Divide and Conquer Algorithms I
Script of Lecture T3

Amalia Duch

October 3, 2019

Contents

- Divide and Conquer Scheme.
- Fibonacci Numbers Revisited.
- Merge-sort: Quick Reminder and Cost Analysis.
- Lower bound for the worst-case cost of sorting algorithms based on comparisons.
- Quick-sort: Algorithm and implementation.

Fibonacci numbers revisited

- How to solve the recurrence for Fibonacci numbers in order to obtain the exact value of the n-th Fibonacci number.
- How to calculate the n-th Fibonacci number in $\Theta(\log(n)$ time using the fast exponentiation algorithm.

Merge_sort

Quick remind on how the algorithm works, how it is implemented and analysis of the cost in all cases $\Theta(n\log(n))$.

Lower bound for the worst-case cost of sorting algorithms based on comparisons.

The sorting scheme (or method) of merge-sort and, in general, of all comparison-based sorting algorithms can be represented by means of a decision tree.

Example: Decision tree to sort a vector of three elements a1, a2, a3.

Lemma 1. *The decision tree of any sorting algorithm based on comparisons of n elements must have at least $n!$ leaves.*

Proof. The argument is valid because of the correctness of the algorithm.
A *binary tree* is either an empty tree, or a node with two binary trees as children. The *depth* of a node in a binary tree is the number of edges in the path from the node to the root of the tree. The depth of the root of a binary tree is 0.

Lemma 2. *A binary tree –whose deepest node has depth d– has at most 2^d leaves.*

The proof is by induction on d.

Lemma 3. *The worst-case cost of any comparison-based sorting algorithm is $\Omega(n \log(n))$.*

Proof: Because of Lemma 1 the decision tree of a sorting algorithm based on comparisons must have at least $n!$ leaves. Because of Lemma 2 the depth d of the deepest node (which is the cost in the worst-case) must be $d > \log(n!) > \log((n/2)^{(n/2)}) = (n/2) * \log(n/2) = \Omega(n \log(n))$.

Theorem 1. *The worst-case cost of Merge-sort is optimal (asymptotically).*

Quick-sort

Algorithm, example and implementation.

```
int partition(vector<elem>& v, int l, int r) {
        int p = v[0];
        int i = l-1;
        int j = r+1;
        for (;;) {
            while (v[++i] < p);
```

```
            while (v[--j] > p);
            if (j <= i) return j;
            swap(v[i],v[j]);
        }
}

void quick_sort(vector<elem>& v, int l, int r) {
    if (l >= r) return;
    int p = partition(v,l,r);
    quick_sort(v,l,p);
    quick_sort(v,p+1,r);
}
```