September 24, 2020

Sorting algorithms

- bubble sort (L)
- insertion sort (L)
- selection sort (P)

Quadratic time algorithms

Today → much more efficient one

Mergesort

→ Divide and conquer algorithm

General pattern

1. Divide the problem into smaller parts where these parts are similar to the original problem
2. Solve the subproblems reqursively (if the size of the subproblems are small enough, then solve them directly)
3. Combine the solutions of the subproblems to obtain the solution of the original problem

How can one apply this for sorting?

We want to sort $A[1:n]$

(1) Divide the problem into <u>TWO</u> parts of size $n/2$ (or $n/2+1$)

$A[1:n] \longrightarrow A[1:q]$ and $A[q+1:n]$

(2) Sort $A[1:q]$ and $A[q+1:n]$ recursively (if a subarray consists of 1 element only you don't have to do anything because a one element array is sorted)

(3) Now we have the sorted $A[1:q]$ and $A[q+1:n]$, we have to produce now the sorted $A[1:n]$.

The name of the algorithm completing this task is MERGE

MERGE

Input:

two sorted arrays $B[1:k]$ and $C[1:\ell]$

Output:

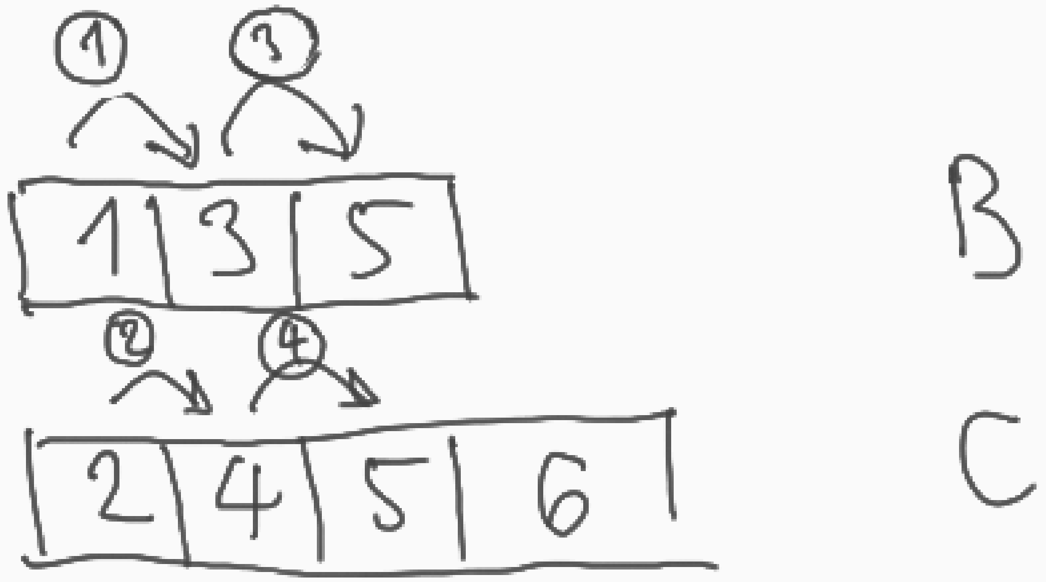
sorted array $A[1:k+\ell]$ consisting of the elements of B and C

Now

$$A[1] = \min(B[1], C[1])$$

In general, suppose that $B[1:i]$ and $C[1:j]$ are already handled and $A[1:i+j]$ contains their elements in increasing order. Then

$$A[i+j+1] = \min(B[i+1], C[j+1])$$



① $B[1] \overset{?}{<} C[1] \Rightarrow A[1] = B[1]$

② $B[2] \overset{?}{<} C[1] \Rightarrow A[2] = C[1]$

③ $B[2] \overset{?}{<} C[2] \Rightarrow A[3] = B[2]$

④ $B[3] \overset{?}{<} C[2] \Rightarrow A[4] = C[2]$

⑤ $B[3] \overset{?}{<} C[3] \Rightarrow A[5] = B[3]$

$A[6] = C[3]$

$A[7] = C[4]$

General technics for handling the "boundary" $\rightarrow$ sentinel

1	3	5	∞

2	4	5	6	∞

we extend the arrays by one slot containig a value bigger than any possible values we want to sort

Merge(A, B, C) $A[1:k+l]$, $B[1:k]$, $C[1:l]$

```
B[k+1] = ∞
C[l+1] = ∞
i := 1 , j := 1
for m = 1 to k+l do
    if B[i] ≤ C[j]
        then
          A[m] := B[i]
          i := i+1
        else
          A[m] := C[j]
          j := j+1
```

Complexity

$k+l$ comparisions : after each comparison one element will occupy its place in A

Now the Mergesort (recursive)

MergeSort (A, p, r) $\leftarrow$ (A, 1, n)

if $p < r$ then

$q = \left[\frac{p+r}{2}\right]$ $\qquad$ $q = p + \left[\frac{r-p}{2}\right]$ is better

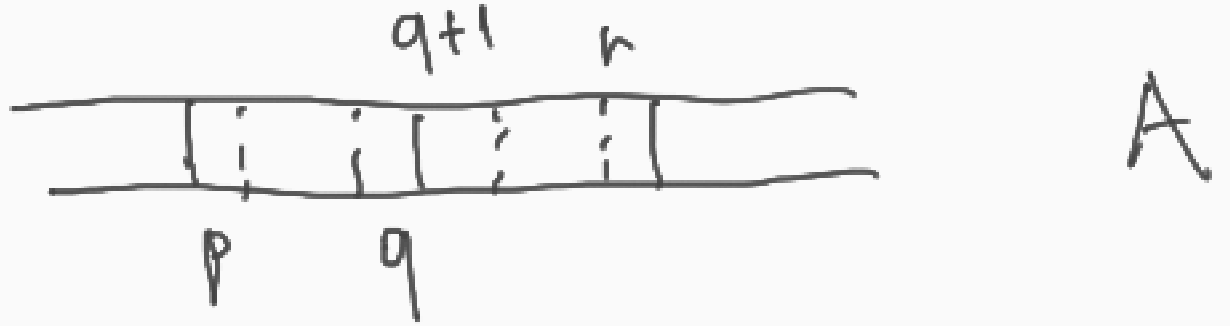
⋮

⋮

MergeSort (A, p, q)

MergeSort (A, q+1, r)

Merge (A, p, q, r)



Merge (A, p, q, r)

$k = q - p + 1$

$l = r - q$

for $i = 1$ to k do

  $B[i] := A[p + i - 1]$

for $j = 1$ to l do

  $C[j] = A[q + j]$

⋮

```
B[k+1] := ∞
C[l+1] := ∞
i := 1 , j := 1
for m = p to r do
    if B[i] ≤ C[j]
        then
            A[m] := B[i]
            i := i+1
        else
            A[m] := C[j]
            j := j+1
```

Complexity

Recursive algorithm $\Rightarrow$ Recursive cost function

For the sake of simplicity assume that n is a power of 2 $(n = 2^k)$

Now, for the number of comparisons we have

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{2 recursive calls}} + \underbrace{n}_{\text{merge}} & \text{if } n > 1 \end{cases}$$

How this cost is related to the cost of babblesort ?

We need a "closed form" to compare them.

$$
\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2\left(n/2 + 2T\left(\tfrac{n/2}{2}\right)\right) \\
&= n + n + 2^2 T(n/2^2) \\
&= 2n + 2^2 T(n/2^2) \\
&= 2n + 2^2\left(n/2^2 + 2T\left(\tfrac{n/2^2}{2}\right)\right) \\
&= 2n + n + 2^3 T(n/2^3)
\end{aligned}
$$

$$= 3n + 2^3 T(n/2^3)$$

$$= 3n + 2^3 \left(n/2^3 + 2T\left(\frac{n/2^3}{2}\right)\right)$$

$$= 3n + n + 2^4 T(n/2^4)$$

$$= 4n + 2^4 T(n/2^4)$$

$$\vdots$$

$$= kn + 2^k T(\underbrace{n/2^k}_{1}) \qquad T(1) = 0$$

$$= kn = n \log_2 n \qquad n = 2^k \Leftrightarrow k = \log_2 n$$

Conclusion

Merge sort in much faster than bubblesort because $n \log_2 n$ is much smaller than $\frac{n(n-1)}{2}$