Divide and Conquer

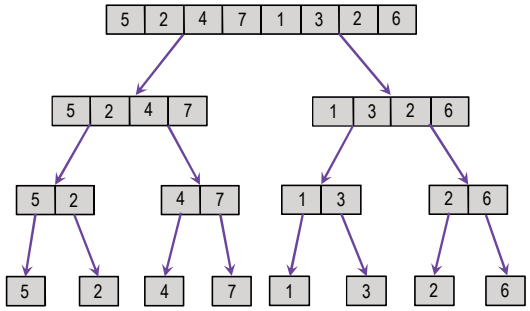
Design and Analysis of Algorithms
Andrei Bulatov

Divide and Conquer, MergeSort

Recursive algorithms: Call themselves on subproblem
Divide and Conquer algorithms:
- Split a problem into subproblems (divide)
- Solve subproblems recursively (conquer)
- Combine solutions to subproblems (combine)

MergeSort
- Divide: Split a given sequence into halves
- Conquer: By calling itself sort the two halves
- Combine: Merge the two sorted arrays into one

MergeSort



MergeSort

MergeSort(A,p,r)
Input: array A, positions p,r
Output: array A such that entries A[p],…,A[r] are sorted
Method:

```
if  p<r then do
    set q:=⌊(p+r)/2⌋
    MergeSort(A,p,q)
    MergeSort(A,q+1,r)
    Merge(A,p,q,r)
endif
```


Merge

The Merge procedure is applied to array A and three positions p, q, r in this array
Assume
$p \le q < r$
A[p], …, A[q] and A[q+1], …, A[r] are ordered
Outputs ordered sequence in positions A[p], …, A[r]

This sequence is generated by comparing the two elements on the top of subarrays and moving the smaller one

MergeSort: Running Time

The running time of Merge when applied to two arrays of total size n is $\Theta(n)$

The running time, T(n), of MergeSort is

$$T(n) = Cn + T(n/2) + T(n/2) + Dn$$

divide recursion Merge

If n = 1 then T(1) = C

MergeSort: Running Time

Recursion tree

There are 2^i nodes on level i

Each node requires $\frac{Cn+Dn}{2^i}$ work

Total work on each level: (C+D)n

There are log n levels

Theorem

The running time of MergeSort is $\Theta(n \log n)$

Substitution and Partial Substitution

If we can guess that $T(n) \leq (C + D)n \log n$, we can verify our guess:

$$\begin{aligned} T(n) &= 2\,T(n/2) + (C + D)\,n \\ &\leq 2(C + D)\,n/2 \log(n/2) + (C + D)\,n \\ &= (C + D)\,n \log(n/2) + (C + D)\,n \\ &= (C + D)\,n \log n - (C + D)\,n + (C + D)\,n \\ &= (C + D)\,n \log n \end{aligned}$$

Or we can do the same in order to find constant factors or parameters

Suppose our guess is $T(n) \leq cn \log n$

Master Theorem

Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, let f(n) be a function, and let T(n) be defined on the nonnegative integers by the recurrence

$$T(n) = a\,T(n/b) + f(n).$$

Then T(n) can be bounded asymptotically as follows:

1. If $f(n) \in O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$
2. If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \log n)$
3. If $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) \in \Theta(f(n))$

Counting Inversions

Comparing two rankings

A ranking is a permutation of some objects

Objects can be numbered, and one of the rankings is just the natural order

The Counting Inversions Problem

Instance:

A permutation $a_1, \ldots, a_n$ of numbers 1, ..., n

Objective:

Find the number of pairs i,j , $i < j$ such that $a_i > a_j$

Algorithm Idea

Straightforward algorithm takes $O(n^2)$ time

Use divide and conquer approach:

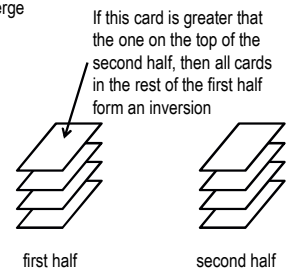
- split the sequence into two halves
- find the number of inversions in the halves
- then what?

Observation:

`Between halves' inversion have the form (a_i, a_j) where a_i is in the first half, a_j is in the second half, and $a_i > a_j$

Algorithm Idea (cntd)

Assuming that the two halves are sorted we can run a procedure similar to Merge



Algorithm

Merge-and-Count(A,B)

```
set curr1:=1, curr2:=1   /* current cards in halves
set count:=0             /* # of inversions
while curr1≠last1+1 and curr2≠last2+1
   if A[curr1]≤B[curr2] then do
      output A[curr1]
      set curr1:=curr1+1
   else do
      output B[curr2]
      set curr2:=curr2+1
      set count:=count+(last1-curr1+1)
   endif
endwhile
output the rest of the non-empty half and count
```


Algorithm (cntd)

Sort-and-Count(L)

```
If last=1 then no inversions
else do
   divide L into two halves:
     A contains the first ⌊last/2⌋ elements
     B contains the remaining elements
   set (p,A):=Sort-and-Count(A)
   set (q,B):=Sort-and-Count(B)
   set (r,L):=Merge-and-Count(A,B)
endif
output p+q+r and the sorted list L
```


Sort-and-Count

Theorem

The Sort-and-Count algorithm correctly sorts the input and counts the number of inversions.

It runs in $O(n \log n)$ time for a list of n elements