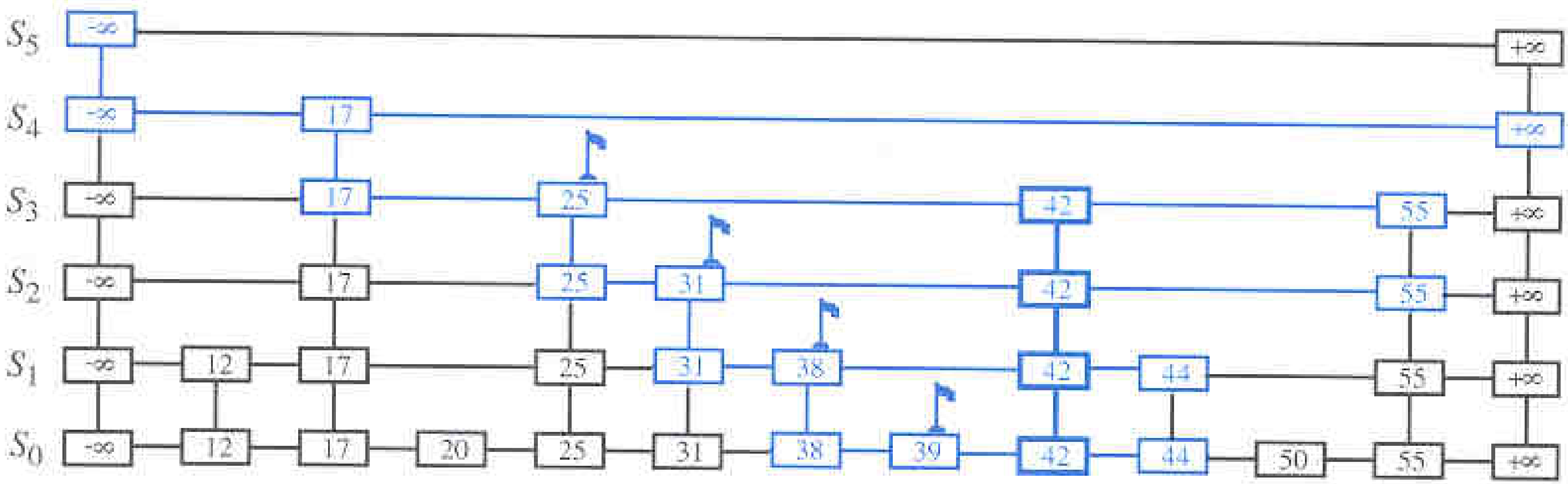
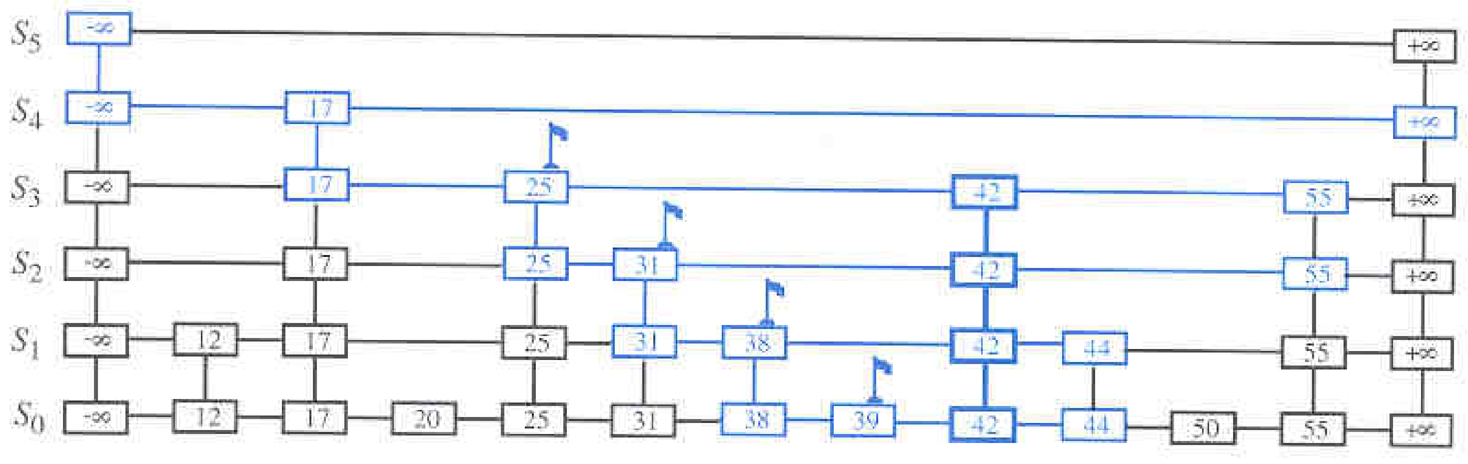


Figure 8.10: Example of a search in a skip list. The positions visited when searching for key 50 are highlighted in blue.

Insertion diagram



Insertion algorithm



Algorithm SkipInsert(k, e):

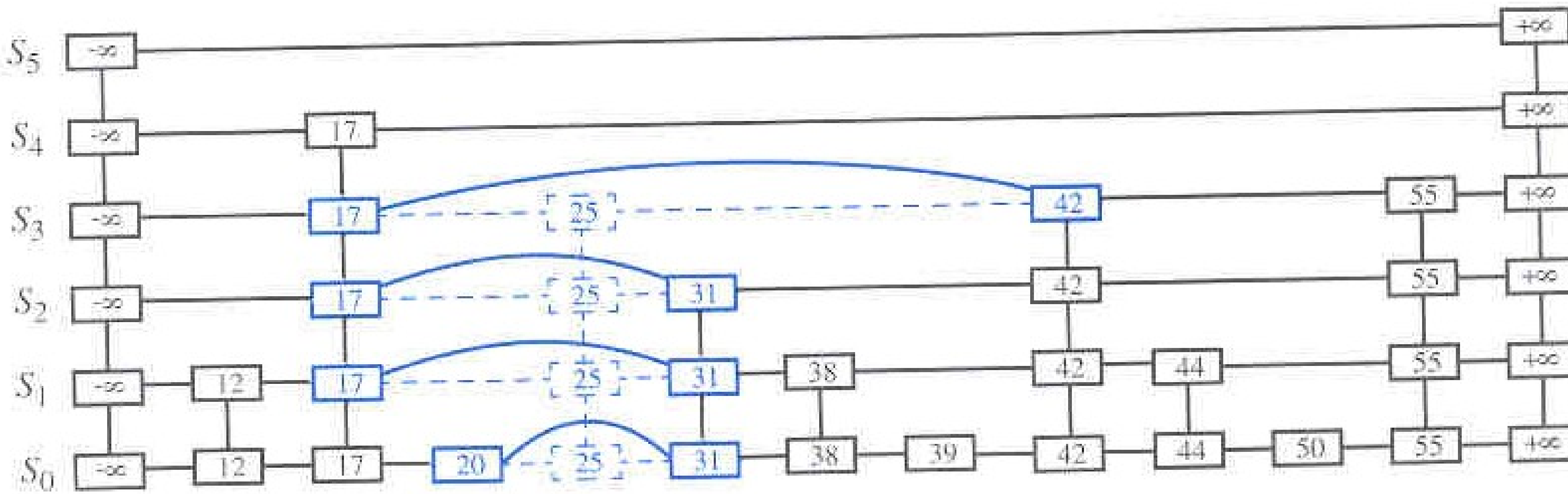
Input: Item (k, e)

Output: None

$p \leftarrow$ SkipSearch(k)
$q \leftarrow$ insertAfterAbove$(p, \textbf{null}, (k, e))$ {we are at the bottom level}
while random() $< 1/2$ **do**
  while above$(p) =$ **null do**
    $p \leftarrow$ before(p) {scan backward}
  $p \leftarrow$ above(p) {jump up to higher level}
  $q \leftarrow$ insertAfterAbove$(p, q, (k, e))$ {insert new item}

Code Fragment 8.5: Insertion in a skip list, assuming random() returns a random number between 0 and 1, and we never insert past the top level.

Remove algorithm



(sort of) Analysis of Skip Lists

- No guarantees that we won't get O(N) behavior.
  - The interaction of the RNG and the order in which things are inserted/deleted *could* lead to a long chain of nodes with the same height.
  - But this is very unlikely.
  - ***Expected*** time for search, insert, and remove are O(log n).