ID1020 2018-09-17 #7

Quicksort (Tony Hoare)
- Shuffle
- Partition so that for some value j
  - The element a[j] is correctly placed
  - No element greater than a[j] is to the left of j
  - No element less than a[j] is to the right of j
- Sort all sub-arrays recursivly

Hoare invented Quicksort to translate from russian to english.

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

Quicksort partitioning:
Repeat until the indices i and j cross
- Scan by i from left to right while (a[i] < a[lo])
- Scan by j from right to left while (a[j] > a[lo])
- Swap a[i] and a[j]

Not stable. Can be made stable with extra array, but then merge sort is probable better anyway.

Can be 30% faster than merge sort.
Harder to implement.

Worst case: $O(N^2)$
Extremly unprobable
Average case: $O(N \log N)$

Randomized divide-and-conquer algorithm.
In-place logarithmic stack space

Improvements:
- Insertion sort on ~10 elements.
- Partitioning element: Median-of-three

Problem: Duplicate keys
Stop scanning when equal keys are found

3-way partitioning (Dijkstra)
Dijkstras method not good with few equal keys
Sedgewick-Bentley

3-way partitioning can make sorting possible in almost linear time.

When to choose different sorting algorithms?

Things to take into account:
- Stable?
- Parallel?
- In-place?
- Determenistic?
- Duplicate keys?
- Different types of keys?
- Linked list or array?
- Large or small elements? (copy/swap)
- Order of the input?
- Performance guarantees?

"System sort" is often fast enough for common cases

Priority Queues

Often: low value → high priority
Enqueue: Insert element
Dequeue: Remove and return element with highest priority

Stability: Preserve FIFO-order among elements with equal priority

Lazyness: Priority queues need only be partially sorted

Linked list implementation
- Keep elements ordered in descending priority order
- Enqueue: Sort the new element into correct position
- Dequeue: Remove the first (highest priority) element.

Binary heap - A balanced binary tree with heap property
- All nodes have higher or equal priority than any of its two children
- The highest priority element is found in the root.

Array heap representation | 1 | 2 | 3 | | 4 | | ...

Bottom-up reheapify (swim)
Usage: Enqueue a new element to the first empty place in the lowest level, restore heap order by ~~bottom-up~~ reheapify.
Swim: If the child has higher prio than its parent - swap them. Continue until no swap or root is reached.
Top-down reheapify (sink)
Usage: Dequeue - remove the top element and replace it with the last (rightmost) element in the bottom level - do a top-down reheapify (sink)
Sink: If the node has lower priority than any child - swap it with highest priority child. Continue until no swap or no children.
Heap: Simple to implement, Theoretical $O(\log N)$ performance for enqueue & dequeue

Heap sort
Bad cache performance - store elements in different order to solve

Special purpose priority queues

Process/thread scheduling in OS
- Fixed number of priorities
- Array of linked lists - one per priority.
- Enqueue and Dequeue is $O(1)$.

Calendar queue
- Array with ~365 elements - one for each day
- One linked list per day. Modulo operations for different years

#days $\frac{N}{2}$, daylength $\sim 2 \Rightarrow O(1)$

Fast general purpose priority queue
- Splay tree, Tarjan & Sleator

Priority queues are also used for many other algorithms.