std::vector

```
#include <vector>
...
std::vector<type> vec_name;

// some examples
std::vector<bool> first; // empty vector of bools
std::vector<int> second (4,0); // 4 ints with value 0
std::vector<int> third (second); // a copy of second
```

https://cplusplus.com/reference/vector/vector/vector/ and

https://en.cppreference.com/w/cpp/container/vector/vector

Nützliche Funktionen von std::vector

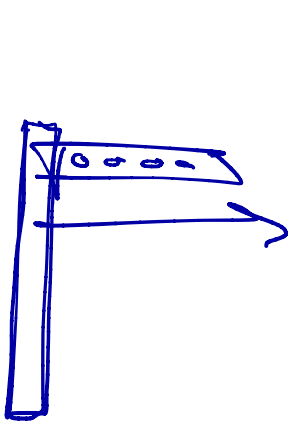
Auf `https://www.geeksforgeeks.org/vector-in-cpp-stl/` werden unter "Capacity", "Modifiers" und "Element access" nützliche Funktionen beschrieben die ein Vektor euch anbietet. Hier einige davon:

- `push_back(element)`
- `at(index)`
- `size()`

```
for (unsigned int i = 0; i < letters.size(); ++i) {
    std::cout << letters.at(i);
}
```

Matrizen als 2D Vektoren

```
// direkt unter den includes:
using irow = std::vector<int>;
using imatrix = std::vector<irow>;

// e.g. in einer Funktion:
std::vector<std::vector<int>> matrix;
```



transpose_matrix (c , irow (r, 0))

FYI: using, typedef

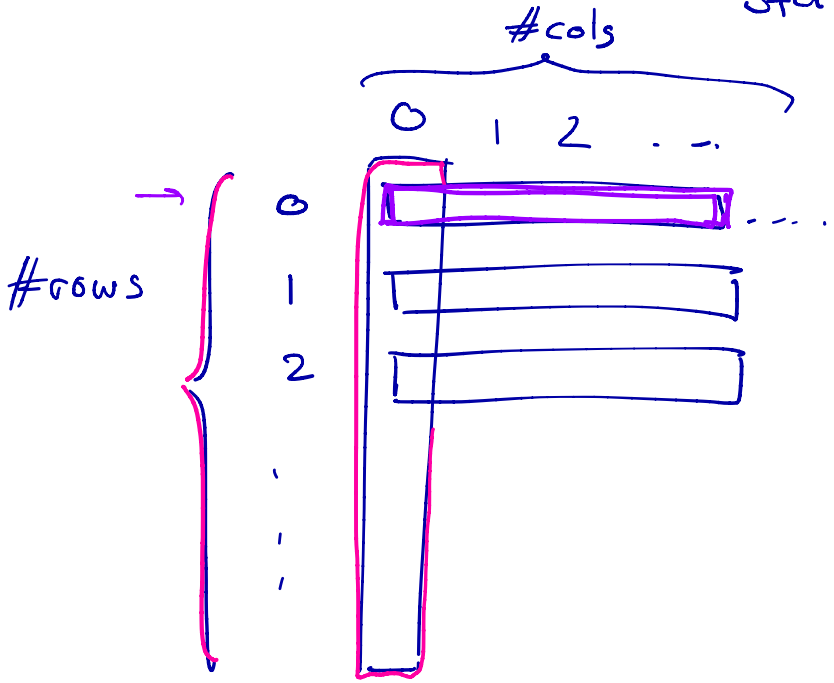
(Denke nicht, dass Ihr zu dem viel wissen müsst.)

```
typedef int MyIntA;
using MyIntB = int;

using imatrix = std::vector<std::vector<int>>;
```

- Heisst: MyIntA bzw. MyIntB könnt Ihr als int verwenden
- Oder eben: Ihr könnt jetzt im Code imatrix verwenden und Ihr meint damit `std::vector<std::vector<int>>`

Matrizen als 2D Vektoren

```
std::vector< std::vector<int> > V;
```



#rows $\hat{=}$ `V.size()`

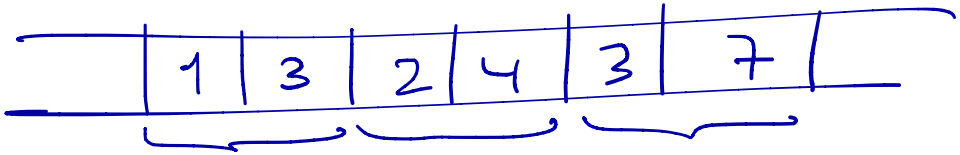
#cols $\hat{=}$ `V[0].size()`

`V.at(0).size()`

`V(0).p_b`

Matrizen als 2D Vektoren

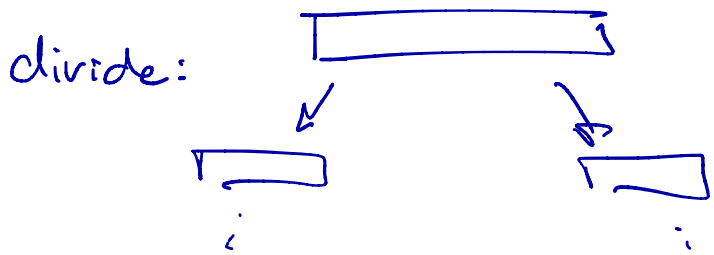
row vs. col. major

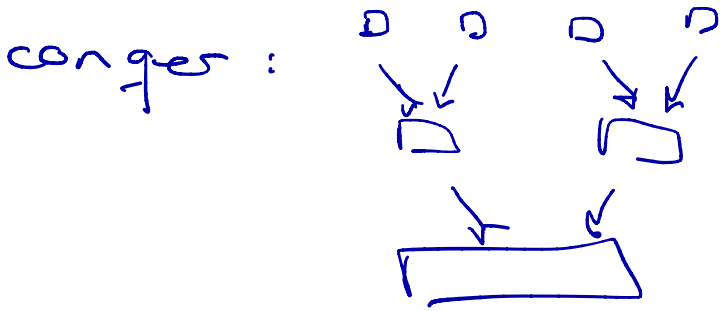



```
int main() {
  imatrix m = {
    {1, 3},
    {2, 4},
    {3, 7},
  };

  m[0][1] = 10;
  m[1][0] = 20;

  print(m);
}
```

```
Output:
1   10
20  4
3   7
```

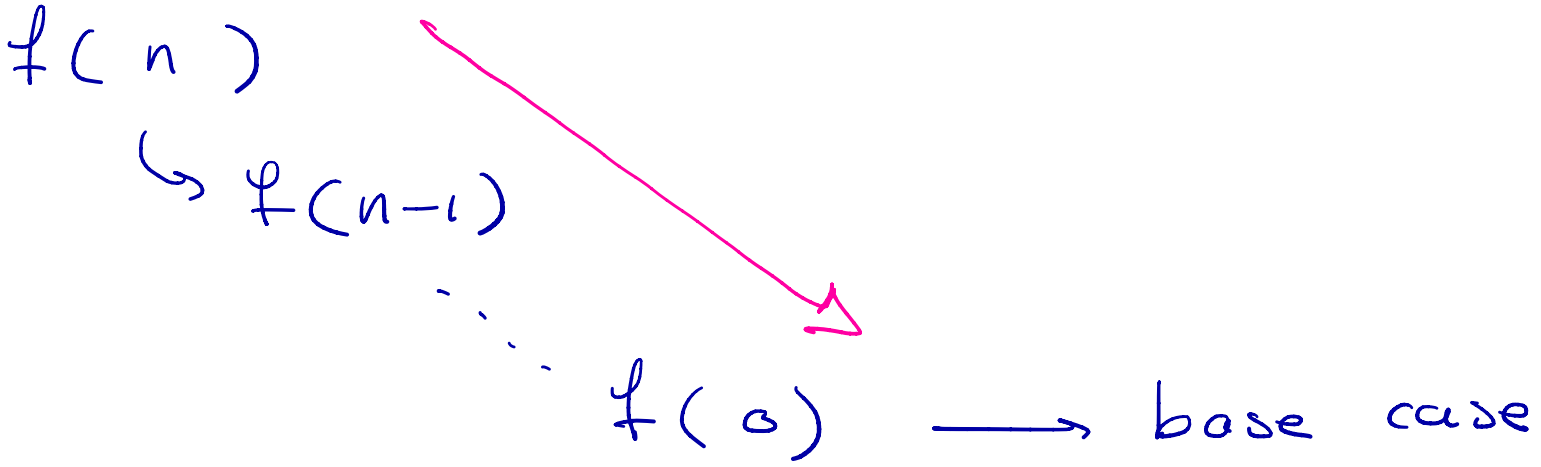
Rekursion



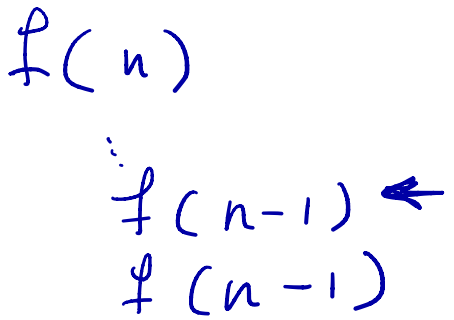


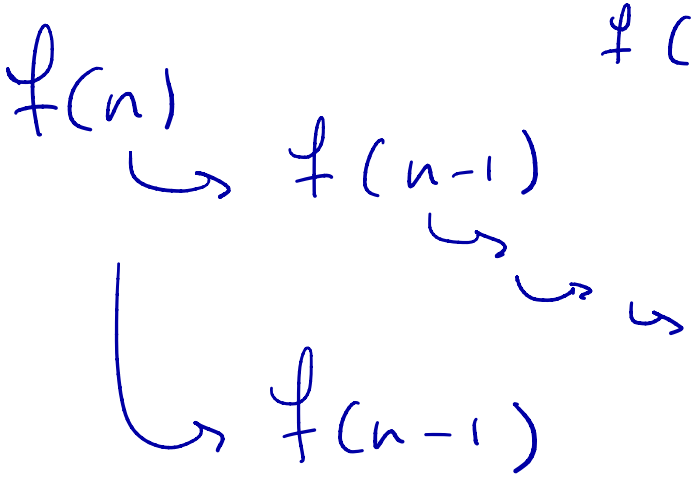
Divide & Conquer (& Combine)

- Divide: Problem ist mit der Grösse n schwer zu Lösen $\rightarrow$ Problem auf kleinere Teilprobleme aufteilen und dann die einfacheren Teilprobleme einzeln Lösen.
- Conquer: Sobald ein Teilproblem einfach genug ist, können wir es Lösen. (E.g. es kann sein, dass wir öfters das Problem verkleinern müssen, bis es "einfach" ist.)
- Combine: Lösung für das Problem mit der ursprünglichen Grösse aus den einzelnen Lösungen der Teilprobleme zusammensetzen.



1) Problem? (Grösse?)
2) base case?)
3) how to get to bc.?
4) Subtasks?





Rekursion

```
void function(input size){
   // base case
      ...
      function(smaller input size) // recursive call here
      ...
}
int main(){
      ...
      function(input size)
      ...
}
```

Stop recursion

Rekursion: function calls

↳ example how to "count" recursive calls

↳ function calls continue to first base case and only then stop for lot time