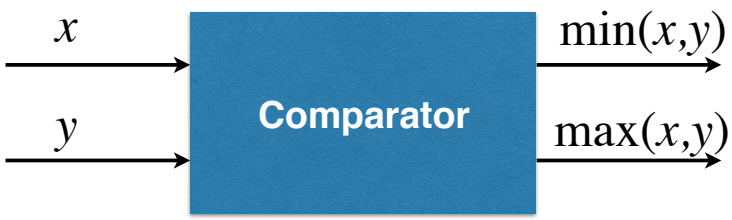
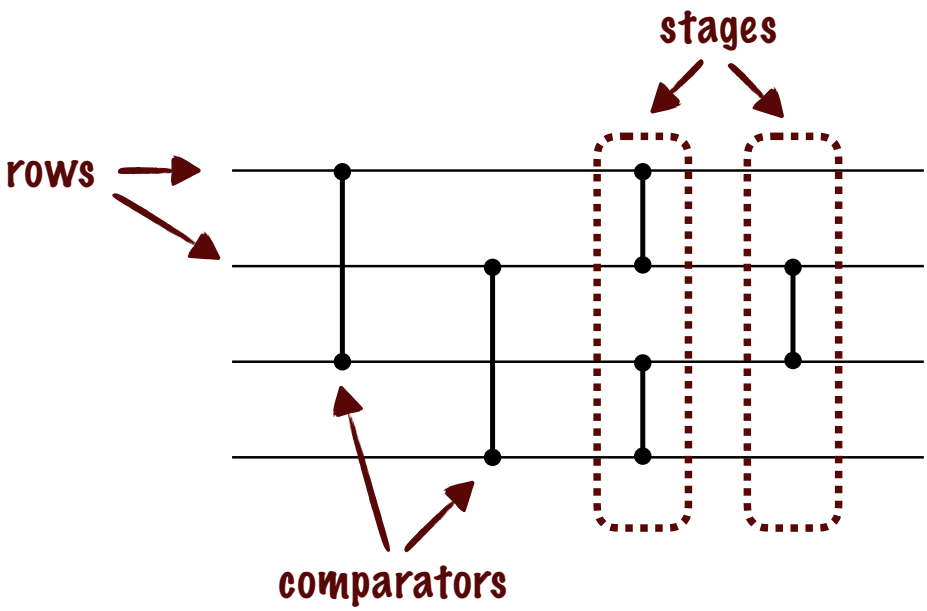
Modular Program Design

Advanced Programming

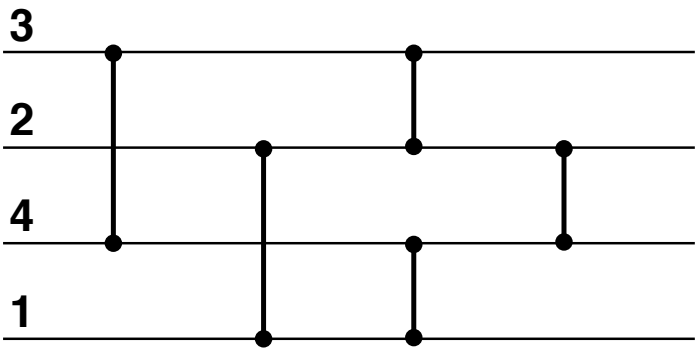
Sorting Networks



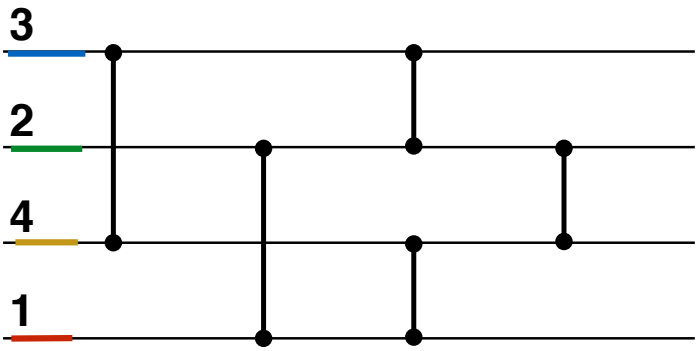
Sorting Networks



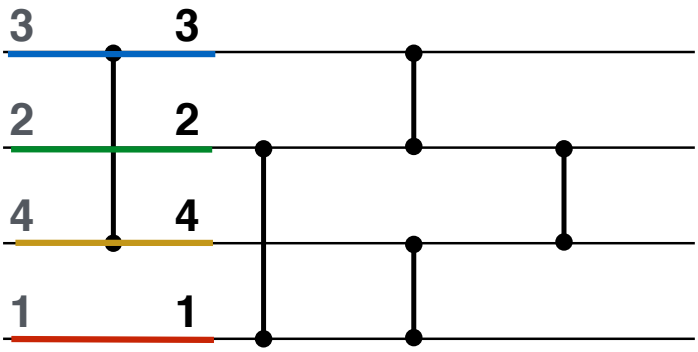
Sorting Networks



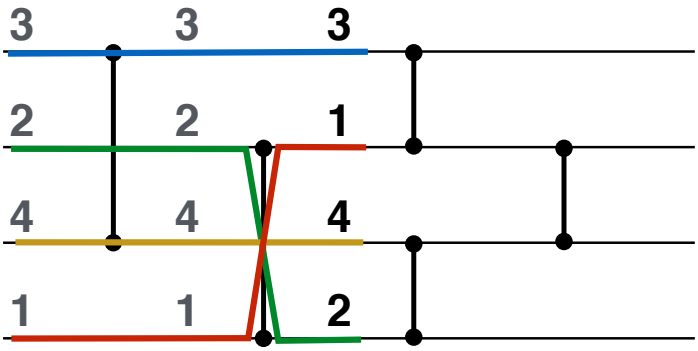
Sorting Networks



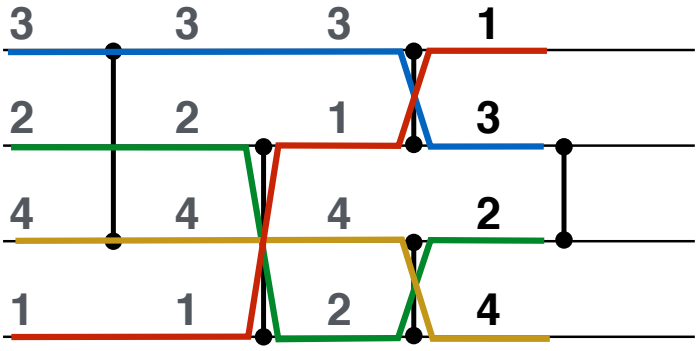
Sorting Networks



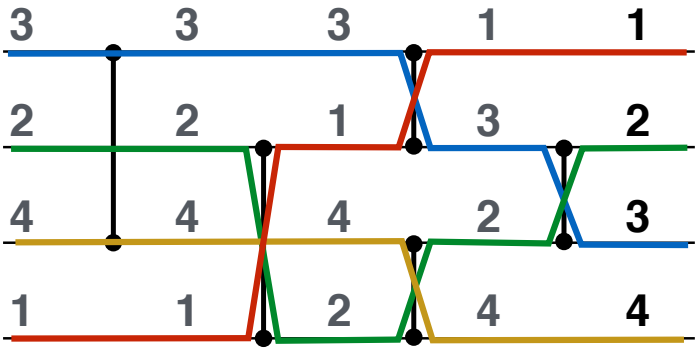
Sorting Networks



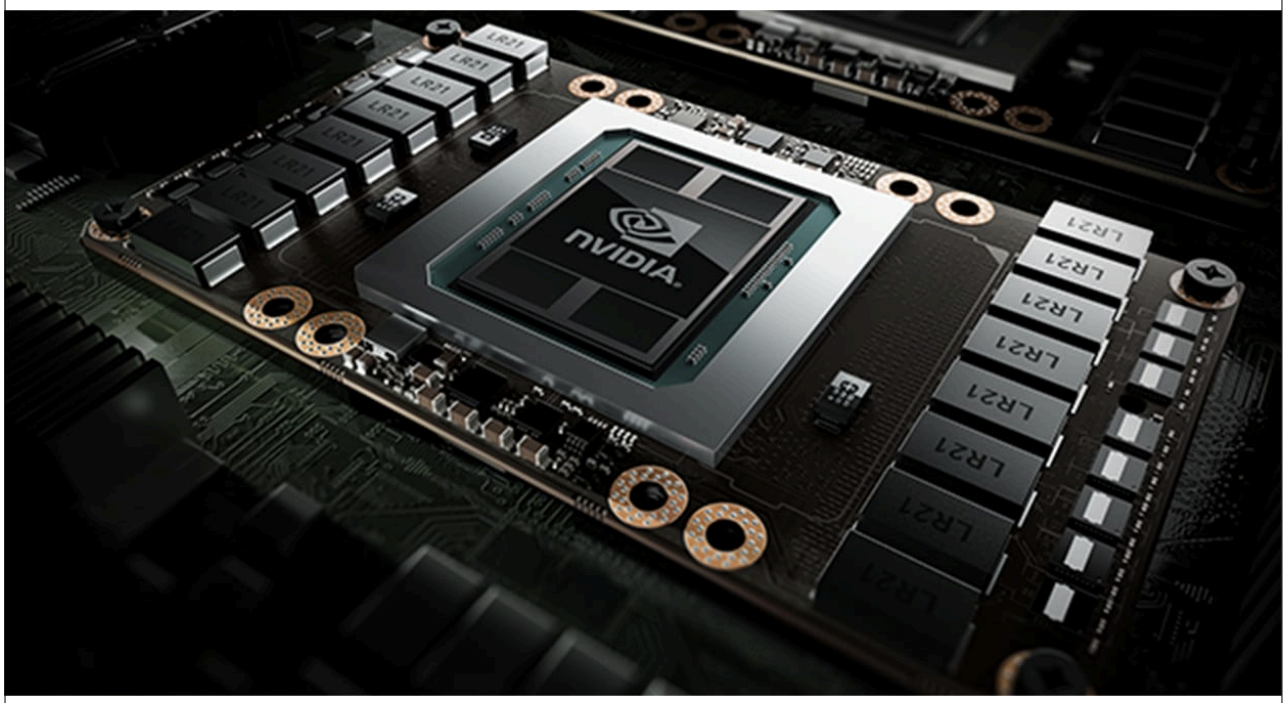
Sorting Networks



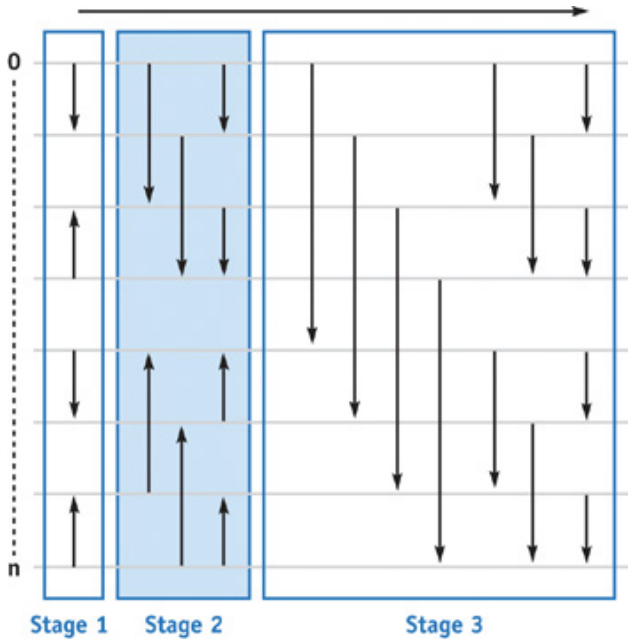
Sorting Networks



Sorting Network in Practice



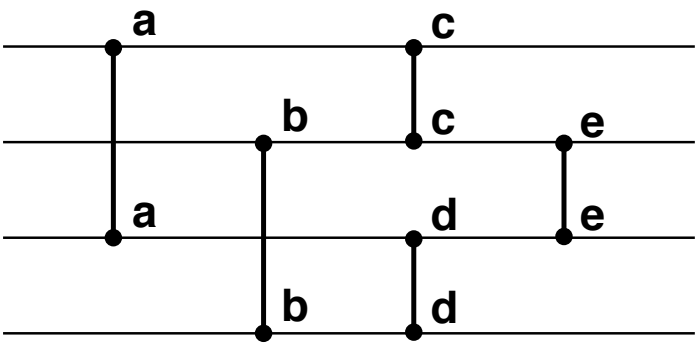
Bitonic Merge Sorting Network



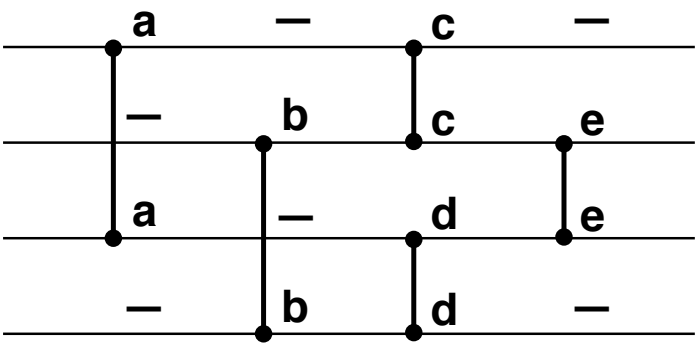
Problem:

Given a sorting network N and a list of numbers L, determine whether N sorts L correctly

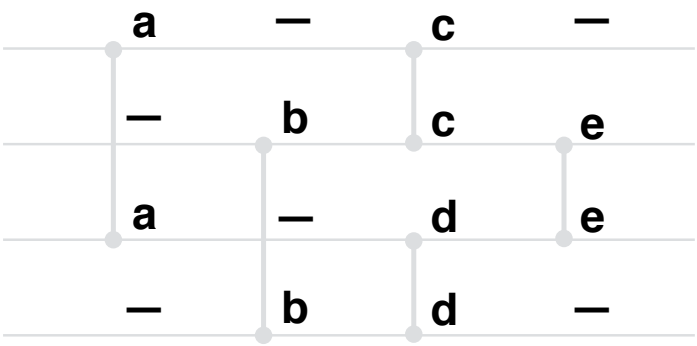
Encoding Input



Encoding Input



Encoding Input

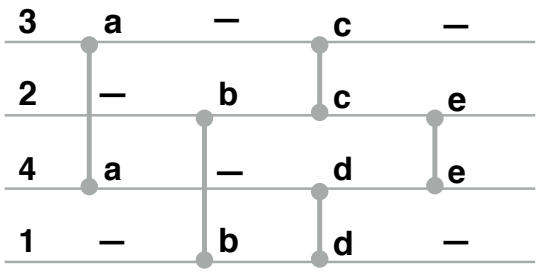


Encoding Input

num of rows

num of stages

```
4 4
a-c-
-bce
a-de
-bd-
3 2 4 1
```



Version 1

Monolithic Solution

…with a bad taste of naming

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;

    vector<string> ar(n);
    vector<int> inp(n);
    bool flag = false, err = false;

    for (int i = 0; i < n; ++i)
        cin >> ar[i];

    for (int i = 0; i < n; i++)
        cin >> inp[i];
```

```
    for (int j = 0; j < m; j++)
        for (int i = 0; i < n - 1; i++)
            if (ar[i][j] != '-') {
                if (ar[i][j] < 'a' || ar[i][j] > 'z') {
                    cout << "Invalid Network\n";
                    err = true;
                }
                if (!err)
                    for (int k = i + 1; k < n; k++) {
                        int cnt = 1;
                        if (ar[i][j] == ar[k][j]) {
                            cnt++;
                            if (inp[i] > inp[k]) {
                                char temp = inp[i];
                                inp[i] = inp[k];
                                inp[k] = temp;
                            }
                        }
                        if (cnt != 2) {
                            cout << "Invalid Network\n";
                            err = true;
                        }
                    }
            }
```

This program has a bug!

```
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

Problems with the Monolithic Solution

- The code is hardly readable
- Debugging is costly
- No part of the code can be reused
- Changes propagate through the whole code

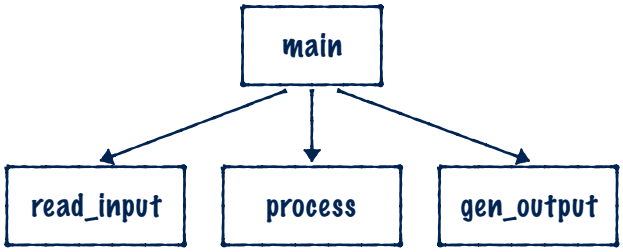
Version *2*

Breaking the Monolith

…a mediocre use of functions

Functions

- Decomposing problems into manageable pieces
- Program parts can be separately read/written
- Functions can be reused in a bottom-up manner
- Functional abstraction helps changing implementations




```
int main() {
    int n, m;
    cin >> n >> m;

    vector<string> ar(n);
    vector<int> inp(n);
    bool flag = false, err = false;

    for (int i = 0; i < n; ++i)
        cin >> ar[i];

    for (int i = 0; i < n; i++)
        cin >> inp[i];

    for (int j = 0; j < m; j++)
        for (int i = 0; i < n - 1; i++)
            if (ar[i][j] != '-') {
                if (ar[i][j] < 'a' || ar[i][j] > 'z') {
                    cout << "Invalid Network\n";
                    err = true;
                }
                if (!err)
                    for (int k = i + 1; k < n; k++) {
                        int cnt = 1;
                        if (ar[i][j] == ar[k][j]) {
                            cnt++;
                            if (inp[i] > inp[k]) {
                                char temp = inp[i];
                                inp[i] = inp[k];
                                inp[k] = temp;
                            }
                        }
                        if (cnt != 2) {
                            cout << "Invalid Network\n";
                            err = true;
                        }
                    }
            }
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

variable declarations

reading input

processing

generating output

```cpp
int main() {
    int n, m;
    cin >> n >> m;

    vector<string> ar(n);
    vector<int> inp(n);
    bool flag = false, err = false;

    for (int i = 0; i < n; ++i)
        cin >> ar[i];

    for (int i = 0; i < n; i++)
        cin >> inp[i];
```

```cpp
void read_input() {
    for (int i = 0; i < n; ++i)
        cin >> ar[i];

    for (int i = 0; i < n; i++)
        cin >> inp[i];
}

int main() {
    int n, m;
    cin >> n >> m;

    vector<string> ar(n);
    vector<int> inp(n);
    bool flag = false, err = false;

    read_input();
```

```cpp
int n, m;
vector<string> ar;
vector<int> inp;
bool flag = false, err = false;

void read_input() {
    cin >> n >> m;
    for (int i = 0; i < n; ++i) {
        string a;
        cin >> a;
        ar.push_back(a);
    }

    for (int i = 0; i < n; i++) {
        int b;
        cin >> b;
        inp.push_back(b);
    }
}
```

```
void process() {
    for (int j = 0; j < m; j++)
        for (int i = 0; i < n - 1; i++)
            if (ar[i][j] != '-') {
                if (ar[i][j] < 'a' || ar[i][j] > 'z') {
                    cout << "Invalid Network\n";
                    err = true;
                }
                if (!err)
                    for (int k = i + 1; k < n; k++) {
                        int cnt = 1;
                        if (ar[i][j] == ar[k][j]) {
                            cnt++;
                            if (inp[i] > inp[k]) {
                                char temp = inp[i];
                                inp[i] = inp[k];
                                inp[k] = temp;
                            }
                        }
                        if (cnt != 2) {
                            cout << "Invalid Network\n";
                            err = true;
                        }
                    }
            }
}
```

This program has a bug!

```
void generate_output() {
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

```cpp
int main() {
    int n, m;
    cin >> n >> m;

    vector<string> ar(n);
    vector<int> inp(n);
    bool flag = false, err = false;

    for (int i = 0; i < n; ++i)
        cin >> ar[i];

    for (int i = 0; i < n; i++)
        cin >> inp[i];

    for (int j = 0; j < m; j++)
        for (int i = 0; i < n - 1; i++)
            if (ar[i][j] != '-') {
                if (ar[i][j] < 'a' || ar[i][j] > 'z') {
                    cout << "Invalid Network\n";
                    err = true;
                }
                if (!err)
                    for (int k = i + 1; k < n; k++) {
                        int cnt = 1;
                        if (ar[i][j] == ar[k][j]) {
                            cnt++;
                            if (inp[i] > inp[k]) {
                                char temp = inp[i];
                                inp[i] = inp[k];
                                inp[k] = temp;
                            }
                        }
                        if (cnt != 2) {
                            cout << "Invalid Network\n";
                            err = true;
                        }
                    }
            }
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

variable declarations

reading input

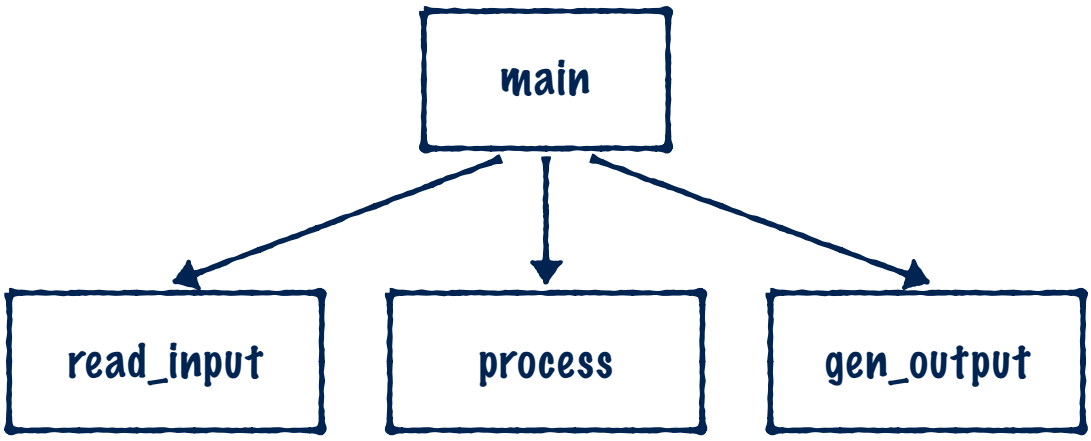
processing

generating output

```cpp
int main() {
    read_input();
    process();
    generate_output();
}
```

This program has a bug!

Program Structure



Are we happy now?

```
void process() {
    for (int j = 0; j < m; j++)
        for (int i = 0; i < n - 1; i++)
            if (ar[i][j] != '-') {
                if (ar[i][j] < 'a' || ar[i][j] > 'z') {
                    cout << "Invalid Network\n";
                    err = true;
                }
                if (!err)
                    for (int k = i + 1; k < n; k++) {
                        int cnt = 1;
                        if (ar[i][j] == ar[k][j]) {
                            cnt++;
                            if (inp[i] > inp[k]) {
                                char temp = inp[i];
                                inp[i] = inp[k];
                                inp[k] = temp;
                            }
                        }
                        if (cnt != 2) {
                            cout << "Invalid Network\n";
                            err = true;
                        }
                    }
            }
}
```

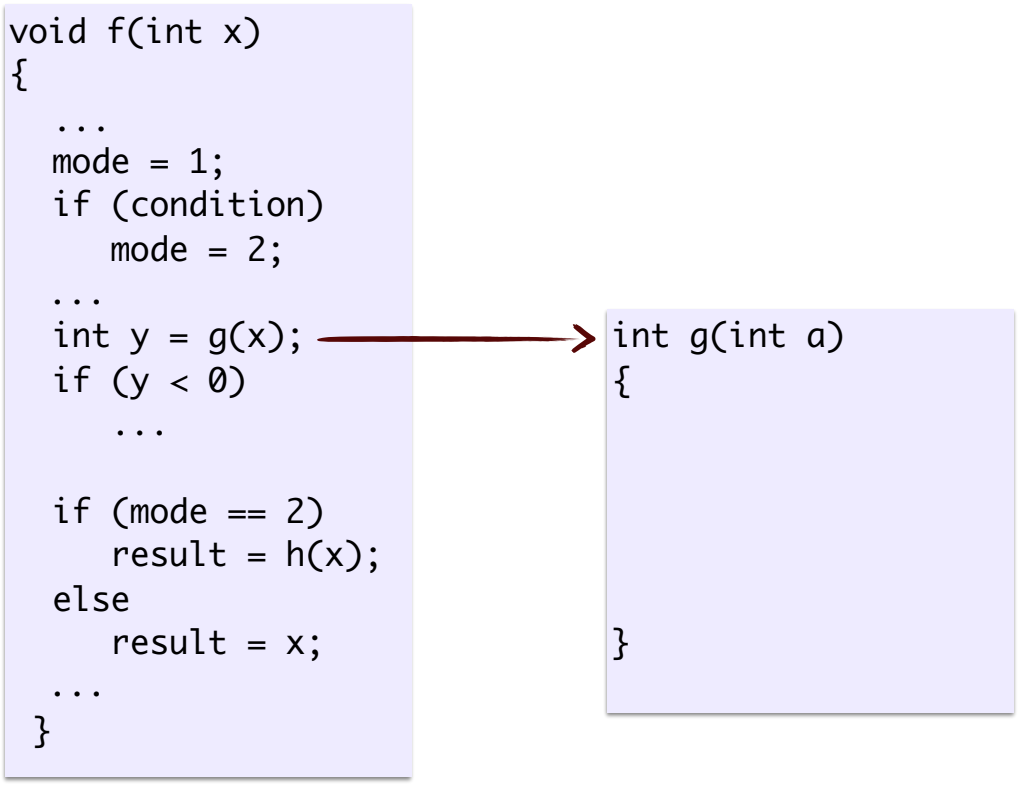
Is this more readable?!

This program has a bug!

Can we understand this function without looking at other parts of the program?

```
void generate_output() {
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

Global variables considered harmful!



```
void f(int x)
{
  ...
  mode = 1;
  if (condition)
     mode = 2;
  ...
  int y = g(x);
  if (y < 0)
     ...

  if (mode == 2)
     result = h(x);
  else
     result = x;
  ...
 }
```

- Implicit interfaces reduce readability

explicit effect

```
int g(int a)
{
  ...
  if (...)
   mode = 2;
  ...
  return something;
}
```

implicit effect

- Reusing g requires reusing mode too.
- Caller must correctly initialize/check mode.

```
int g(int a)
{
  ...
  if (...)
    mode = 2;
  ...
  return something;
}
```

```
int mode = 0;
```

Testing `generate_output` separately is hard

```
void generate_output() {
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

Getting rid of global variables...

```
void generate_output() {
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

Getting rid of global variables...

Declare inp as an argument

Check this before calling the function

```
void generate_output() {
    if (!err) {
        for (int i = 0; i < n - 1; i++)
            if (inp[i] > inp[i+1]) {
                cout << "Not sorted\n";
                flag = true;
            }
        if (!flag)
            cout << "Sorted\n";
    }
}
```

No global variables...

```
void generate_output(vector<int> numbers) {
    for (int i = 0; i < n - 1; i++)
        if (numbers[i] > numbers[i+1]) {
            cout << "Not sorted\n";
            flag = true;
        }
    if (!flag)
        cout << "Sorted\n";
}

int main() {
    read_input();
    process();
    if (!err)
        generate_output(inp);
}
```

Must bring the declaration inside

Better naming...

```
void check_sorted(vector<int> numbers) {
    bool not_sorted = false;
    for (int i = 0; i < n - 1; i++)
        if (numbers[i] > numbers[i+1]) {
            cout << "Not sorted\n";
            not_sorted = true;
        }
    if (!not_sorted)
        cout << "Sorted\n";
}

int main() {
    read_input();
    process();
    if (!invalid_network)
        check_sorted(inp);
}
```

```
void check_sorted(vector<int> numbers) {
    bool not_sorted = false;
    for (int i = 0; i < n - 1; i++)
        if (numbers[i] > numbers[i+1]) {
            cout << "Not sorted\n";
            not_sorted = true;
        }
    if (!not_sorted)
        cout << "Sorted\n";
}
```

check_sorted has two different responsibilities:

1. checking if the input vector is sorted
2. producing output

Having more than one responsibility reduces reusability

```
bool check_sorted(vector<int> numbers) {
    for (int i = 0; i < n - 1; i++)
        if (numbers[i] > numbers[i+1])
            return false;
    return true;
}

int main() {
    read_input();
    process();
    if (!invalid_network)
        if (check_sorted(inp))
            cout << "Sorted\n";
        else
            cout << "Not sorted\n";
}
```

using is_sorted function from C++ standard library (since C++11)

```
int main() {
    read_input();
    process();
    if (!invalid_network) {
        if (is_sorted(inp.begin(), inp.end()))
            cout << "Sorted\n";
        else
            cout << "Not sorted\n";
    }
}
```

Discussion

Why reusability is so important?

```
void process() {
    for (int j = 0; j < m; j++)
        for (int i = 0; i < n - 1; i++)
            if (ar[i][j] != '-') {
                if (ar[i][j] < 'a' || ar[i][j] > 'z') {
                    cout << "Invalid Network\n";
                    err = true;
                }
                if (!err)
                    for (int k = i + 1; k < n; k++) {
                        int cnt = 1;
                        if (ar[i][j] == ar[k][j]) {
                            cnt++;
                            if (inp[i] > inp[k]) {
                                char temp = inp[i];
                                inp[i] = inp[k];
                                inp[k] = temp;
                            }
                        }
                        if (cnt != 2) {
                            cout << "Invalid Network\n";
```

process has two different responsibilities:

1. checking the validity of the network
2. sorting the numbers (if valid)

This program has a bug!

Version 3

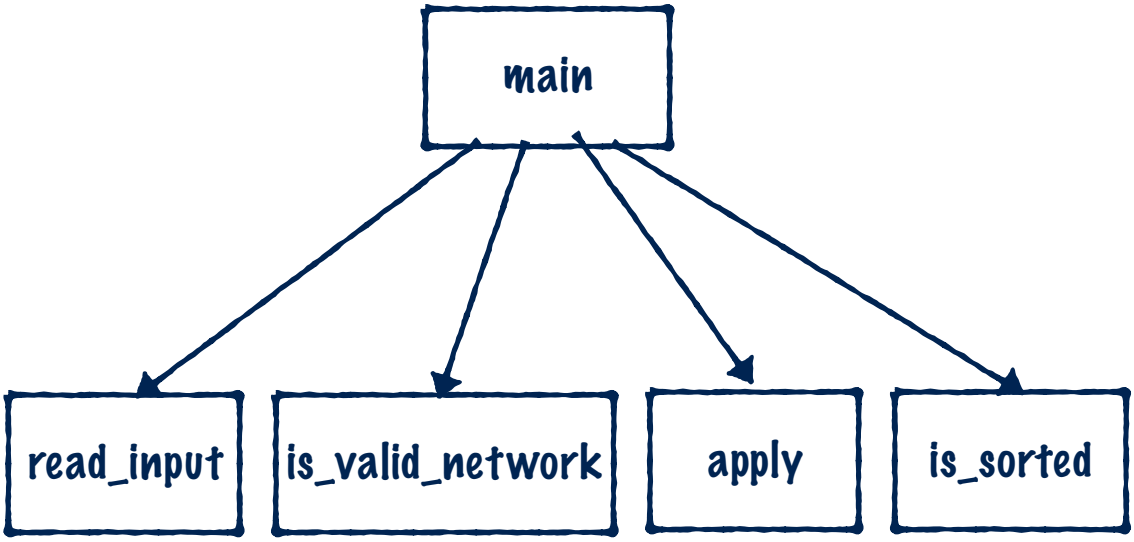
Functional Abstraction

Top-Down Design

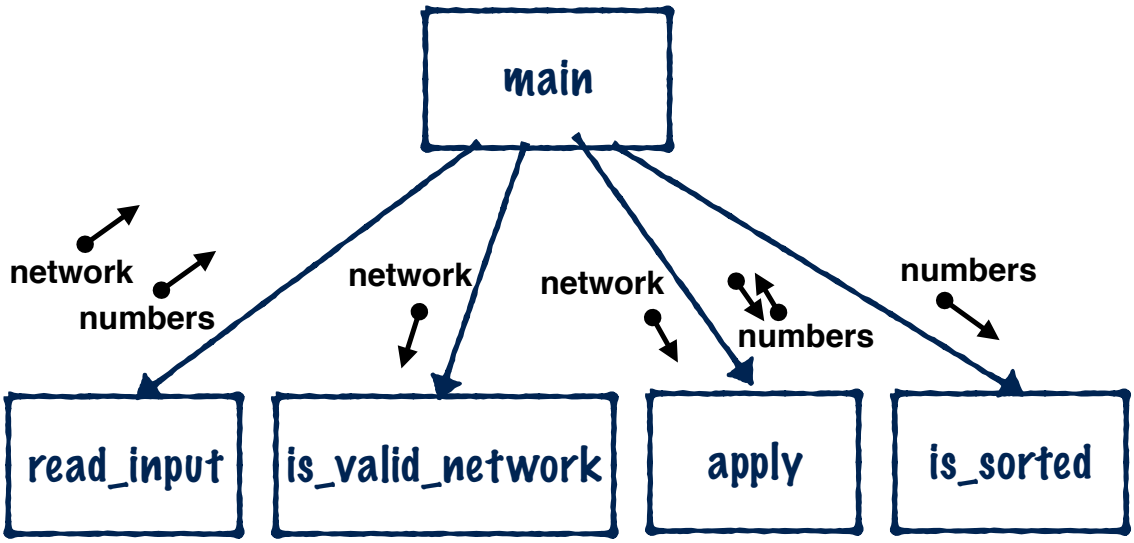
pseudo-code for the sorting network program:

```
read input
validate the network
if the network is invalid
    produce an error message and exit
else
    apply the network on the numbers
    check if the numbers are sorted and produce output
```

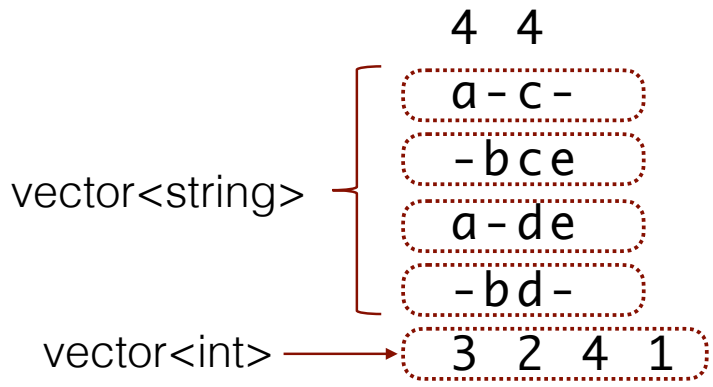
Program Structure



Data Flow



Choosing Data Structures



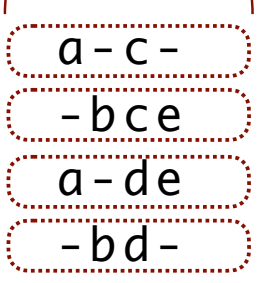

```
int main() {
    int num_of_inputs;
    int num_of_stages;

    cin >> num_of_inputs >> num_of_stages;

    vector<string> network = read_network(num_of_inputs);
    vector<int> numbers = read_numbers(num_of_inputs);

    if (!is_valid_network(network, num_of_stages)) {
        cout << "Invalid network\n";
        return -1;
    }

    apply_network(network, numbers);

    if (is_sorted(numbers.begin(), numbers.end()))
        cout << "Sorted";
    else
        cout << "Not sorted";
    cout << endl;
}
```

```
vector<string> read_network(int num_of_inputs) {
    vector<string> result;
    for (int i = 0; i < num_of_inputs; ++i) {
        string line;
        cin >> line;
        result.push_back(line);
    }
    return result;
}
```

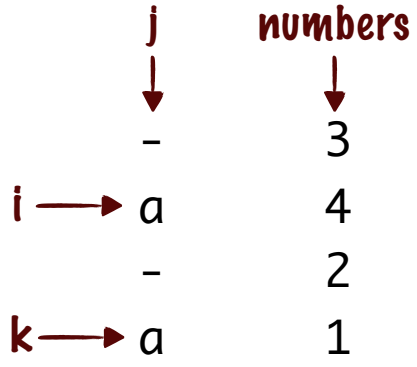
```
void apply_network(vector<string> network,
                   vector<int>& numbers)
{
    for (int j = 0; j < network[0].size(); j++)
        apply_stage(network, j, numbers);
}
```

network[0].size() = num_of_stages

network



```
void apply_stage(vector<string> network, int j,
                 vector<int>& numbers) {
    for (int i = 0; i < network.size() - 1; i++) {
        if (network[i][j] == '-')
            continue;

        for (int k = i + 1; k < network.size(); k++)
            if (network[i][j] == network[k][j])
                if (numbers[i] > numbers[k])
                    swap(numbers[i], numbers[k]);
    }
}
```

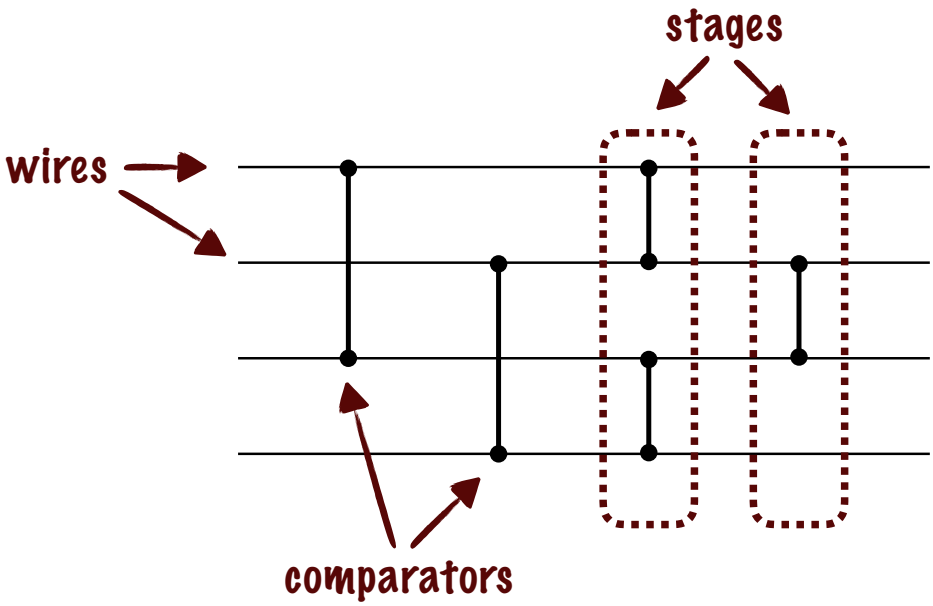



```
void apply_network(vector<string> network,
                   vector<int>& numbers)
{
    for (int j = 0; j < network[0].size(); j++)
        apply_stage(network, j, numbers);
}
```

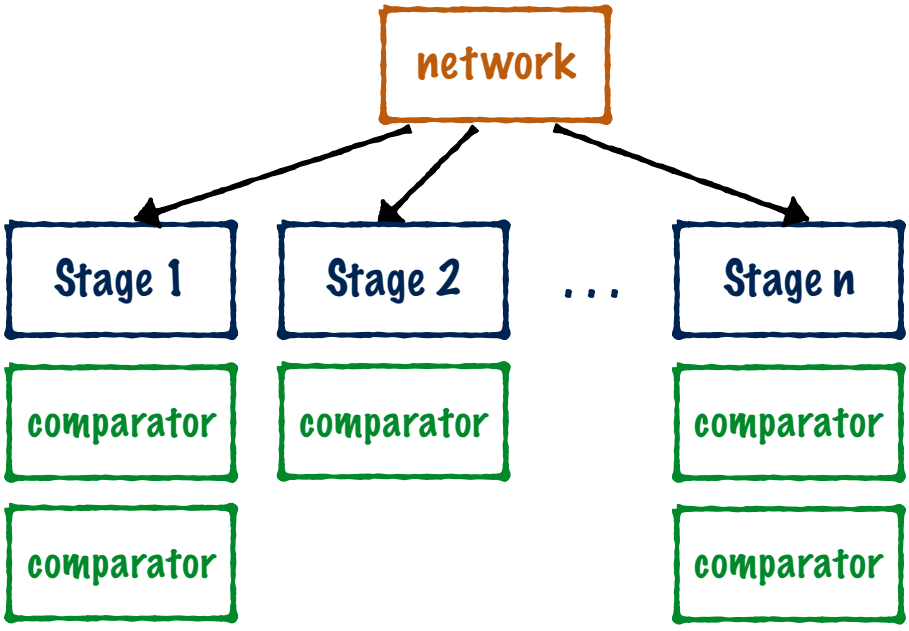
Question:

Can't we abstract data types like what we did with functions?

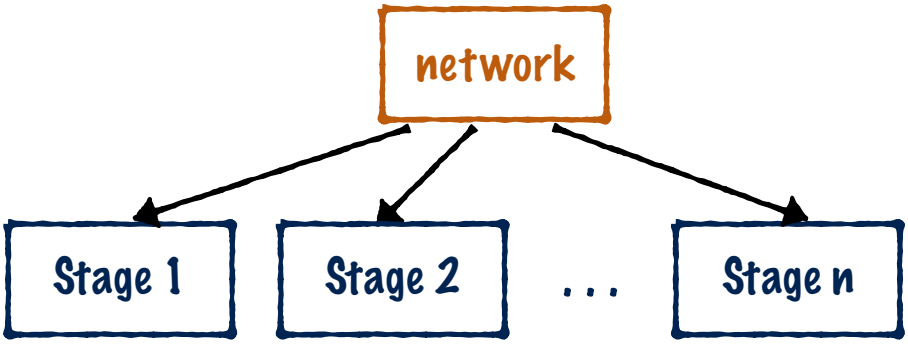
Top-Down Data Structure Design



Data Structure Design

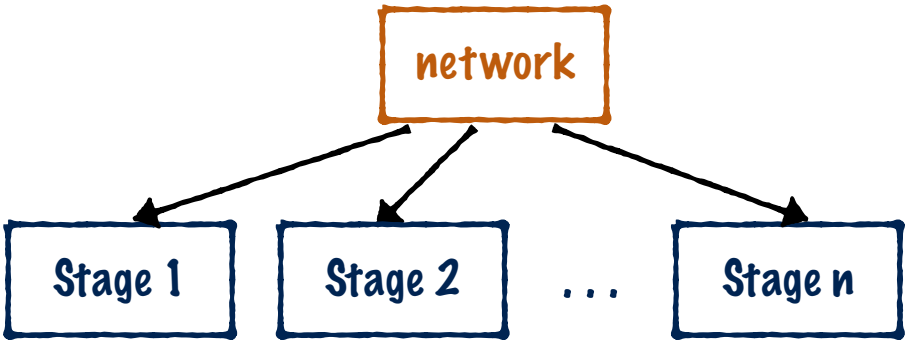


Modeling Data Types




```
typedef vector<Stage> Network;
Network net;
```

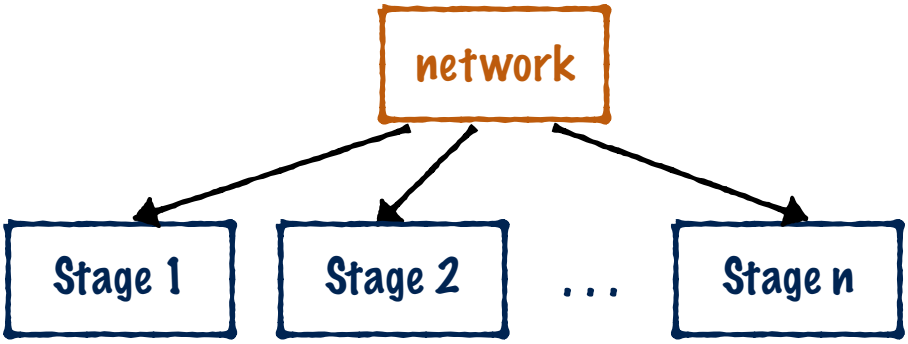
Modeling Data Types




```
typedef vector<Stage> Network;
Network net;
```

Independent of how Stage type is defined

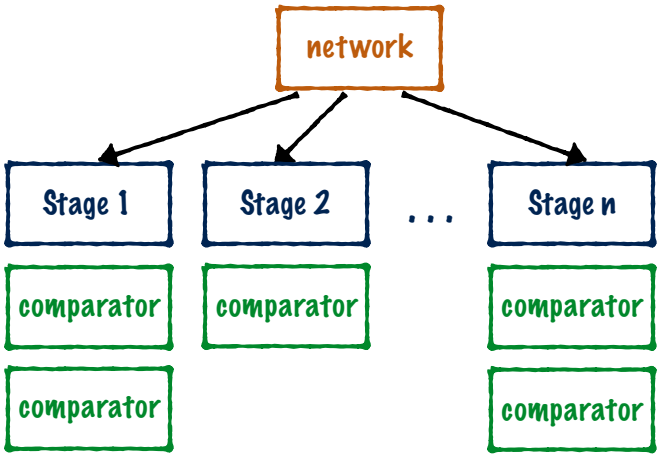
Modeling Data Types




```
typedef vector<Stage> Network;
typedef vector<char> Stage;
```

is it a good choice?

Modeling Data

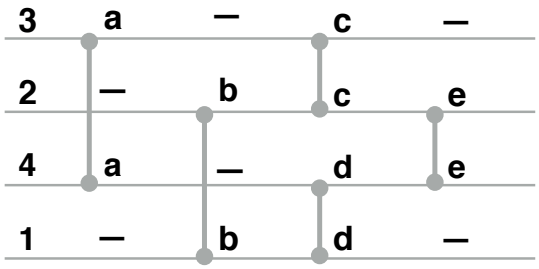



```
typedef vector<Stage> Network;
typedef vector<char> Stage;
```

where has this 'char' thing come from?

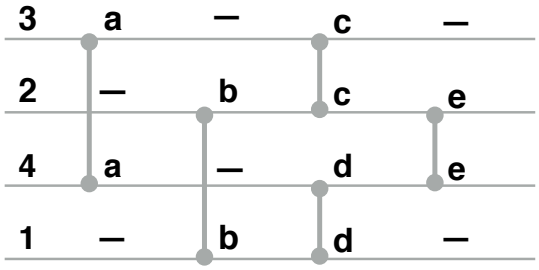
Recall…

```
4 4
a-c-
-bce
a-de
-bd-
3 2 4 1
```



Defining data models based on the input format is not a good decision.


```
4 4
a-c-
-bce
a-de
-bd-
3 2 4 1
```

On Data Formats…

XML

```
<student>
 <name>Mansoor Nasiri</name>
 <stdid>810198143</stdid>
 <GPA>12.80</GPA>
</student>
<student>
 <name>Naser Mansoori</name>
 <stdid>810198563</stdid>
 <GPA>16.34</GPA>
</student>
<student>
 <name>Nosrat Naseri</name>
 <stdid>810198117</stdid>
 <GPA>18.33</GPA>
</student>
<student>
 <name>Nasir Nosrati</name>
 <stdid>810198883</stdid>
 <GPA>11.56</GPA>
</student>
```

CSV

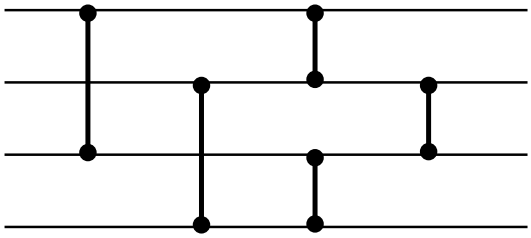
```
name,stdid,GPA
Mansoor Nasiri,810198143,12.80
Naser Mansoori,810198563,16.34
Nosrat Naseri,810198117,18.33
Nasir Nosrati,810198883,11.56
```

JSON

```
{
  "class": {
    "student": [
      {
        "name": "Mansoor Nasiri",
        "stdid": "810198143",
        "GPA": "12.80"
      },
      {
        "name": "Naser Mansoori",
        "stdid": "810198563",
        "GPA": "16.34"
      },
      {
        "name": "Nosrat Naseri",
        "stdid": "810198117",
        "GPA": "18.33"
      }
    ]
  }
}
```

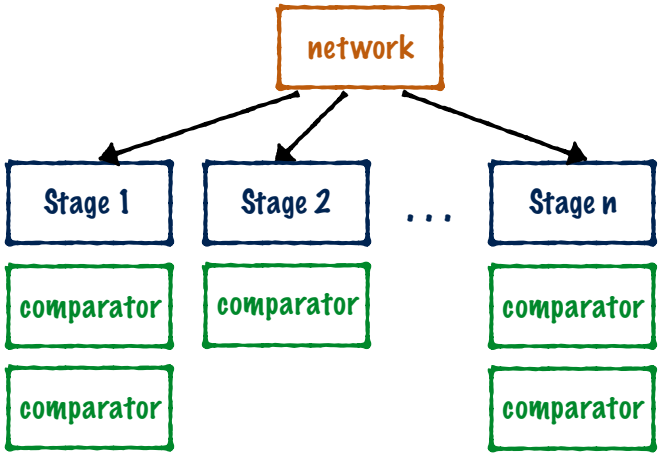
Network in XML Format

```
<sorting-net>
  <stage>
    <comp end1=1, end2=3 />
  </stage>
  <stage>
    <comp end1=2, end2=4 />
  </stage>
  <stage>
    <comp end1=1, end2=2 />
    <comp end1=3, end2=4 />
  </stage>
  <stage>
    <comp end1=2, end2=3 />
  </stage>
</sorting-net>
```



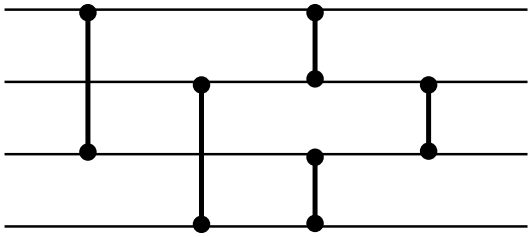
Modeling Data

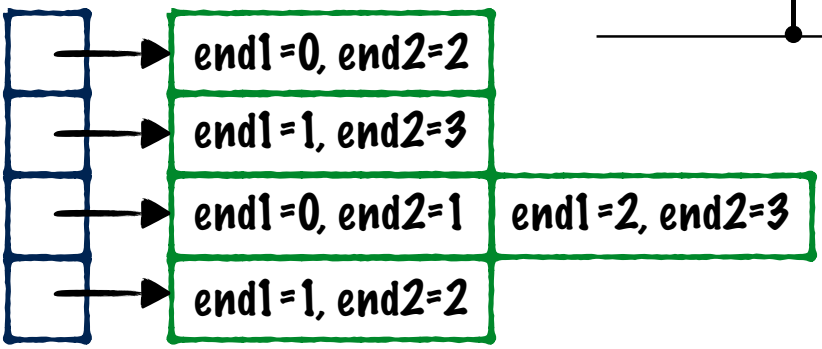
```
typedef vector<Stage> Network;

typedef vector<Comparator> Stage;

struct Comparator {
  int end1;
  int end2;
};
```



Modeling Data

```
typedef vector<stage> Network;

typedef vector<comparator> Stage;

struct Comparator {
  int end1;
  int end2;
};
```





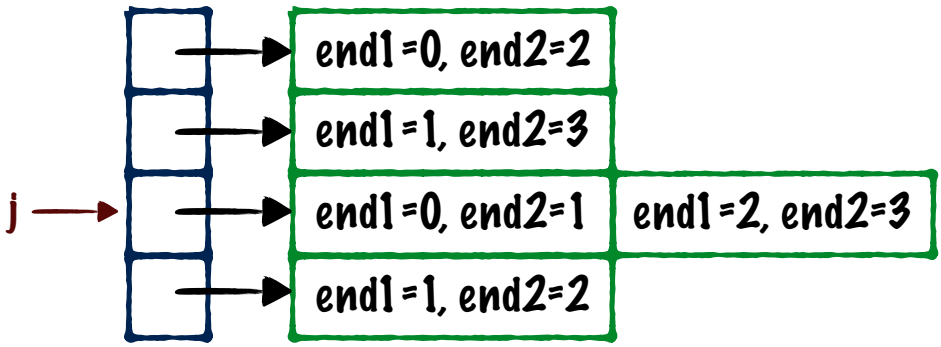
Version 4

Data Abstraction

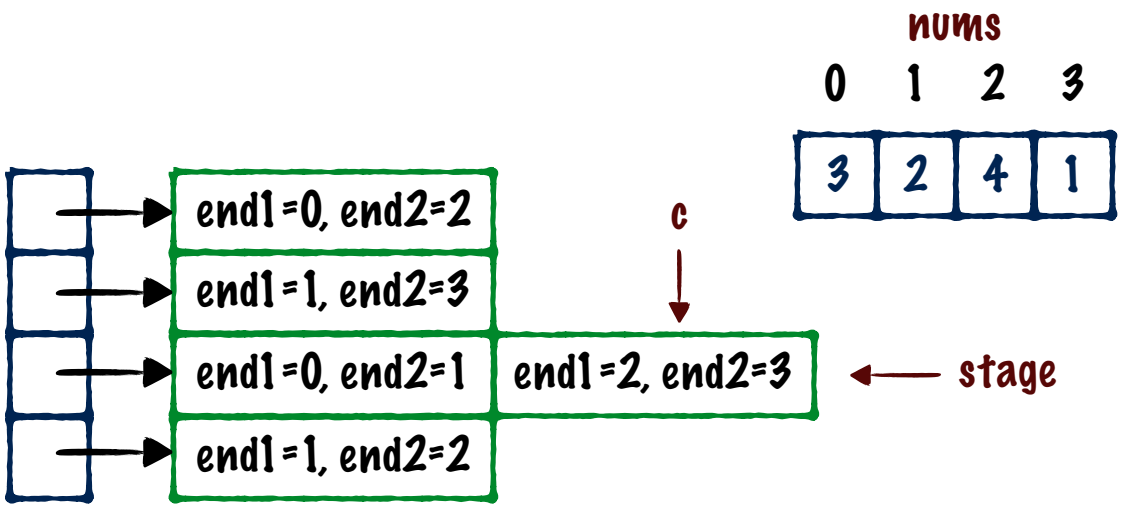
… a first step

```
int main()
{
    int num_of_inputs, num_of_stages;
    cin >> num_of_inputs >> num_of_stages;

    Network network = read_network(num_of_inputs,
                                   num_of_stages);
    Numbers numbers = read_input(num_of_inputs);

    if (!is_valid_network(network)) {
        cout << "Invalid network\n";
        return -1;
    }

    apply_network(network, numbers);

    if (is_sorted(numbers.begin(), numbers.end()))
        cout << "Sorted\n";
    else
        cout << "Not sorted\n";
}
```

```
void apply_network(Network network,
                   Numbers& numbers)
{
    for (int j = 0; j < network.size(); j++)
        apply_stage(network[j], numbers);
}
```



```
void apply_stage(Numbers& nums, Stage stage)
{
    for (int c = 0; c < stage.size(); c++)
        if (nums[stage[c].end1] > nums[stage[c].end2])
            swap(nums[stage[c].end1],
                 nums[stage[c].end2]);
}
```




```
Network read_network(int num_of_inputs, int num_of_stages) {

    Network net(num_of_stages);

    vector<string> temp_net;
    for (int i = 0; i < num_of_inputs; i++) {
        string line;
        cin >> line;
        temp_net.push_back(line);
    }

    for (int cur_stage = 0; cur_stage < num_of_stages; cur_stage++) {
        for (int i = 0; i < num_of_inputs; i++) {
            if (temp_net[i][cur_stage] == '-')
                continue;
            for (int k = i+1; k < num_of_inputs; k++)
                if (temp_net[i][cur_stage] == temp_net[k][cur_stage])
                    net[cur_stage].push_back(new_comparator(i, k));
        }
    }
    return net;
}
```

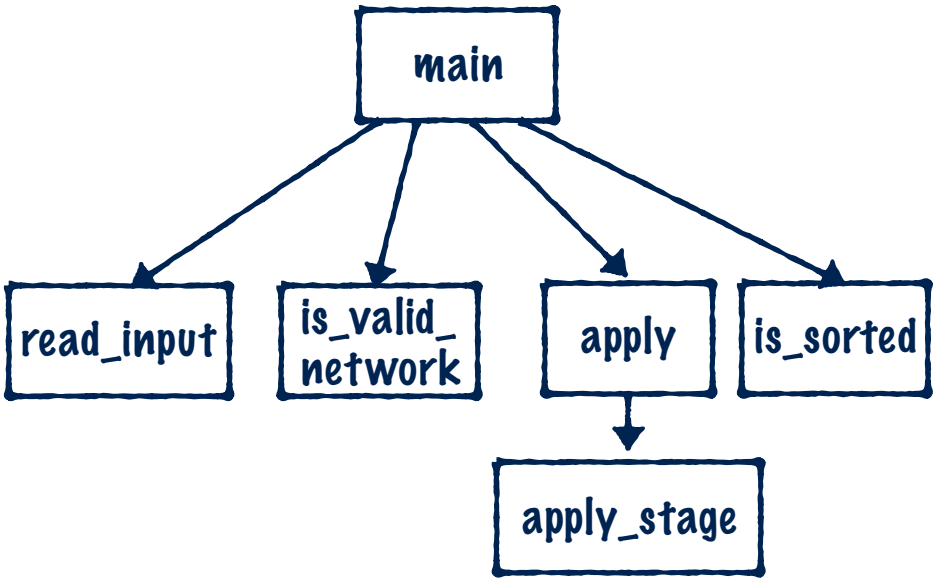
Note: the messiness is moved to read_network

Guideline:
Build your data model around computation, not input/output data formats.

Mistake:
Changing input/output data format to conform to internal data model. This reduces program usability.

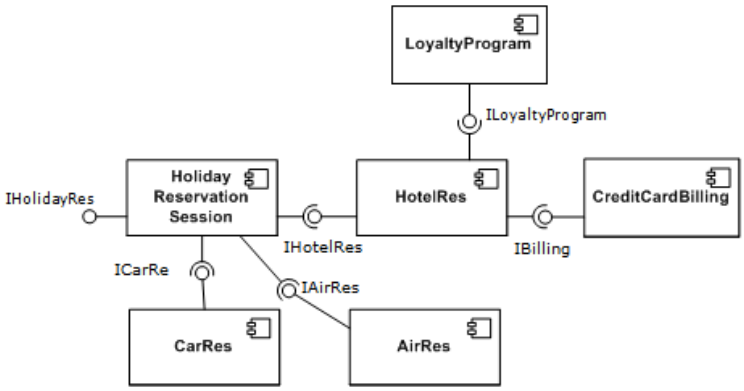
Points to Remember - 1

- Top-Down Design: use functions to decompose problem into manageable pieces



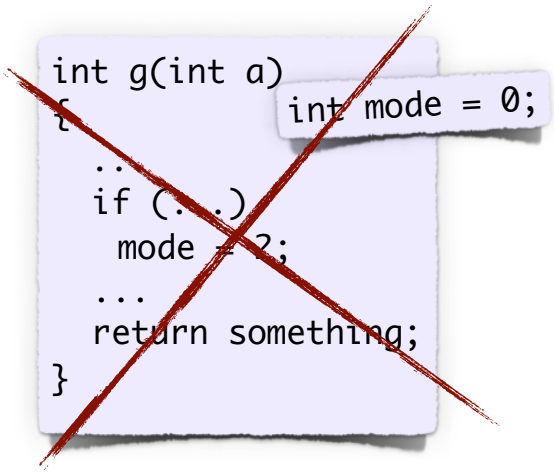
Points to Remember - 2

- Design functions for reusability (bottom-up composition)



Points to Remember - 3

- Avoid global variables



Points to Remember - 4

- Give each function a single responsibility, denoted by a meaningful name

Points to Remember - 5

- Use functions to abstract out implementation details

```
sort(v.begin(), v.end());
```

We don't know how sort works.
And, we are happy about this!