CS F364
Design & Analysis of Algorithms

Analysis – Problems – Lower Bounds

Analysis of Problems

- Analysis of Comparison-Based Sorting
- Lower Bound for (Comparison-Based) Sorting
    - Performance of Sorting Algorithms
- Decision Tree Model
    - Lower Bound for Searching

3/17/2015 Sundar B. CSIS, BITS, Pilani

SORTING – COMPARATIVE PERFORMANCE OF ALGORTIHMS

Metric Algo.	Insertion Sort	Merge Sort	QuickSort
Worst Case Time	O(N*N)	O(NlogN)	O(N*N) w. low prob.
Average Case Time	O(N*N)	O(NlogN)	O(NlogN) w. high prob.
Performance on small lists	Extremely good	Not good	Not good
Space	O(1)	O(N)	O(logN)
Online/Offline	Online	Partly Online	Offline
Memory access	Seq. Read (find) Random Write (insert)	Seq. Read Seq. Write	Random Read Random Write

Why?

Sorting – Sorting Model

- Is this the best we can do for Sorting?
  - Algorithm Complexity vs. Problem Complexity
- Caveats / Simplification:
  1. *We are considering only comparison based sorting algorithms*:

     This is more of a restriction on the problem rather than on the solution – Why?
  2. *We are counting only the number of comparisons*

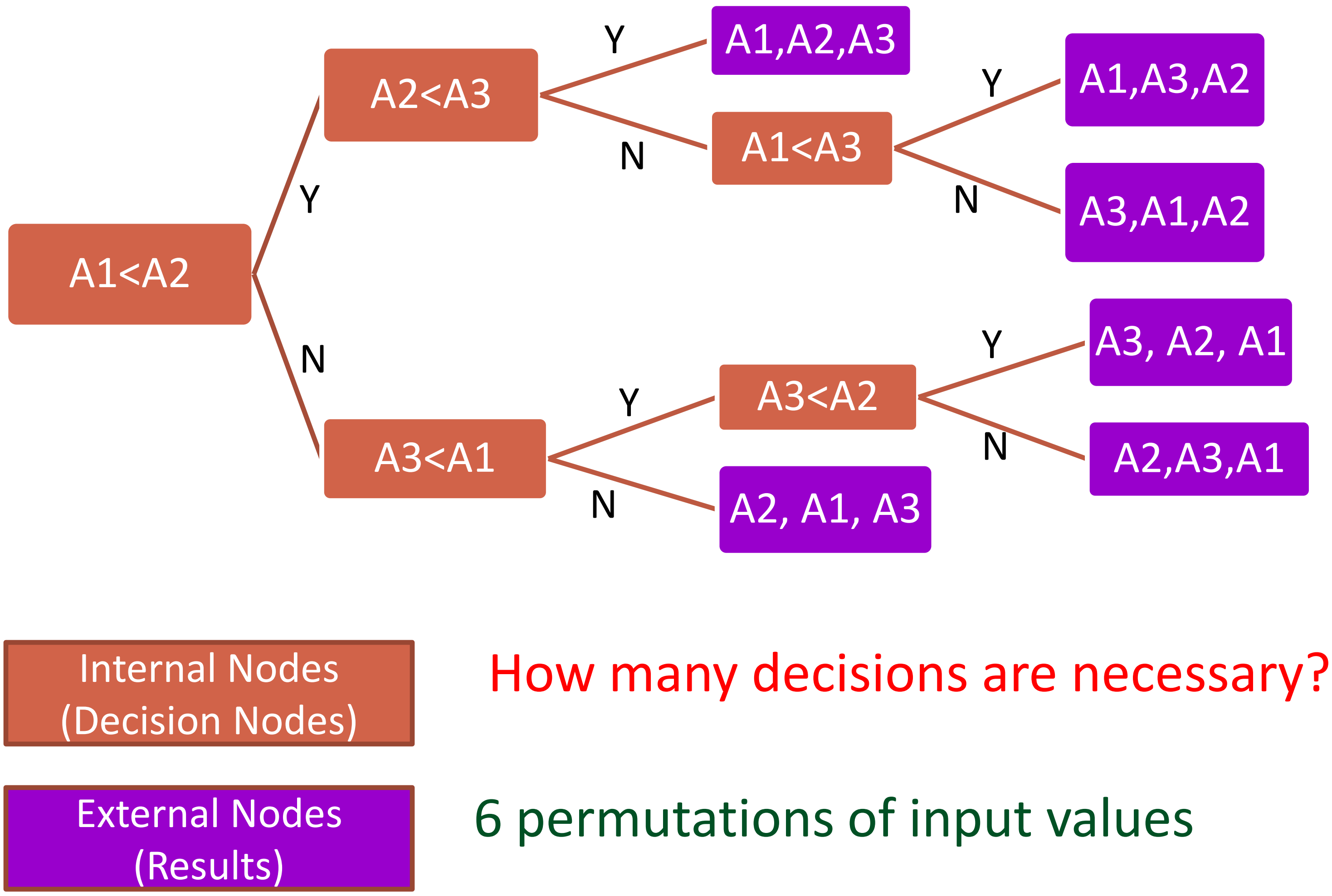
     Are there scenarios where other operations dominate comparison operations?
- (Comparison) Sorting
  - Can be solved in polynomial time – in particular in O(NlogN) time (worst case)
    - Witness: **Merge Sort**

Sorting – Lower Bound

- Is this the best we can do for Sorting?
  - Algorithm Complexity vs. Problem Complexity
- Sorting
  - Is there a lower bound (on ***worst case time complexity***) for sorting?
    - i.e. is there a (lower) limit for the time taken – in the worst case – for sorting a list of N elements using any sorting algorithm (that must compare values)?

SORTING – LOWER BOUND - EXAMPLE

The problem of sorting can be depicted as a a Decision Tree: (given a list of 3 unique values)

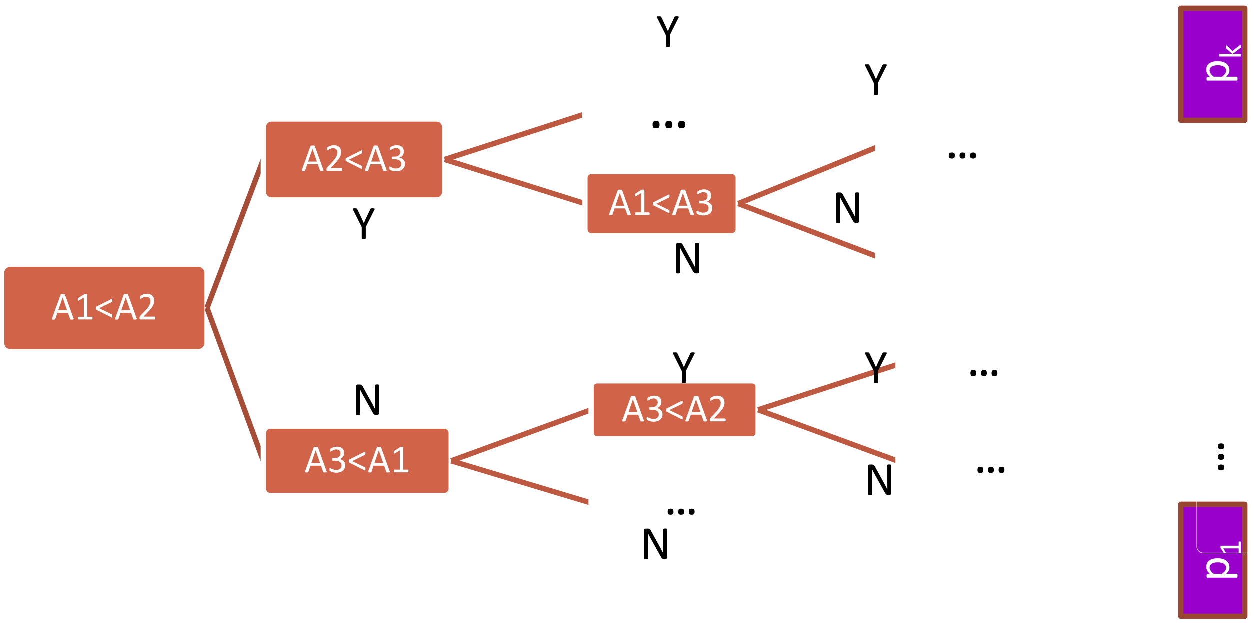


How many decisions are necessary?

6 permutations of input values

SORTING – LOWER BOUND – GENERAL SCENARIO

Decision Tree for Sorting (a list of N unique values)



Internal Nodes (Decision Nodes)

External Nodes (Results)

How many decisions are necessary?

N! permutations of input values

Sorting – Lower Bound

- Minimum number of decisions necessary
  - is the same as the minimum depth of a (binary) tree with N! nodes.
- Depth of a tree is the length of the longest path from root to an external node:
  - the number of nodes can grow geometrically
    - in the best case - at every level there are two branches
    - i.e. # nodes at each level is the sequence: 1, 2, 4, …
- Thus if the total number of nodes in the tree is M the longest path would be at least $\log_2 M$
  - In our case, the depth is $\log_2(N!)$
  - and so the minimum number of decisions is log(N!)

SORTING – LOWER BOUND [2]

- log(N!) can be simplified by Stirling's approximation:

  $$n! = (\sqrt{(2\pi n)})(n/e)^n(1+\Theta(1/n))$$

  i.e. $\log(n!) = (1/2)\log(n) + n\log(n) - n\log(e) + O(1)$
- This is the minimum number of comparisons required for sorting a list of N items
  - i.e. this is the lower bound on the (worst case) number of comparisons for sorting that requires comparison.

Sorting Algorithms

- Question:
  - How close are the actual algorithms to the lower bound?
- Insertion Sorting *(if binary search is used for location)*:
  - Number of comparisons in the worst case is given by
    - $\Sigma_{i=1 \text{ to } n-1} 2*\log_2 i = 2*\log (n-1)!$
  - Exercise: *Plot this against the lower bound.*

Sorting Algorithms

- Question:
  - How close are the actual algorithms to the lower bound?
- Merge Sorting (bottom up implementation):
  - In step j (= 0 to ceil($\log_2 N$))
    - for each i (= 0 to ceil(N/k)-1)
      - lists A[i .. i+k-1] and A[i+k .. i+2*k-1] are merged, where $k=2^j$
  - Each merger of lists of size m each requires 2*m – 1 comparisons:
  - Number of comparisons:
    - $\Sigma_{j=1 \text{ to } \log N}$ (N/k)*(k-1)
  - Exercise: *Simplify the expressions and plot it against the lower bound.*

Searching – Lower Bound

- Exercise:
  - Can this technique be applied for searching?
    - i.e. Can you use a decision tree to carry out analysis of searching algorithms?
      a) assuming that the list is sorted and
      b) not assuming that that list is sorted