Algorithms and data structures

Speaker:

Jyrki Katajainen

Today:

- §§7 and 9

Textbook:

Cormen, Leiserson, Rivest, Stein, *Introduction to Algorithms*, 3rd edition, The MIT Press (2009)

Probability theory needed

Sample space: S: all possible outcomes of an experiment

Event: R: a subset of S

Probability: For $R \subseteq S$, $\mathbf{Pr}\{R\} = \sum_{s \in R} \mathbf{Pr}\{s\}$

Uniform probability distribution: For $s \in S$, $\mathbf{Pr}\{s\} = 1/|S|$

Random variable: X: $S \to \mathbb{R}$;
$X = x$ is the event $\{s \in S : X(s) = x\}$;
thus $\mathbf{Pr}\{X = x\} = \sum_{s \in S : X(s) = x} \mathbf{Pr}\{s\}$

Expected value: $\mathbf{E}[X] = \sum_{x} x \cdot \mathbf{Pr}\{X = x\}$

Linearity of expectation: $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$

Example: Flip a fair coin.

$S = \{\mathsf{H}, \mathsf{T}\}$

$\mathbf{Pr}\{\mathsf{H}\} = 1/2$

Y: # heads when flipping a coin n times

$\mathbf{E}[Y] = n/2$

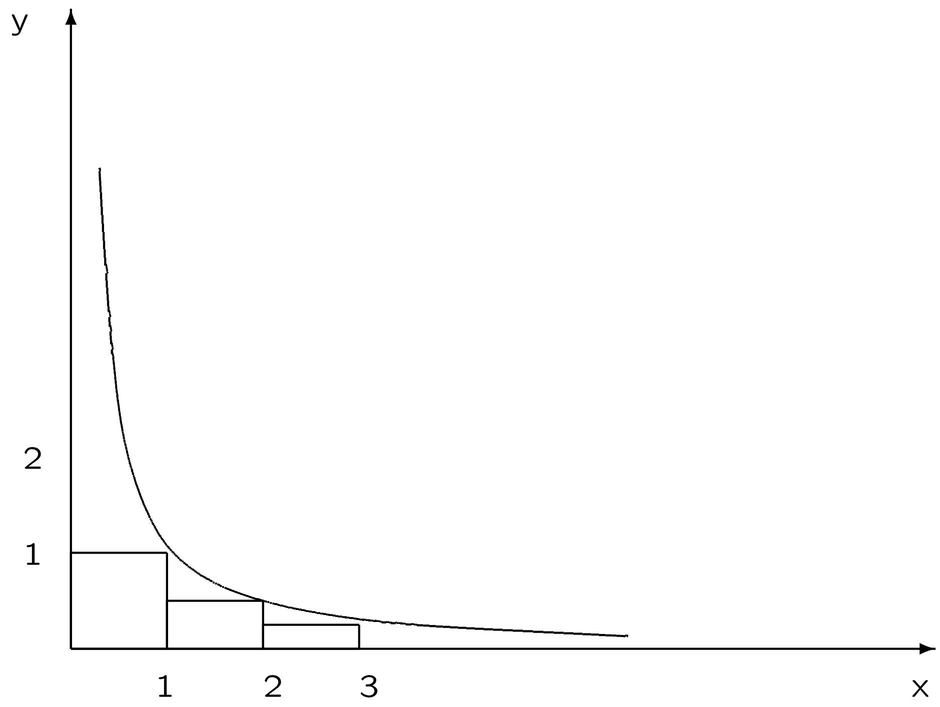
Read §5 and Appendix C

Online exercise (1 min)

What is the probability that James Bond gets 7 when throwing a pair of ordinary dice?

Harmonic series

$$
\begin{aligned}
H_n &= 1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\ldots+\frac{1}{n} = \sum_{k=1}^{n}\frac{1}{k} \\
&\leq 1+\int_1^n \frac{1}{x}\,dx = 1 + \Big/_{1}^{n} \ln x = 1 + \ln n - \ln 1 = 1 + \ln n
\end{aligned}
$$



Randomization

Randomized algorithms make random choices in the course of their execution.

One may want to prove that a randomized algorithm

- has a good expected running time (which is a random variable);
- works well on **any** input with high probability (not considered in this course).

This is **different** from probabilistic analysis of algorithms, where input to an algorithm is chosen from a probability distribution.

Advantages of randomized algorithms: simplicity, efficiency.

RAM with random choices

Extend the normal RAM instructions to include

$$r = \text{Random}(\ell,\ h)$$

which returns an integer between ℓ and h, inclusive, with each such integer being equally likely.

Classification of randomized algorithms

Monte Carlo algorithm can fail to produce a correct answer. However, it should be possible to reduce the failure probability by increasing the running time.

Las Vegas algorithm gives a correct answer; that is, it always produces the correct result or informs about the failure. However, the failure probability must reduce when the running time increases.

Sherwood algorithm produces a correct answer and terminates always. The randomization is only used to improve the running time.

Algorithm design paradigms

Randomized

- Random re-ordering §5
- Random sampling §§7 and 9
- Universal hashing §11
- Abundance of witnesses §31
- Fingerprinting §32
- Randomized rounding §35

Quicksort

Quicksort is a simple divide-and-conquer sorting algorithm that performs well in practice.

Initial call: Quicksort(A, 1, $A.length$)

```
Quicksort(A, p, r)
if p < r
    p′ = Pivot(A, p, r)
    q = Partition(A, p, p′, r)
    Quicksort(A, p, q−1)
    Quicksort(A, q+1, r)
```

$\leq x$	x	$\geq x$

Pivot chooses some element in $A[p \mathinner{.\,.} r]$ as a partitioning element x.

Partition rearranges $A[p \mathinner{.\,.} r]$ so that every element in $A[p \mathinner{.\,.} q-1]$ is smaller than or equal to x, $A[q] = x$, and every element in $A[q+1 \mathinner{.\,.} r]$ is larger than or equal to x.

History

- Original quicksort [Hoare, 1961]
- Tuned quicksort [Sedgewick, 1978]
- Stable quicksort [Motzkin, 1981]
- In-place quicksort, e.g. [Ďurian, 1986]
- Stable in-place quicksort [Katajainen & Pasanen, 1992]
- C library quicksort [Bentley & McIlroy, 1993]
- STL introspective sort [Musser, 1997]
- Samplesort [Chen, 2006]

Choosing a partitioning element

Randomized method

Pivot(A, p, q)
1 **return** Random(p, q)

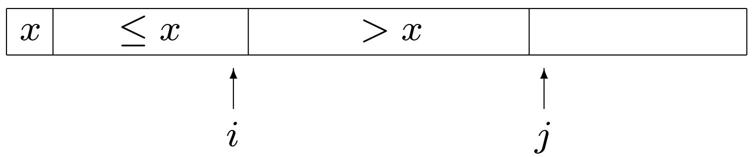
Deterministic methods

- first
- last
- middle
- median of 3 (first, middle, last)
- pseudo-median of 9

Lomuto's partitioning

```
Partition(A, p, q, r)
    x = A[q]
    exchange A[p] with A[q]
    i = p
    for j = p+1 to r
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[p] with A[i+1]
    return i+1
```

Loop invariant:



Running time: $\Theta(n)$, where $n = r - p + 1$.

Worst-case analysis of Quicksort

If $A[p\,..\,r]$ is already sorted and the first element $A[p]$ is chosen as the pivot, Partition splits $A[p\,..\,r]$ into $A[p\,..\,p-1]$ and $A[p+1\,..\,r]$ without changing the order of elements. If this happens at each recursion level, the running time $T(n)$ satisfies:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(0) + T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$$

This implies that

$$T(n) = \Theta\left(\sum_{i=1}^{n} i\right) = \Theta(n^2).$$

In the case of a random pivot, probability for this is close to 0!

Best-case analysis of Quicksort

If the median is chosen as the pivot, Partition splits the array of size n into two subarrays of size $\lceil n/2 \rceil - 1$ and $\lfloor n/2 \rfloor$, respectively. If this happens at each recursion level, the running time $T(n)$ satisfies:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 \end{cases}$$

This implies that

$$T(n) = \Theta(n \lg n).$$

Average-case analysis of Quicksort

Suppose that all the input elements are distinct. Let $n = r - p + 1$ and consider what happens when we sort $A[p\,..\,r]$. Since at each recursion level each of the elements is chosen as the pivot with equal probability, the expected running time $T(n)$ satisfies:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ \dfrac{1}{n} \displaystyle\sum_{i=1}^{n} (T(i-1) + T(n-i)) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Hence, we get

$$T(n) = \frac{2}{n} \sum_{i=0}^{n-1} T(i) + \Theta(n)$$

$$\vdots$$

$$T(n) = \Theta(n \lg n).$$

SGI STL: Introsort

```
template <typename Ran, typename T, typename Size, typename Comparator>
void introsort_loop(Ran first, Ran last, T*, Size depth_limit, Comparator less) {
  while (last - first > stl_threshold) {
    if (depth_limit == 0) {
      partial_sort(first, last, last, less);
      return;
    }
    --depth_limit;
    Ran cut = unguarded_partition(first, last,
      T(median(*first, *(first + (last - first)/2), *(last - 1), less)), less);
    introsort_loop(cut, last, (T*) 0, depth_limit, less);
    last = cut;
  }
}

template <typename Ran, typename Comparator>
inline void sort(Ran first, Ran last, Comparator less) {
  if (first != last) {
    introsort_loop(first, last, VALUE_TYPE(first), lg(last - first) * 2, less);
    final_insertion_sort(first, last, less);
  }
}
```

Order statistic

The kth **order statistic** of a sequence of n elements is the kth largest element.

The **minimum** and the **maximum** are respectively the first and the nth order statistic.

A **median** is the "halfway point". Medians occur at $k = \lfloor (n+1)/2 \rfloor$ and $k = \lceil (n+1)/2 \rceil$. For odd n, there is a unique median—it is the $((n+1)/2)$th order statistic. For even n, the medians are the $(n/2)$th and the $(n/2+1)$st order statistic.

Selection

Input: A sequence A of n (distinct) elements, an integer $k \in \{1, \ldots, n\}$, and an ordering function $\otimes$ returning true or false.

Task: Report the kth largest element in A with respect to $\otimes$, i.e. the kth order statistic of A. (Normally, side-effects are allowed so that in A the order of elements can be modified.)

Example: If $A = \langle 5, 8, 1, 7, 9, 4, 2, 3, 6 \rangle$ and $\otimes$ the normal less-than comparison function for integers, report 1 when $k = 1$, 5 when $k = 5$, etc.

Trivial solution

Initial call: Select-With-Sort(A, 1, k, $A.length$)

Select-With-Sort(A, p, q, r)
1 Sort(A, p, r)
2 **return** $A[q]$

Running time: $\Theta(n \lg n)$, where $n = r - p + 1$.

Randomized selection algorithm

Initial call: Randomized-Select(A, 1, k, $A.length$)

Randomized-Select(A, p, q, r)

if $p = r$
    return $A[p]$
$p' =$ Pivot(A, p, r)
$q' =$ Partition(A, p, p', r)
if $q < q'$
    return Randomized-Select(A, p, q, $q'-1$)
if $q = q'$
    return $A[q]$
return Randomized-Select(A, $q'+1$, q, r)

Analysis of Randomized-Select

Let $T(n)$ denote an upper bound on the expected running time on a sequence of n elements. At each recursion level each of the elements is chosen as the pivot with equal probability. Thus, we get the recurrence

$$T(n) \le \begin{cases} \Theta(1) & \text{if } n = 1 \\ \dfrac{1}{n}\displaystyle\sum_{k=1}^{n} \max\{T(k-1), T(n-k)\} + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

Using the substitution method, one can show that

$$T(n) = \Theta(n)\,.$$

Sieving (prune and search)

Idea: Search for an element from the set of possible answers and use it to make the set smaller. Do this repeatedly until the final answer is found.

Example problem: Prime numbers

Input: A positive integer n, $n \geq 2$.

Output: All the prime numbers in $\{1, 2, \ldots, n\}$.

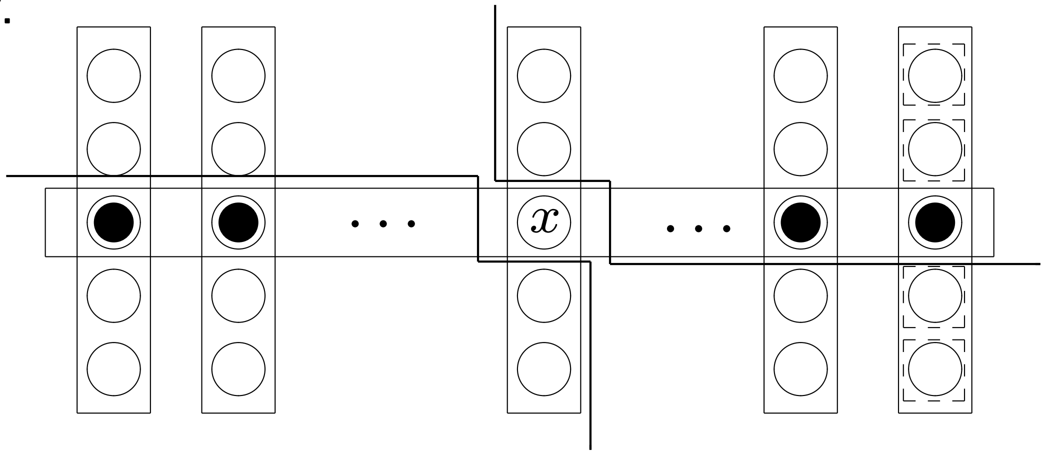
```
Eratosthenes-sieve(n)
C = {2, ..., n}
P = ∅
while C ≠ ∅
    Select the smallest element e from C
    P = P ∪ {e}
    Remove all multiples of e from C
return P
```

Online exercise (1 min)

What can you say about the running time of Eratosthenes-sieve?

Selection by sieving

1. Divide the n elements into groups of five, where only the last group can have less than 5 elements.
2. Find the median of each of the $\left\lceil \frac{n}{5} \right\rceil$ groups using insertion sort.
3. Move the medians into the beginning of the input array.
4. Recursively compute the median x of these medians.
5. Use x as the pivot to run Partition.
6. Pick the part that contains the desired order statistic, and recur if necessary.



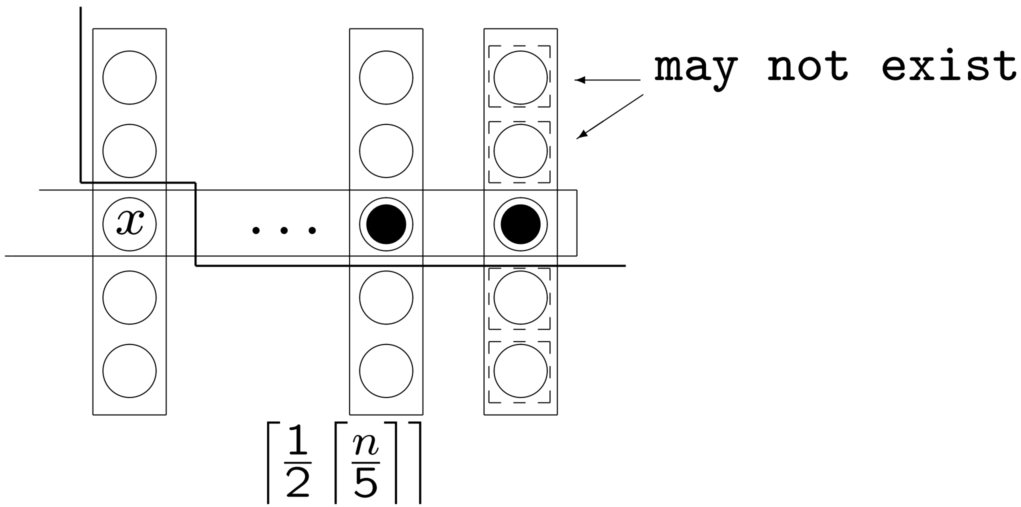
More detailed description

Initial call: Select(A, 1, k, $A.length$)

```
Select(A, p, q, r)
if r-p+1 < 90
    Merge-Sort(A, p, r)
    return q
for i = p to r step 5
    Insertion-Sort(A, i, min(i+4, r))
i = p-1
for j = p+2 to r step 5
    i = i+1
    exchange A[i] with A[j]
p' = ⌊(p+i)/2⌋
q' = Select(A, p, p', i)
q' = Partition(A, p, q', r)
if q < q'
    return Select(A, p, q, q'-1)
if q = q'
    return q
return Select(A, q'+1, q, r)
```

Analysis of Select

The number of groups is $\left\lceil \frac{n}{5} \right\rceil$. A half of the groups contribute 3 elements larger than x, except that the one consisting of less than 5 elements may contribute just 1 and that x's group has one less element to contribute.



So the number of elements larger than x is at least $3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil\right) - 3 \geq \frac{3n}{10} - 3$. The same holds for the number of elements smaller than x. Thus, the size of the subarray to be examined becomes at most $\frac{7n}{10} + 3$.

Analysis cont.

Note that, for every $n \geq 90$, $\frac{7n}{10}+3 \leq \frac{11n}{15}$ and $\left\lceil \frac{n}{5} \right\rceil \leq \frac{7n}{30}$. Suppose that we will resolve the problem without recursive calls when $n < 90$, e.g. by merge sorting. Thus, we get the following recurrence for the worst-case running time $T(n)$ of Select.

$$T(n) \leq \begin{cases} c_1 n \lg n < 7c_1 n & \text{if } n < 90 \\ c_2 n + T(m_1) + T(m_2) & \text{if } n \geq 90\,, \end{cases}$$

where c_1 and c_2 are constants, and $m_1 \leq \frac{7n}{30}$ and $m_2 \leq \frac{22n}{30}$. We can solve the recurrence by the substitution method. Assume that, for some constant d, $T(m) \leq dm$ for all positive $m < n$. Using the inductive assumption, we get that for $n \geq 90$

$$T(n) \leq c_2 n + \frac{29dn}{30}\,.$$

Setting d to $\max(7c_1, 30c_2)$ makes the inductive assumption work for all n. That is, $T(n) = O(n)$.

Conclusions

You have seen two types of algorithms:

- A randomized algorithm that makes random choices in the course of its execution.
- A deterministic algorithm that, for a given input, always produces the same output and follows the same execution trace.

And three types of analyses:

- worst-case analysis,
- best-case analysis, and
- average-case analysis.

Errors in the textbook

p. 170 Quicksort was claimed to be an in-place sorting algorithm; this is not true since the algorithm requires a recursion stack.

p. 181 and elsewhere In the analysis of quicksort, quickselect, and select the elements were assumed to be distinct. This is seldom the case in practice.

p. 217 and p. 221 The recurrences being solved were assumed to be monotonically increasing, but this assumption was not justified.

Summary

After reading §§7 and 9, you should know the following concepts:

- randomization
- quicksort
- quickselect
- sieving
- linear-time selection algorithm