


Estruturas de Dados / Programação 2
Algoritmos de Ordenação - Parte 2

Márcio Ribeiro
marcio@ic.ufal.br
twitter.com/marciomribeiro

Sorting cards

- Hand of playing cards
  - Remove a card from the table
  - Insert it into the correct position in the left hand
- Finding correct position
  - From right to left
  - Compare with each of the cards already in the hand

```
void insertion_sort(int v[], int size)
{
  int i, j, key;
  for (i = 1; i < size; i++) {
    key = v[i];
    j = i - 1;

    while ((j >= 0) && (v[j] > key)) {
      v[j+1] = v[j];
      j--;
    }
    v[j+1] = key;
  }
}
```

INSERTIONSORT
INSERTIONSORT
INSERTIONSORT
INSERTIONSORT
EINSRTIONSORT
EINRSTIONSORT
...

Efficiency

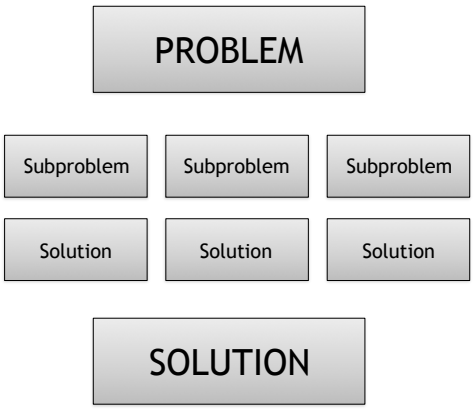
- Best case
  - Already sorted `v[j] <= key`
  - O(n)
- Worst case
  - Reverse sorted order
  - $O(n^2)$

```
…
for (i = 1; i < size; i++) {
  …
  while (… && (v[j] > key)) {
    …
  }
}
```

Divide and Conquer

Divide and Conquer

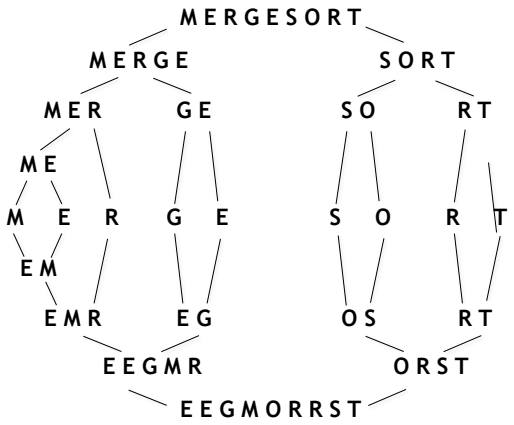
- Many useful algorithms are recursive
- They call themselves to solve subproblems
- Break the problem into several subproblems
  - Similar to the original problem
  - But smaller in size



Steps

- **Divide** the problem into a number of subproblems that are smaller instances of the same problem
- **Conquer** the subproblems by solving them recursively
- **Combine** the solutions to the subproblems into the solution for the original problem

Quick sort is a divide and conquer algorithm. Let's see another one?



Instantiating the steps

- **Divide:** the n-element array to be sorted in two sub arrays of n/2 elements each
- **Conquer:** sort the two sub arrays recursively using merge sort
- **Combine:** merge the two sorted sub arrays to produce the sorted answer

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

Assumptions:

A[p..q] and A[q+1..r] are already sorted

From Cormen et Al, Introduction to Algorithms, 3rd edition

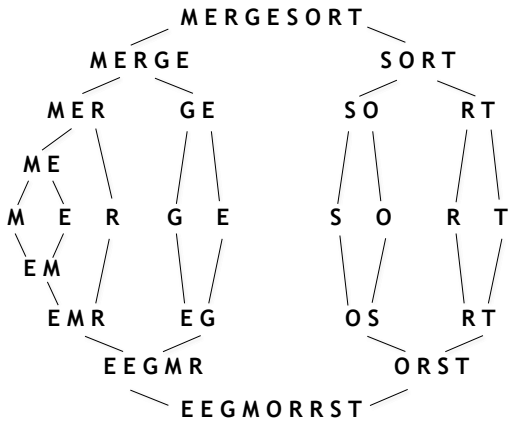
Exercise

- Using the previous algorithm, merge the two following arrays

2	4	5	7	∞

1	2	3	6	∞

Now we know how to merge. But, how to sort?



```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

From *Cormen et Al, Introduction to Algorithms, 3rd edition*

Execution: Merge Sort

6 5 3 1 8 7 2 4

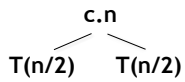
Efficiency

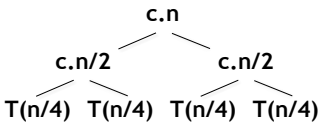
Last class we used substitution to solve the recurrence equation

Now, let's see another method!

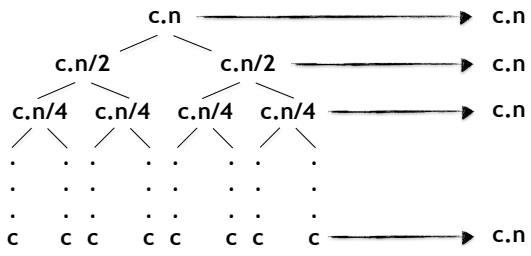
Recursion Tree

T(n)

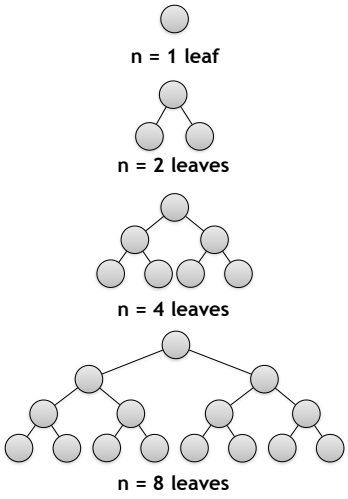




Recursion Tree



How many levels?



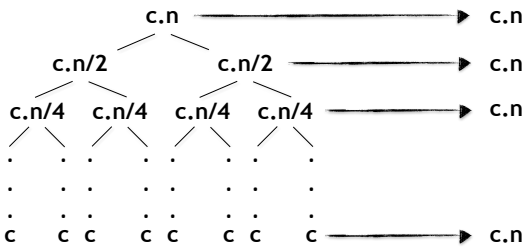
$\log_2(1) + 1 = 1$ level

$\log_2(2) + 1 = 2$ levels

$\log_2(4) + 1 = 3$ levels

$\log_2(8) + 1 = 4$ levels

Levels = log(n) + 1

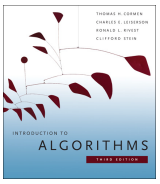


Each level costs c.n

We have c.n . (log(n) + 1) times, i.e., c.n.log n + c.n

O(n.log n)

References



Chapter 2

Chapter 9