Median-of-3 Decision Tree

Sorting with Decision Trees

```c
int[] decisionTreeSort(int a, int b, int c) {
    if (a < b) {
        if (b < c) return {a, b, c};
        else if (a < c) return {a, c, b};
        else return {c, a, b};
    }
    else {
        if (c < b) return {b, a, c};
        else if (c < a) return {b, c, a};
        else return {c, b, a};
    }
}
```

In the worst case, how many questions would you need to ask to definitively sort \{a, b, c, d\}? 

- 3
- 4
- 5
- 6
- Not sure

Optimal Comparison Sort


On random arrays, decision tree sorting is optimal in the number of comparisons.

Cost model. Number of comparisons.

Optimal decision tree sort doesn’t exist. Provable optimal for \(N < 16\) and \(N = 22\).

Towards Optimal Sorting of 16 Elements

Marcin Peczarski
Institute of Informatics, University of Warsaw,
ul. Banacha 2, 02-097 Warsaw, Poland,
marcin@mini.uw.edu.pl

Abstract. One of the fundamental problems in the theory of sorting is to find the optimal number of comparisons sufficient to sort a given number of elements. Currently 16 is the largest number of elements for which we do not know the exact value. We know that 46 comparisons suffice and that 44 do not. There is an open question if 45 comparisons are sufficient. We present an attempt to resolve that problem by performing an exhaustive computer search. We also present an algorithm for counting linear extensions which substantially speeds up computations.
Upper Bound for N!

**Goal.** Find an asymptotic complexity bound for the function $\log(N!)$.  

**Subgoal.** Find an upper bound for the function $N!$.

$$N! = 1 \cdot 2 \cdot 3 \cdots (N - 2) \cdot (N - 1) \cdot N$$

Upper Bound for log(N!)

**Goal.** Find an asymptotic complexity bound for the function $\log(N!)$.  

**Subgoal.** Find an upper bound for the function $\log(N!)$.

$$N! < N^N$$

Lower Bound for log(N!)

**Goal.** Find an asymptotic complexity bound for the function $\log(N!)$.  

**Subgoal.** Find a lower bound for the function $\log(N!)$.

$$? \leq N! \leq N^N$$

$$\log ? \leq \log N! \leq N \log N$$

What can we say about decision tree sorting?

- $\log N! \in O(N \log N)$
- $\log N! \in \Omega(N \log N)$

- Both
- Neither
- Not sure
**Algorithm Design Paradigms**

**Greedy Algorithms.** Consider each option in order of lowest-cost.
- Prim's Algorithm.
- Kruskal's Algorithm.
- Dijkstra's Algorithm.

**Caveat.** Can lead to suboptimal solutions.
- Dijkstra's algorithm on negative edge weighted graphs.

**Divide-and-Conquer Algorithms.** Solve two or more subproblems recursively, and then combine the results.
- Merge sort.
- Quicksort.

**Prototypical usage.** Turn brute-force $N^2$ runtime algorithm into $N \log N$ algorithm.

---

**Algorithm Design Process**

**Hypothesize.** How do invariants affect the behavior for each operation?

**Identify.** What strategies have we used before? What examples can we apply?

**Plan.** Propose a new way from findings.

**Analyze.** Does the plan do the job? What are potential problems with the plan?

**Create.** Implement the plan.

**Evaluate.** Check implemented plan.

**Find a lower and upper bound.** Define a slow but totally correct solution. Build a mental model: identify key properties.

**Consider each algorithm that you know.** Which ones might work? How do the existing algorithms break down?

**Apply an algorithm design idea.** Perform a reduction: transform input and output. Or modify the data structures used.

**Use an algorithm design paradigm.**

---

**Counting Inversions**

Given a permutation of length $N$, count the number of inversions.

```
0 2 3 1 4 5 7 6
```

3 inversions: $2-1$, $3-1$, $7-6$

Lower bound? Upper bound? Desired runtime? Algorithm paradigm?

---

**Optical Character Recognition**

Suppose we’re building an optical character recognition system.

We want to separate lines of text. There is some white space between the lines but problems like noise and the tilt of the page makes it hard to find.

How can we do line segmentation?