Algorithms

- Instructors:
  - Vladimir Kolmogorov
  - Krzysztof Pietrzak

- Teaching assistant:
  - Asya Pentina

- Course webpage:
  http://pub.ist.ac.at/courses/AY1314/Algorithms_F13/

- Final grade = 50% homeworks + 50% exam

- HW1: available today, due next Monday before the lecture

Topics

- Design techniques
  1. Divide-and-Conquer (VK)
  2. Prune-and-Search (VK)
  3. Dynamic programming (KP)
  4. Greedy algorithms (KP)

- Search trees, priority queues
  5. Binary search trees (KP)
  6. Amortized analysis (KP)
  7. Heaps, heapsort (VK)

- Graph algorithms (VK)
  8. Graph search
  9. Shortest paths

- String algorithms
  10. Knuth-Morris-Pratt alg. (KP)
  11. Data structures for strings (VK)

- “Hard” problems
  12. Concluding lecture (KP)

Big O notation

Complexity $O(f(n))$:

Algorithm’s runtime on an input of size $n$ is at most $c \cdot f(n)$ for some constant $c > 0$, if $n$ is sufficiently large (i.e. if $n > N$ for some constant $N$)

- Sorting $n$ numbers:
  - Merge sort: $O(n \log n)$
  - Insertion sort: $O(n^2)$

- Querying $i$-th element of array $A[1..n]$: $O(1)$

- $O(n \log n)$ alg. typically faster than $O(n^2)$ alg. for sufficiently large inputs
Big O notation

\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = c > 0 \quad \Rightarrow \quad O(f(n)) = O(g(n)) \]

- \( O(an^2 + bn + c) = O(n^2) \) \((a > 0)\)
- \( O(an \log n + bn + c) = O(n \log n) \)

Divide-and-conquer

- Divide the problem into several subproblems
- Solve each subproblem recursively
- Combine solutions

- Applications:
  - Sorting (QuickSort, MergeSort)
  - Fast Fourier Transform (FFT)
  - Matrix multiplication (Strassen’s algorithm)
  - ...

The sorting problem

- Input: sequence of items
  - e.g. integers, words in a dictionary, ...
  - stored in an array

  \[ 5 \quad 4 \quad 3 \quad 7 \quad 2 \quad 9 \quad 1 \quad 8 \quad 1 \]

- Output: sorted sequence

  \[ 1 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 7 \quad 8 \quad 9 \]
void QuickSort(int left, int right)

if left < right then {
    i = Split(left, right);
    QuickSort(left, i - 1);
    QuickSort(i + 1, right);
}

Divide
Conquer
Combine:
not needed

int Split(int left, int right)

• Can be implemented in-place
  - no extra memory allocated

• Idea:
  - make sure that elements $x < 5$ come before elements $y > 5$
int Split(int left, int right)

pivot = A[left];  i = left;  j = right + 1;
while TRUE do {
deco j -- while i < j and A[j] ≥ pivot;
deco i ++ while i < j and A[i] ≤ pivot;
if i < j then exchange A[i] and A[j];
else {
exchane A[left] and A[i];
return i;
}
}
Running time

- Running time: sum of all lengths

Running time: worst case

- Already sorted sequence:

\[ T(n) = n + T(n-1) \]
\[ = n + (n-1) + \ldots + 1 = \frac{n(n+1)}{2} \]

- Quadratic running time: \( T(n) = O(n^2) \)

  - i.e. \( T(n) \leq const \cdot n^2 \) for some \( const \)

Running time: best case

\[ T(n) = n + 2 \cdot T \left( \frac{n-1}{2} \right) \]

- Master theorem (see e.g. wikipedia):
  recipe for solving such recurrences

- Not covered in this course
Running time: best case

\[ T(n) = n + 2 \cdot T \left( \frac{n-1}{2} \right) \]

\[ n = 2^k - 1 : \]

\[ T(n) = n - (2^0 - 1) + n - (2^1 - 1) + n - (2^2 - 1) + \ldots + n - (2^{k-1} - 1) \]

\[ = k n - (2^k - 1) + k \]

For general \( n \):

\[ T(n) = O(n \log n) \]

\[ T(n) \leq \text{const} \cdot n \log n \]
Running time

- QuickSort: runtime depends on input data
  - worst case: \( O(n^2) \) (e.g. if already sorted)
  - best case: \( O(n \log n) \)

- Randomized QuickSort: randomly select pivot

```c
int rSplit(int left, int right)
{
  p = Random(left, right);
  exchange A[left] and A[p];
  return Split(left, right);
}
```

Randomized QuickSort - Average complexity

- \( T(n) \): expected runtime for \( n \) elements
- Assumption: all elements are distinct

\[
T(n) = n + \frac{1}{n} \cdot \sum_{m=0}^{n-1} (T(m) + T(n - m - 1))
\]

Randomized QuickSort - Average complexity

- \( T(n) \): expected runtime for \( n \) elements
- Assumption: all elements are distinct

\[
T(n) = n + \frac{1}{n} \cdot 2 \cdot \sum_{i=0}^{n-1} T(i)
\]
Randomized QuickSort - Average complexity

\[ nT(n) = n^2 + 2 \cdot \sum_{i=0}^{n-1} T(i) \]

\[ (n - 1)T(n - 1) = (n - 1)^2 + 2 \cdot \sum_{i=0}^{n-2} T(i) \]

\[ nT(n) - (n - 1)T(n - 1) = n^2 - (n - 1)^2 + 2T(n - 1) \]

Randomized QuickSort - Average complexity

\[ \frac{T(n)}{n + 1} = \frac{T(n - 1)}{n} + \frac{2n - 1}{n(n + 1)} \]

\[ U(n) = \frac{T(n)}{n + 1} \]

\[ \frac{1}{U(n)} = \frac{1}{U(n - 1)} \]

Randomized QuickSort - Average complexity

\[ U(n) = U(n - 1) + \frac{2n - 1}{n(n + 1)} \]

\[ U(n) = \frac{T(n)}{n + 1} \]

\[ = \sum_{i=1}^{n} \frac{2i - 1}{i(i + 1)} \]

\[ = 2 \sum_{i=1}^{n} \frac{1}{i + 1} - \sum_{i=1}^{n} \frac{1}{i(i + 1)} \]
Randomized QuickSort - Average complexity

\[ \frac{T(n)}{n+1} = 2 \sum_{i=1}^{n} \frac{1}{i+1} - \sum_{i=1}^{n} \frac{1}{i(i+1)} \]

\[ \sum_{i=1}^{n} \left( \frac{1}{i} - \frac{1}{i+1} \right) = 1 - \frac{1}{n+1} \]

Randomized QuickSort - Average complexity

\[ \frac{T(n)}{n+1} < 2 \sum_{i=1}^{n} \frac{1}{i+1} < \int_{1}^{n+1} \frac{dx}{x} \]

\[ = 2 \log(n+1) \]

Randomized QuickSort - Average complexity

- Complexity: \( T(n) = O(n \log n) \)

\[ T(n) < 2 \cdot (n + 1) \cdot \log(n + 1) \]

- Approximately \( \frac{2}{\log_2 e} \approx 1.386 \ldots \) slower than the best case
Stack & extra space

```java
void QuickSort(int left, int right) {
    if left < right then {
        i = Split(left, right);
        QuickSort(left, i - 1);
        QuickSort(i + 1, right);
    }
}
```

- Worst-case stack size: \( O(n) \)
- **QuickSort** is tail-recursive
  - as the last step calls itself
  - naive implementation (with stack) inefficient

Removing tail recursion

```java
void QuickSort(int left, int right) {
    while left < right do {
        i = Split(left, right);
        QuickSort(left, i - 1);
        left = i + 1;
    }
}
```

- Worst-case stack size: still \( O(n) \) ...

Removing tail recursion for larger side

```java
void QuickSort(int left, int right) {
    while left < right do {
        i = Split(left, right);
        if i - left < right - i then
            QuickSort(left, i - 1); left = i + 1;
        else
            QuickSort(i + 1, right); right = i - 1;
        endif
    }
}
```

- Worst-case stack size: \( O(\log n) \)
Summary

- Deterministic QuickSort:
  - Worst-case: $O(n^2)$
  - Average over data instances (random permutations): $O(n \log n)$
- Randomized QuickSort:
  - Worst-case: $O(n^2)$
  - Average: $O(n \log n)$ [assuming distinct elements]
- Extra space (worst-case): $O(\log n)$
  - Use recursive call only for the smaller side
- One of the fastest sorting algorithms in practice

- Techniques:
  - Divide-and-conquer
  - Randomization