EE360C: Algorithms

The Basics

Summer 2019

Department of Electrical and Computer Engineering
University of Texas at Austin

Recap

Asymptotic bounds

Big-Oh

Given $f(n)$, we denote by $\Omega(g(n))$ the set of functions:

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

Big-Ω

Given $f(n)$, we denote by $\Omega(g(n))$ the set of functions:

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$$

Asymptotic bounds

Θ

Given $g(n)$, we denote by $\Theta(g(n))$ the set of functions:

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

- Limit Theorems
- Properties of asymptotics

Common Running Times

Linear Time: $O(n)$

Linear Time

Running time is at most a constant factor times the size of the input.

What are some things you think you can do in linear time?

Compute the maximum of n numbers $a_1, \ldots a_n$

$max \leftarrow a_1$
for $i = 2$ **to** n
  do if $(a_i > max)$
    then $max \leftarrow a_i$

Linear Time: $O(n)$ (cont.)

Merge

Combine two sorted lists $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_m$ into a sorted whole

$i = 1, j = 1$
while (both lists are nonempty)
    do if $(a_i < b_j)$ append a_i to output and increment i
        else append b_j to output and increment j
append remainder of nonempty list to output

Claim

Merging two sorted lists of total size n takes $O(n)$ time.

Analysis

After each comparison, the length of the output list increases by 1.

Linearithmic Time: $O(n \log n)$

Linearithmic Time

Commonly arises in divide and conquer algorithms.

What kinds of problems do you think take $O(n \log n)$ time?
Sorting!

Largest Empty Interval

Given n time stamps $x_1, \ldots, x_n$, on which copies of a file arrive at a server, what is the largest interval of time when no copies of the file arrive?

An $O(n \log n)$ Solution

Sort the time stamps. Scan the sorted list in order, identifying the maximum gap between successive time stamps.

Quadratic Time: $O(n^2)$

What kinds of things do you think take quadratic time?

- Enumerate all pairs of elements. If there are n elements, then there are
$$\binom{n}{2} = \frac{n(n-1)}{2}$$
pairs
- Nested loops.

Quadratic Time: $O(n^2)$ (contd.)

Closest Pair of Points

Given a list of n points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is the closest.

$O(n^2)$ **solution?**

Try all pairs of points

```
min ← (x_1 − x_2)^2 + (y_1 − y_2)^2
for i = 1 to n
     do for j = i + 1 to n
             do d ← (x_i − x_j)^2 + (y_i − y_j)^2
                  if d < min
                     then min ← d
```

Do you think $\Omega(n^2)$ is a lower bound?

Cubic Time: $O(n^3)$

What kinds of things do you think take cubic time?

Set Disjointness

Given n sets $S_1, \ldots S_n$, each of which is a subset of $\{1, 2, \ldots, n\}$, is there some pair of these which are disjoint?

$O(n^3)$ Solution

For each pair of sets, determine if they're disjoint.

for each set S_i
    do for each other set S_j
        do for each element $p \in S_i$
            do determine if $p \in S_j$
        if no element of S_i belongs to S_j
          then report S_i and S_j are disjoint

Polynomial Time: $O(n^k)$

Independent Set of Size k

Given a graph, are there k nodes such that no two are joined by an edge? (Where k is a constant.)

$O(n^k)$ Solution

Enumerate all subsets of k nodes.

```
for each subset S of k nodes
    do check whether S is an independent set
       if S is an independent set
          then report S is an independent set
```

Polynomial Time: $O(n^k)$ (contd.)

- The number of subsets of size k ("n choose k"):

  $$\frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots(2)(1)} \leq \frac{n^k}{k!}$$

  This is $O(n^k)$.
- Check if S is an independent set—for each pair of nodes, check if an edge exists between them. This is $O(k^2)$.
- $O(k^2 n^k / k!) = O(n^k)$

Exponential Time

Some things are just plain expensive. And they're not always obviously expensive. (More later.)

Independent Set

Given a graph, what is the size of the largest independent set?

$O(n^2 2^n)$ Solution

Enumerate all subsets.

```
S* ← ∅
for each subset S of nodes
    do check whether S is an independent set
        if S is largest independent set seen so far
            then update S* ← S
```

Sublinear Time

Some things, when phrased properly, are just plain ~~easy~~ fast.

Can you think of anything you can do faster than $O(n)$?

Binary Search

Given a sorted array A of size n, determine whether p is in the array.
Start with BINARYSEARCH$(A, 1, n, p)$.

```
BinarySearch(A, i, j, p)
  m ← ⌊(j − i)/2⌋
  if A[m] = p
      then return true
  else if p < A[m]
      then return BinarySearch(A, i, m − 1, p)
  else return BinarySearch(A, m + 1, j, p)
```

Binary search's running time is $O(\log n)$.

Lower Bounds for Sorting

Comparison Sorting

The sorted order of the output is based on a comparison between input elements. Such sorting algorithms are called comparison sorting.

Comparisons could be of the form $<$, $\leq$, $=$, $>$, $\geq$.

Lower Bounds for Sorting Algorithms

- All the inputs are distinct.
- Without loss of generality we consider $\leq$ comparisons.

Decision Trees Abstraction

Decision Trees

We abstract comparison sorting in terms of *decision trees*.

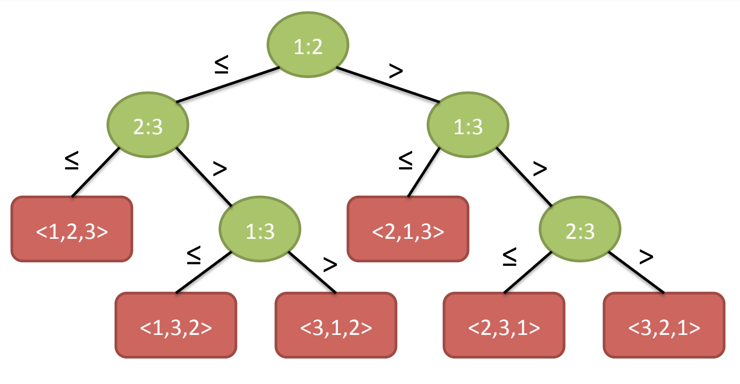
- a full binary tree
- represents the comparisons between elements performed by a sorting algorithm
- each internal node is annotated by $i : j$ for some i and j in $1 \leq i, j \leq n$, for n elements in the input sequence
- each leaf is annotated by a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$
- executing the sorting algorithm equates to tracing a path through the decision tree

Decision Trees Abstraction (contd.)

Decision Trees (contd.)

- Each internal node indicates the comparison $a_i \leq a_j$.
- The left subtree captures subsequent comparisons, once we know that $a_i \leq a_j$.
- The right subtree captures subsequent comparisons, once we know that $a_i > a_j$.
- When we come to a leaf the sorting algorithm has established the desired ordering.

Decision Tree Example



Any correct sorting algorithm must be able to generate each of the n! peruations on n elements; each permutation must appear as a leaf in the decision tree.

Lower Bound for the Worst-Case

Worst Case

The length of the longest path from the root of a decision tree to any of its reachable leaves is the worst-case number of comparisons that the corresponding sorting algorithm performs.

Said another way... the worst case number of comparisons for a comparison sort algorithm is the height of its decision tree.

Lower Bound

A lower bound on the heights of all decision trees in which each permutation appears as a reachable leaf is therefore a lower bound on the running time of any comparison sort algorithm.

Lower Bound for the Worst-Case (cont.)

Theorem

Any comparison sort algorithm requires $\Omega(n \lg n)$ comparisons in the worst case. (We have to make at least $\Omega(n \lg n)$ comparisons for any comparison sorting algorithm.)

Proof

We have to determine the height of a decision tree with $n!$ leaves. A binary tree of height h has no more than 2^h leaves. Therefore $n! \leq 2^h$, so $h \geq \lg(n!) = \Omega(n \lg n)$.

Corollary

Heapsort and Mergesort are asymptotically optimal comparison sorts.

Questions?