CS3230 Semester 2 2024/2025

Design and Analysis of Algorithms

Tutorial 05
D&C, Sorting and Average-case analysis
For Week 06

Document is last modified on: February 4, 2025

1 Lecture Review: Decision Tree

A decision tree contains:

- Vertices (Internal): A comparison
- Branches: Outcome of comparison
- Leaves: Output / decision for the input

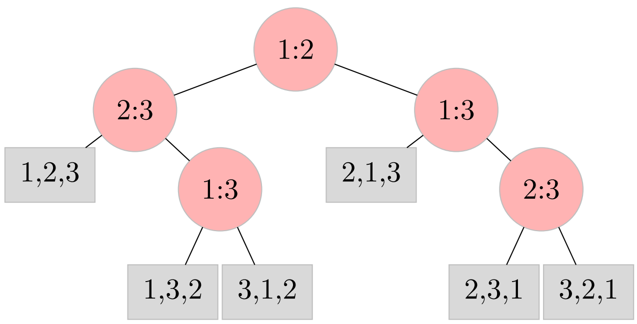

Figure 1: Worst case runtime is the height of the decision tree.

The classic example to illustrate the usage of decision tree is for showing the lower bound of comparison-based sorting is $\Omega(n \log n)$. Here is a picture of decision tree of sorting $n = 3$ elements and there are $3! = 6$ possible outputs (decision for the inputs) which must all been catered for. As each comparison of two comparable elements a versus b yields two possible outcomes: $a < b$ (which means a must be in front of b) or $a \geq b$ (here b must be in front of a if $a > b$; on the other hand, if $a = b$, then it does not matter which order a and b are put in). This decision tree is thus a binary tree. The height of binary decision tree so that its number of leaves is at least $n!$ is $\log n! \approx n \log n$ (use Stirling's formula).

2 Tutorial 05 Questions

Q1, Q2, Q3, and Q4 involve Polynomial Multiplication of two polynomials of degree n.
Let $A(x) = a_n \cdot x^n + \ldots + a_2 \cdot x^2 + a_1 \cdot x + a_0$.
Let $B(x) = b_n \cdot x^n + \ldots + b_2 \cdot x^2 + b_1 \cdot x + b_0$.
Let $C(x) = A(x) \times B(x) = c_{2n} \cdot x^{2n} + \ldots + c_2 \cdot x^2 + c_1 \cdot x + c_0$
Assume all coefficients a_i, b_i, c_i are Integers.
Assume that all addition and multiplication operations of two Integers take $O(1)$ time.

We can compute the coefficients c_i of $C(x)$ in $O(n^2)$ using complete search:
For each $i \in [2n..0]$, $c_i = \sum a_j \cdot b_{i-j}$ where both j and $i - j$ are between 0 and n (inclusive).

Q1). Let $x = 10$ to make it easier to visualize this as a normal base 10 multiplication and $n = 2$.
Let $A(10) = 352 = 3 \cdot 10^2 + 5 \cdot 10 + 2$, i.e., $a_2 = 3, a_1 = 5, a_0 = 2$.
Let $B(10) = 221 = 2 \cdot 10^2 + 2 \cdot 10 + 1$, i.e., $b_2 = 2, b_1 = 2, b_0 = 1$.
Compute the coefficients of $C(10) = A(10) \times B(10) = 77\,792$ using the $O(n^2)$ algorithm above.

Q2). Suppose that you are given the following Divide and Conquer (D&C) algorithm:
Rewrite $A(x) = x^{\frac{n}{2}} \cdot A_1(x) + A_2(x)$
Rewrite $B(x) = x^{\frac{n}{2}} \cdot B_1(x) + B_2(x)$
where $A_1(x), A_2(x), B_1(x), B_2(x)$ are all now polynomials of degree (up to) $\frac{n}{2}$.

We now compute four smaller polynomial multiplications:
$A_1(x) \times B_1(x), \quad A_1(x) \times B_2(x), \quad A_2(x) \times B_1(x), \quad A_2(x) \times B_2(x)$

And we compute:
$C(x) = x^n \cdot [A_1(x) \times B_1(x)] + x^{\frac{n}{2}} \cdot [A_1(x) \times B_2(x) + A_2(x) \times B_1(x)] + A_2(x) \times B_2(x)$

Apply this D&C algorithm to compute the multiplication of the same two polynomials of degree $n = 2$:
Rewrite $A(10) = 352 = 10 \cdot (3 \cdot 10 + 5) + 2$
Rewrite $B(10) = 221 = 10 \cdot (2 \cdot 10 + 2) + 1$

Compute $A_1(10) \times B_1(10), \quad A_1(10) \times B_2(10), \quad A_2(10) \times B_1(10), \quad A_2(10) \times B_2(10)$.
Then, compute $C(10)$.

Q3). What is the time complexity of that recursive D&C algorithm?

Q4). Introducing: the Karatsuba's algorithm.
We still compute two smaller polynomials: $A_1(x) \times B_1(x),\ A_2(x) \times B_2(x)$.
But instead of computing: $A_1(x) \times B_2(x), A_2(x) \times B_1(x)$ that requires two polynomial multiplications,
we compute: $[A_1(x)+A_2(x)] \times [B_1(x)+B_2(x)]$ that requires two additions and just one multiplication.
Note: $A_1(x) \times B_2(x) + A_2(x) \times B_1(x) = [A_1(x)+A_2(x)] \times [B_1(x)+B_2(x)] - A_1(x) \times B_1(x) - A_2(x) \times B_2(x)$.
We now have the elements needed to compute $C(x)$ in faster time.

What is the time complexity of Karatsuba's algorithm?

Q5). Decision Tree

You are given 243 balls, all but one of which have the same weight; the remaining one is heavier. Your job is to find which of the balls is heavier. Your friend has a balance scale, but will charge you for each weighing. You want to minimize the (worst-case) number of weighings needed. What is the minimum number of weighings needed to find the ball?

You can assume that we only use comparison model (comparison returns $<$, $=$, or $>$). You can decide how to compare the ball(s). What is the lower bound of <u>any</u> algorithm to solve this problem?

Q6) You are given an array $A[1..n]$ that is sorted in **non-increasing order**. Your task is to find the largest index i such that $A[i] \geq i$. Design an efficient algorithm to solve this problem.
To guide your approach, consider the following properties of the sorted array:

- If $A[j] \geq j$, then it must hold that $A[j-1] \geq j-1$, unless $j = 0$.
- If $A[j] < j$, then it must follow that $A[j+1] < j+1$, unless $j = n$.

For ease of notation, assume that the array is extended such that $A[0] > 0 \quad \text{and} \quad A[n+1] < n+1$. Thus, there is a unique i such that $A[i] \geq i$ but $A[i+1] < i+1$.
Q7) Bogosort is an extremely inefficient sorting algorithm. It repeatedly generates random permutations of the input array until it encounters one that is sorted by chance. What is the best-case, worst-case and average-case time complexity of Bogosort for an array of length n?

Algorithm 1: Bogosort($A[0..n-1]$)

```
while not IsSorted(A) do
    RandomlyShuffle(A)
return A
Function IsSorted(A):
    for i ← 1 to n − 1 do
        if A[i] < A[i − 1] then
            return false
    return true
```

Note: The RandomlyShuffle function can be implemented in $O(n)$ time using the Fisher-Yates shuffle algorithm.