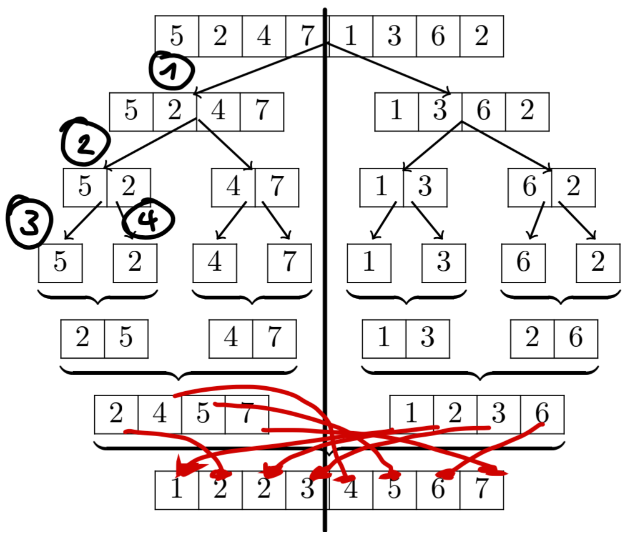
Divide and conquer

Divide and conquer has 3 central steps

- Divide into subproblems
- Concquer subproblems with recursive solution
- Combine solution of subproblems into solution for the original problem

Merge sort

- Divide: divide the sequence into two $n/2$-element sequences
- Conquer: sort the two sequences recursively using merge sort
- Combine: merge the two sorted sequences to produce the solution




```
Algorithm MergeSort(l, h)
{
    if(l<h)
    {
        mid = (l+h)/2;
        MergeSort(l, mid)
        MergeSort(mid + 1, h)
        Merge(l, mid, h)
    }
}
```

Time complexity for Merge Sort is $\Theta(n \log n)$. The logic behind this is determined from the recursion calls. With 8 elements split into “groups” of 2 elements each, we have 3 levels, where a merge happens ($2^3 = 8$).

Reccurences

The running times of algorithms with recursive calls can be described using recurrences. There exist 4 basic ways how we can get to the reuccurence formula:

- Repeated (backward) substitution: Expand the recurrence by substitution and then notice the pattern
- Substitution method: Guess a bound and then use induction to prove that the guess is correct
- Recursion trees: Convert a recurrence in a tree whose nodes represent the costs
- Master method: Templates for different classes of recurrences

Repeated substitution

```
T (n)
= 2T (n/2) + 2n + 3                    // Substitute
= 2(2T (n/4) + n + 3) + 2n + 3         // Expand
= 4T (n/4) + 4n + 9                    // Substitute
= 4(2T(n/8) + n/2 + 3) + 4n + 9        // Expand
=8T(n/8)+3·2n+(4+2+1)3                 // Find Pattern
```

The pattern we see here is $2^i T(\frac{n}{2}) + 2in + 3 \sum_{i-1}^{j=0} 2^j$ - to clarify, the i and j stand for the number of substituions done. In $8T(\frac{n}{2}) + 3 \cdot 2n + (4+2+1)3$, i and j would be equal to 3.

We set the upper bound for i to $log\, n$, insert this for i and can then calculate the asymptotic bound for the function, which is $\Theta(n \log n)$.

Substitution method

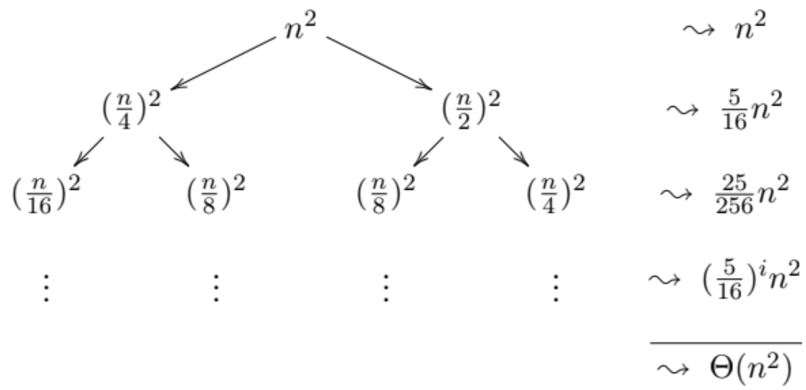
```
T (n)   = 4T (n/2) + n  (recurrence)
<= 4c(n/2)^3 + n         (inductive hyp.)
= cn^3/2 + n            (simplification)
= cn^3 - (cn^3/2 - n)   (rearrangement)
<= cn3                   (for c >= 2, n >= 1)
```

We assume an inductive hypothesis, in this example we assume that $T(n) = n^3$. We then check if our hypothesis is true.

Recursion trees

We visualize the recursion tree to see what happens when the recurrence is iterated.

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



Master method

Decreasing recurrences

Few examples:

$T(n) = T(n-1) + 1 \rightarrow O(n)$

$T(n) = T(n-1) + n \rightarrow O(n^2)$

$T(n) = T(n-1) + log\, n \rightarrow O(n\, log\, n)$

$T(n) = 2T(n-1) + 1 \rightarrow O(2^n)$

$T(n) = 3T(n-1) + 1 \rightarrow O(3^n)$

$T(n) = 2T(n-1) + n \rightarrow O(n \cdot 2^n)$

What we are seeing here, can be declared into a formula:

$T(n) = a \cdot T(n-b) + f(n)$, where $a, b > 0$ and $f(n) = O(n^k), k \geq 0$

Furthermore, we apply following rules:

- $a < 1 = O(n^k) = O(f(n))$
- $a = 1 = O(n^{k+1}) = O(n \cdot f(n))$
- $a > 1 = O(n^k \cdot a^{\frac{n}{b}}) = O(f(n) \cdot a^{\frac{n}{b}})$

Case 1, 2 and 3

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

$a \geq 1$
$b > 1$ $f(n) = O(n^k \cdot \log^p n)$

① $\log_b a$

② k

case 1: $\log_b a > k$ then $\Theta(n^{\log_b a})$

case 2: $\log_b a = k$
- $p > -1$: $\Theta(n^k \cdot \log^{p+1} n)$
- $p = -1$: $\Theta(n^k \cdot \log \log n)$
- $p < -1$: $\Theta(n^k)$

case 3: $\log_b a < k$
- $p \geq 0$: $\Theta(n^k \cdot \log^p n)$
- $p < 0$: $\Theta(n^k)$

case 1	case 2	case 3
$T(n) = 8 \cdot T(n/2) + n$	$T(n) = 2\,T(n/2) + n$	$T(n) = T(n/2) + n^2$
$\log_2 8 = 3 > k = 1$	$\log_2 2 = 1 \Leftrightarrow k = 1$ $(p = 0)$	$\log_2 1 = 0 < k = 2$
$\Theta(n^3)$	$\Theta(n \cdot \log n)$ $p \geq -1$ when $\log n$ is denominator	$\Theta(n^2)$