Probabilistic Analysis & Randomized Algorithm-3

Department of Computer Science, Tsinghua University

Randomized Algorithm

- We call an algorithm ***Randomized*** if its behavior is determined not only by its input but also by values produced by a ***random-number generator***.
- Two categories:
  - ***Las Vegas:*** for any input, the algorithm always produces the same correct output, the running time of the algorithm depends on the output of a random-number generator.
  - ***Monte Carlo:*** for any input, the output can be different, depending on the output of a random-number generator.
- Randomized algorithms are often more simple and have better asymptotic bounds, ***with only a small probability of being slow or wrong***.

Randomized HIRE-ASSISTAN

```
RANDOMIZED-HIRE-ASSISTANT(n)
randomly permute the list of candidates
best = 0  ▷ candidate 0 is a least
   qualified dummy candidate
for i = 1 to n
    interview candidate i
    if i is better than best
        best = i
        hire candidate i
```

The expected hiring cost of the procedure: $O(c_h \ln n)$

Randomized HIRE-ASSISTAN

- Compare HIRE-ASSISTANT & RANDOMIZED-HIRE-ASISTANT.
  - *Change:* the expectation is for any input, rather than for inputs drawn from a particular distribution.
  - *Not change:* the expected cost is the same.

- Note1: HIRE-ASSISTANT is deterministic. For any input, the number of hires is fixed. The expectation is over different inputs.
  - A_1 =<1,2,3,4,5,6,7,8,9,10>; 10 hires
  - A_2 =<10,9,8,7,6,5,4,3,2,1>; 1 hire
  - A_3 =<5,2,1,8,4,7,10,9,3,6>; 3 hires

- Note2: RANDOMIZED-HIRE-ASISTANT is not. For input A_3, the number of hires depends on the outcome of its permutation, could be 10, could be 1. The expectation is over different executions.

Design Points

- ***Las Vegas:*** for any input, the algorithm always produces the same correct output, the running time of the algorithm depends on the output of a random-number generator.

- *Design randomized algorithm*: by introducing randomization operations.
  - unknown input distribution
  - avoid worst cases

- *Analyze randomized algorithm*: by using probabilistic analysis.

SELECT

SELECT (A, p, r, i)

if $p == r$

  return $A[p]$

q = PARTITION (A, p, r)

$k = q - p + 1$

if $i == q$

  return $A[p]$

elseif $i < k$

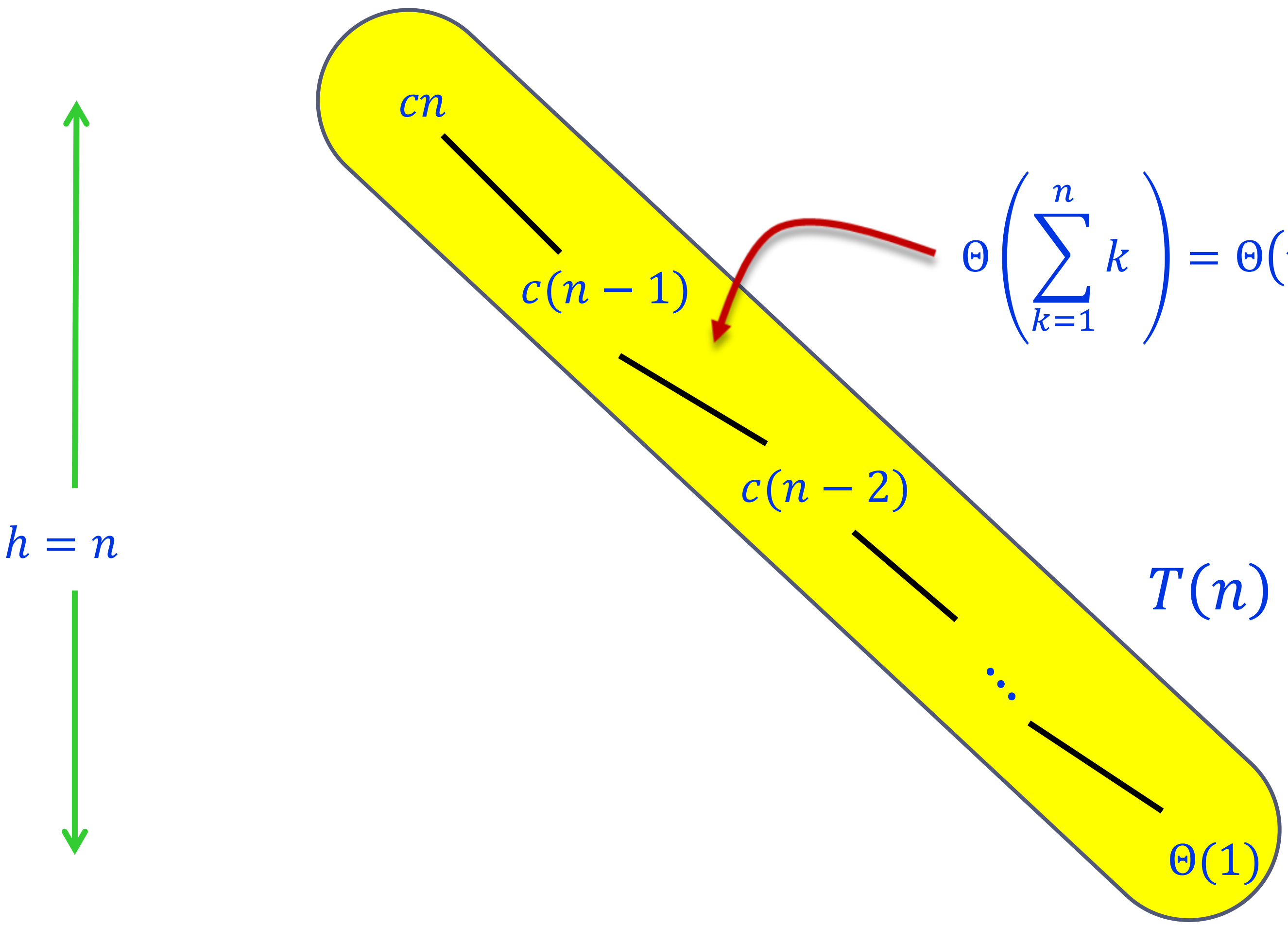
  return SELECT $(A, p, q - 1, i)$

else return SELECT $(A, q + 1, r, i - k)$

Initial call: SELECT$(A, 1, n, i)$

SELECT
Worst-case

$$T(n) = T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$

$$T(n) = \Theta(n^2)$$

Randomized Partition

- IDEA: Partition around a random element.
  - Running time is independent of the input order.
  - No assumptions need to be made about the input distribution.
  - No specific input elicits the worst-case behavior.

```
RANDOMIZED-PARTITION(A, p, r) ▻ A[p..r]
    i = RANDOM(p, r)
    exchange(A[p], A[i])
    return PARTITION(A, p, r)
```

Randomized SELECT

```
RANDOMIZED-SELECT(A, p, r, i)
  if p == r
      return A[p]
  q = RANDOMIZED-PARTITION(A, p, r)
  k = q - p + 1
  if i == k
      return A[q]
  elseif i < k
      return SELECT(A, p, q - 1, i)
  else return SELECT(A, q + 1, r, i - k)
```

Initial call: SELECT($A, 1, n, i$)

Analysis (1)

- $T(n) = \begin{cases} T(\max(0, n-1)) + O(n) & \text{if } 0: n-1 \text{ split} \\ T(\max(1, n-2)) + O(n) & \text{if } 1: n-2 \text{ split} \\ \dots & \\ T(\max(0, n-1)) + O(n) & \text{if } n-1: 0 \text{ split} \end{cases}$

Analysis (2)

For $k = 1, \cdots, n$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k-1 : n-k \text{ split} \\ 0 & \text{otherwise} \end{cases}.$$

$E[X_k] = 1/n$, since all splits are equally likely, assuming elements are distinct.

$$T(n) = \sum_{k=1}^{n} X_k(T(\max(k-1, n-k)) + O(n))$$

Analysis (3)

- $E[T(n)] = E[\sum_{k=1}^{n} X_k (T(\max(k-1, n-k)) + O(n))]$
- $= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n)$
- $= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n)$
- $= \sum_{k=1}^{n} \frac{1}{n} E[T(\max(k-1, n-k))] + O(n)$
- $\leq \frac{2}{n} \sum_{k=\left\lfloor \frac{n}{2} \right\rfloor}^{n-1} E[T(k)] + O(n)$

Recurrence

- $E[T(n)] \leq \frac{2}{n}\sum_{k=\lfloor\frac{n}{2}\rfloor}^{n-1} E[T(k)] + O(n) = O(n)$
- Prove by induction: $E[T(n)] \leq cn$ for a constant $c > 0$.
- Choose a constant a to bound the $O(n)$ term.
- $E[T(n)] \leq \frac{2}{n}\sum_{k=\lfloor\frac{n}{2}\rfloor}^{n-1} ck + an$
- $= \frac{2c}{n}(\sum_{k=1}^{n-1} k \ (1) - \sum_{k=1}^{\lfloor\frac{n}{2}\rfloor-1} k \ \ (2)) + an$
- $\leq \frac{3cn}{4} + \frac{c}{2} + an = cn - (\frac{cn}{4} - \frac{c}{2} - an\ )$
- (when $n \geq \frac{2c}{c-4a}$ and $c > 4a$, $E[T(n)] \leq cn$)

Summary

- Randomized Algorithms
  - ***Las Vegas:*** for any input, the algorithm always produces the same correct output, the running time of the algorithm depends on the output of a random-number generator.
  - ***Monte Carlo:*** for any input, the output can be different, depending on the output of a random-number generator.
  - ***Monte-Carlo Dropout:*** DNN with Monte-Carlo dropout (approximating Bayesian DNN) to generate different predictions.