Data Structures

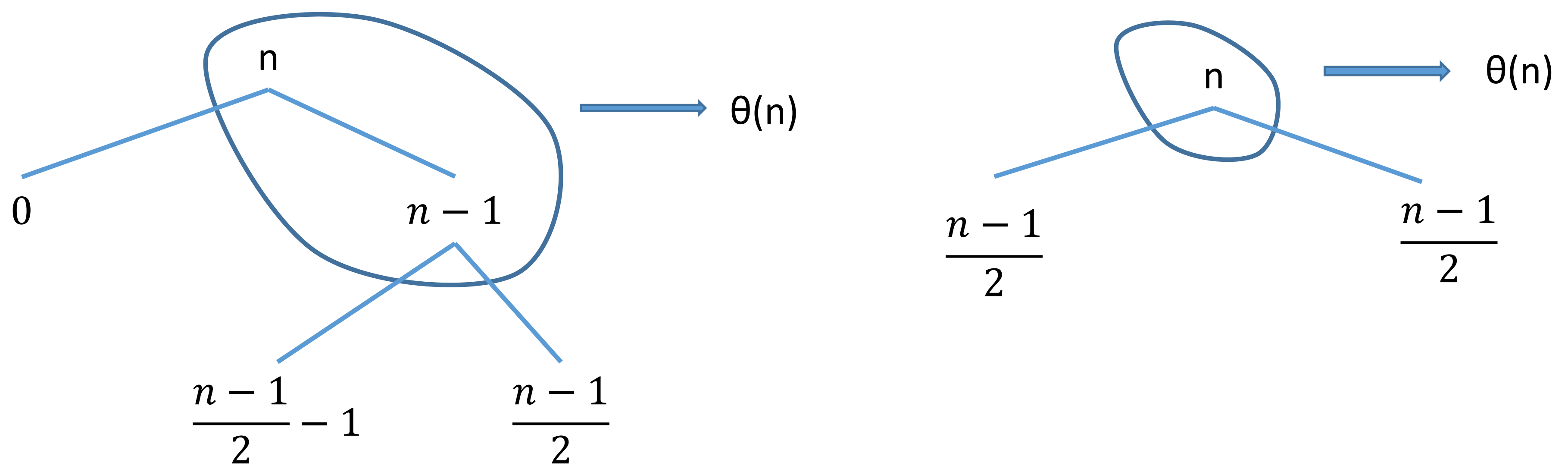
Analysis of Quick Sort: average case

- The behaviour of quick sort depends upon the relative ordering of the elements in the input array
- All permutations of input numbers are equally likely (assumption)
- On a random input array, some of the splits will be reasonably balanced and some are fairly unbalanced
- In the average case, Partition procedure produces a mix of good (best-case splits) and bad splits (worst-case splits)
- In a recursion tree for an average-case execution of Partition, the good and bad splits are distributed randomly throughout the tree

Analysis of Quick Sort: average case

- Suppose that the good and bad splits alternate levels in the tree
- A bad split happens at the root which produces two sub arrays of sizes "0" and "n-1"
- At the next level a good split happens which produces two sub arrays of sizes "(n-1)/2-1" and "(n-1)/2"
- The combination of a bad split followed by a good split produces three sub arrays of sizes 0, (n-1)/2-1 and (n-1)/2
- The partitioning cost of these splits is: $\theta(n) + \theta(n-1) = \theta(n)$

Analysis of Quick Sort



Randomized quick sort

- In a practical situation all permutations are not equally likely
- Can add randomization to an algorithm to obtain a good expected performance over all inputs
- The resulting algorithm is called randomized quick sort
- A randomization technique called “random sampling” is used
- Use a randomly chosen element from given subarray as the pivot instead of the rightmost element
- The input array is expected to get split into reasonably balanced sets on average

Randomized quick sort

```
Randomized_Partition(A, p, r)
        i ← RANDOM(p, r)
        exchange A[r] with A[i]
        return Partition(A, p, r)
Randomized_QuickSort(A, p, r)
        if(p < r)
                q ← Randomized_Partititon(A, p, r)
                Randomized_QuickSort(A, p, q-1)
                Randomized_QuickSort(A, q+1, r)
```

Randomized quick sort: Worst-case

T(n) = $\max_{0 \le q \le n-1} (T(q) + T(n-q-1))$ + θ(n)

Use guess-and-test method: T(n) ≤ cn^2 for some constant c (hypothesis)

T(n) ≤ $\max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2)$ + θ(n)

= c $\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2)$ + θ(n)

$(q^2 + (n-q-1)^2)$ achieves the maximum at both the end points of the range

$\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) \le (n-1)^2 = n^2 - 2n + 1$

T(n) ≤ $cn^2 - c(2n - 1)$ + θ(n)

≤ cn^2

Randomized quick sort: Worst-case

T(n) = $\max_{0 \le q \le n-1} (T(q) + T(n - q - 1))$ + θ(n)

Use guess-and-test method: T(n) ≥ cn^2 for some constant c (hypothesis)

T(n) ≥ $\max_{0 \le q \le n-1} (cq^2 + c(n - q - 1)^2)$ + θ(n)

= c $\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2)$ + θ(n)

$(q^2 + (n-q-1)^2)$ achieves the maximum at both the end points of the range

$\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) = (n-1)^2 = n^2 - 2n + 1$

T(n) ≥ $cn^2 - c(2n - 1)$ + θ(n)

≥ cn^2, where c is chosen so that θ(n) dominates c(2n – 1)

A few basics

- Consider a sample space S and an event A
- Indicator random variable associated with event A is denoted as I{A} and defined as:
  - $I\{A\} = \begin{cases} 1 & \text{if A occurs} \\ 0 & \text{if A does not occur} \end{cases}$

A few basics

- The expected value or the expectation of a discrete random variable is:

  E[X] = $\sum_x x . \Pr\{X = x\}$

- I{A} = $\begin{cases} 1 & \text{if A occurs} \\ 0 & \text{if A does not occur} \end{cases}$

- Linearity of expectation: The expectation of sum of two random variables is the sum of their expectations

  E[X + Y] = E[X] + E[Y]

A few basics

Experiment: flipping a fair coin

$S = \{H, T\}$; $\Pr\{H\} = ½$ and $\Pr\{T\} = ½$

H: the coin coming up with heads

X_H : indicator random variable associated with H

$X_H = I\{H\}$

$$I\{H\} = \begin{cases} 1 & \text{if H occurs} \\ 0 & \text{if T occurs} \end{cases}$$

A few basics

The expected number of heads in one flip of coin is the expected value of our indicator X_H:

$E[X_H] = E[I\{H\}]$

$= 1.Pr\{H\} + 0. Pr\{T\}$

$= 1.1/2 + 0.1/2$

$=1/2$

Lemma: Given a sample space S and an event A in S, let $X_A = I\{A\}$. Then

$E[X_A] = Pr\{A\}$

Quick sort: Analysis

Algorithm Partition(A, p, r)

```
    x ← A[r]
    i ← p-1
    for j ← p to r-1
        if(A[j] ≤ x)
            i ← i +1
            exchange A[i] and A[j]
    exchange A[i+1] and A[r]
    return i+1
```

Randomized quick sort: Expected running time

- Randomized quick sort works similar to quick sort except for pivot selection
- Analyse Randomized Quick_Sort by discussing Quick_Sort and Partition algorithms (randomly selected pivot)
- Consider an array of n distinct elements
- The running time of Quick_Sort is dominated by the time spent in Partition procedure
- How many times an element is selected as pivot?
- Observation 1: An element selected as pivot never included in the future recursive calls to Quick_Sort and Partition
- There can be at most n calls to Partition

Randomized quick sort: Expected running time

- One call to Partition:
  - constant amount of time and
  - Amount of time proportional to number of iterations of the **for** loop
- In each iteration of **for** loop, the pivot is compared with an element in the array (pivot-array element comparison)
- What is the total time spent in the **for** loop over all calls to Partition procedure?
- By counting the number of times the pivot is compared to an array element, we can bound the total time spent in the **for** loop

Randomized quick sort: Expected running time

Lemma: Let X be the number of pivot-array element comparisons performed over the entire execution of Quick_Sort on an n-element array. Then the running time of Quick_Sort is O(n + X)

Proof: The algorithm makes at most n calls to Partition procedure

Each call does a constant amount of work and executes the **for** loop some number of times

Each iteration of **for** loop performs one pivot-array element comparison

We have to compute “X”, the total number of comparisons performed over all calls to Partition

Randomized quick sort: Expected running time

- Rename the elements in array A as $z_1, z_2, \ldots, z_n$, where z_i is the i^{th} smallest element in A
- $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ be the set of elements between z_i and z_j inclusive
- Observation 2: each pair of elements are compared at most once, why?
- Define indicator random variable as:

  $X_{ij} = I\{z_i \text{ is compared to } z_j\}$
  (during entire execution of the algorithm)
- Since each pair of elements is compared at most once,

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

Randomized quick sort: Expected running time

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

- We have to compute the quantity, $\Pr\{z_i \text{ is compared to } z_j\}$
- Consider an input array A = {41, 11, 21, 51, 81, 61, 31, 71, 101, 91}
- Assume that the first call to Partition separates this array into two sets: {41, 11, 21, 51, 31} and {81, 71, 101, 91}
- An element from either of these sets will ever be compared with the elements in the other set?