

Last updated April 22, 2016

# Summary for Week 0

- How the course works;
- How to solve simple (ad-hoc) programming challenges;
- Four simple programming challenges;

# Results for Week 0

Problem	Total	Submissions	Solutions
Division of Nlogonia	31	27	27
Cancer or Scorpio	31	25	19
$3n+1$ Problem	31	23	21
Request for Proposal	31	18	13

Congratulations to all who submitted!  
The problems for this week are already online.

# Outline for Week 1

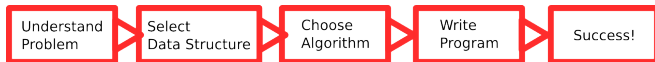
## Week Theme: Data Structures

This week, we will review and discuss the different data structures that are often used in competitive programming problems.

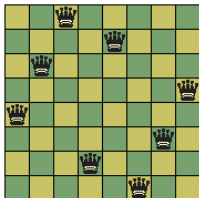
- Types and Variations of Data Structures (Linear 2D, 3D, Dynamic, Non-linear)
- Discussion on Implementation and notes;
- Discussion about next week exercises;
- Bitmasks (Monday?)

# Motivation: Why study data structures?

- The correct data structure makes the algorithm **faster**;  
*Example:* As we saw last week, the solution space of the 8-queen problem depends on the chosen structure.
- A good **implementation** of a data structure makes the implementation easier;  
*Exemple:* Using a linked list, you raise the possibility of null pointer errors.



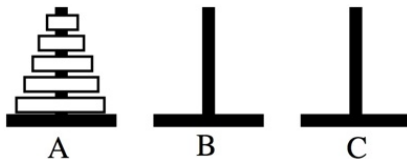
# Example 1: 8 Queen Problem (UVA 750)



Given the initial position of one queen, how many correct solutions exist?

- X,Y position representation =  $58^8$  total solutions
- Column position representation =  $8^8$  total solutions
- Permutation representation =  $8!$  total solutions

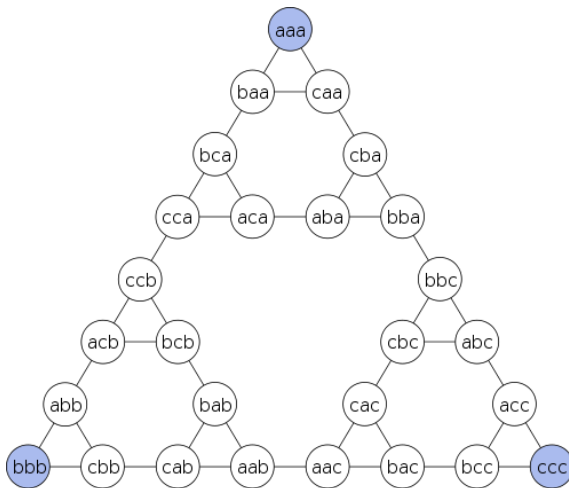
## Example 2: The Towers of Hanoi



- You have  $N$  disks and  $K$  poles. Each disk has unique size  $s_i$ .
- A disk  $i$  can be moved from one pole to another.
- A move of disk  $i$  to pole  $k$  is only valid if  $k$  has no disks smaller than  $i$ .
- Find the list of moves to move all disks from pole 1 to pole  $K$ .

How do you represent the data in this problem?

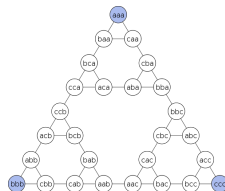
# Another way to visualize the Towers of Hanoi





# Explaining the Tower of Hanoi Data Structure

- Each node identifies an state in the problem;
- Each character in the string represents one disk and its position;
- We can have at most 3 state transitions at each state (can you prove it?)
- To solve the Towers of Hanoi problem, we find the path between the start and end states.
- (just beware of state explosion)



# What are data structures?

A data Structure (DS) is a mean of **storing** and **organizing** data.

Different Data structures have different strengths. You need to have a general idea of those, in order to **select** and **use** the right DS for the right problem.

# What do you need to know about DS?

## Characteristics of DS

- Time Complexity: Insertion, Removal, Initialization, Access;
- Space Complexity: Repetitions of data, Empty Space;
- Others: Restrictions, Special behaviors;

## Usage of DS

- Is the DS in the STL or Java API? How to call it?
- If not, how to implement the DS?
- How to perform basic operations on the DS?  
(Modification/Sort/Search)

As you learn new things, don't forget to expand your library

# CAVEAT: When theory is important

When solving programming challenges, we will normally focus more on the practical aspects of Data Structures (how fast it is? how to use it?). It is not important to us to understand WHY they work as they do.

However, for a well rounded scientist, it is important to know both the practical and theoretical aspects of many techniques.

# Linear Data Structures

One of the most common DS used in Programming Contests are.

## Linear Data Structures

The elements can be ordered in a linear sequence:  
left to right, top to down (Including multiple dimensions).

- **Static Array:**
  - The size is fixed, or estimated in advance.
  - Native data types. Make them bigger than necessary!
- **Dynamic Array:**
  - C++ STL vector / Java ArrayList;
  - Number of elements is truly unknown;
  - Learn how to use .iterators!

# Specialized Linear DSs

- **Array of Booleans (C++ bitset):**  
Useful operations such as reset, set (initialization) and test (bounded access)
- **Stack (C++ stack/Java stack):**  
 $O(1)$  push and pop. Used for recursion/Postfix/Bracket matching
- **Queue (C++ queue):**  
push (to back), pop (from front), used for Breadth First Search, Topological Sorting
- **Deque (C++ deque):**  
push and pop from both ends. Used for “sliding window” algorithms.

# Main operations in Arrays

When discussing linear DS, it is important to discuss two common actions done on them:

- **Sorting** – Changing the position of items in the array according to some ordering.
- **Searching** – Finding an item in an array with a certain value.

# Actions on Linear DS – Sorting

## Super formal Definition

Given unsorted stuff, sort them!

In programming contests, sorting is usually a preliminary step:

- sort a list to facilitate searching;
- sort a list to find highest values;
- sort a list for a greedy algorithm;
- sort a list to calculate cumulative values;
- etc...



# Sorting Example – Vito's Family (UVA 10041)

## Problem description

A gangster wants to move to a new house. The new house should have minimal distance from all his family, who live on the same street.

This program can be summarized as “find the median of the addresses and calculate the distance to all the houses.”

```
for (i = 0; i < m; i++)  
{  
    cin >> add[i];  
}  
sort(add, add+m);  
med = add[(m/2)];
```

# Sorting algorithms

You probably have heard of a large number of sorting algorithms:

- $O(n^2)$  algorithms: Bubble, Selection, Insertion sort;
- $O(n \log n)$  algorithms: Merge/Heap/Quicksort;
- $O(n)$  algorithms: Bucket/Radix sort;

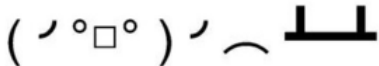
Can you remember the main characteristics of these algorithms?

# How many sorting Algorithms are there?

- bubblesort
- insertion sort
- selection sort
- heapsort
- mergesort
- quicksort
- radix sort
- bin sort
- gnome sort
- library sort
- comb sort
- tree transversal
- sorting networks
- cocktail shaking sort
- bucket sort
- bogo sort
- bitonick sort
- ...
- And many more!

# How many sorting Algorithms are there?

- bubblesort
- insertion sort
- selection sort
- heapsort
- mergesort
- quicksort
- radix sort
- bin sort
- gnome sort
- library sort
- comb sort
- tree transversal
- sorting networks
- cocktail shaking sort
- bucket sort
- bogo sort
- bitonick sort
- ...
- And many more!



# Implementing Sorting

You don't need to know every one of those methods. At least for programming contest, you need a good idea of how to use sorting in your language's API

## C++

Sorting is implemented by *STL algorithm*.

Eg: `sort(memory_start,memory_end,function)`

See also `partial_sort()` and `stable_sort()`

## Java

To sort with Java we use “`Collections.sort(array,function)`”

# Relax time

- 1 The song of the sorting people;
- 2 The quantum bogosort;

# Searching in an array

## Finding the highest value

- Unsorted array:  $O(n)$
- Sorted array:  $O(n \log n + 1)$

What if we need to repeat the operation  $k$  times?

## Finding a specific value

- Unsorted array:  $O(n)$
- Sorted array:  $O(n \log n + \log n)$  – binary search!

But can you implement a sorted array quickly?

# Binary Search in C++

You can use `algorithm::lower_bound` and `algorithm::upper_bound`

```
#include <iostream>
#include <algorithm>
#include <vector>
int main () {
    int myints[] = {10,20,30,30,20,10,10,20};
    std::vector<int> v(myints,myints+8);
    std::sort (v.begin(), v.end());

    std::vector<int>::iterator low,up;
    low=std::lower_bound (v.begin(), v.end(), 20);
    up= std::upper_bound (v.begin(), v.end(), 20);

    std::cout << "lower at " << (low- v.begin()) << '\n';
    std::cout << "upper at " << (up - v.begin()) << '\n';

    return 0; // up and low are memory indexes.
}
```



# Non Linear Data Structures

## Balanced Binary Search Tree

- C++ Map/Set (implemented as RB tree)
- Java TreeMap/TreeSet

Access/Deletion/Search in  $O(\log n)$ .

## Heap

- C++ priority\_queue
- Java PriorityQueue

Gets the highest value individual in  $O(1)$

# Tree Linear Structures: Example Problem

## UVA 10226 – Hardwood Species

Input is up to 1.000.000 trees, up to 10.000 types. Sort and output the speciation. Test cases are not limited.

Because of the huge numbers, storing the trees in an array will not be enough. We need a fast data structure to solve this problem in time.

Otherwise the problem is not actually difficult.

## Conclusion and problems

- Seven problems are listed, all of them related to DS;
- However, you don't need to solve ALL of them (unless you want to!)
- Try to discover which problems are easier, and solve those first.

# Know your data structures!

To be able to do data structure tricks, we need to be familiar with a variety of data structures.

- What is the time and memory efficiency of each data structure?
- What is **programming efficiency** of each data structure?
- What are the common uses?
- What are the common bugs?

# Data Structure Libraries

The “Library” word can have many meanings:

## Language Library

- What function is used to create a dictionary?
- What parameters are passed in this function?

## Personal Library

- How many data structures do you personally remember?
- Notes on paper can be very useful!

# Low Level Data Structures

- Array
- Linked List

The Array is usually simpler, and less bug prone. But be careful with index overflows!

When in doubt, use a bigger array!

# Medium Level Data Structures

- Stack
- Queue

The stack is the simplest “complex” data structure. It is implemented with an array and an index.

How do you implement a Queue using two stacks?

Queue and Stack are used very often.

# High level data structures

- Sets
- Dictionaries
- Priority Queues

These structures attach extra information to data: Key, Uniqueness, order.

Try to think of them as combinations of the above techniques. How would you implement them?



# Other data structures

- Trees
- Graphs

We will talk about these in future classes

# String Representation in Computers

When you type “/s”, why do numbers appear before letters?

Strings in a computer system are represented as an **Array of characters**, and each character is represented as an index.

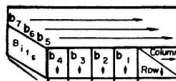
# Characters as numerical indexes: what are the consequences?

- Operation on characters (Addition, comparison);
- Arbitrary order of glyphs;

# Encodings

Encodings are mappings between a set of indexes and a set of glyphs. Different encodings cause Mojibake.

USASCII code chart



								0 0	0 0	0 1	0 1	1 0	1 0	1 1	1 1
								0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	NUL	DLE	SP	0	@	P	\	p
0	0	0	1	1	1	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	0	0	0	0	2	STX	DC2	"	2	B	R	b
0	0	1	1	1	1	1	1	3	ETX	DC3	#	3	C	S	c
0	1	0	0	0	0	0	0	4	EOT	DC4	\$	4	D	T	t
0	1	0	1	1	1	1	1	5	ENQ	NAK	%	5	E	U	u
0	1	1	0	0	0	0	0	6	ACK	SYN	&	6	F	V	v
0	1	1	1	1	1	1	1	7	BEL	ETB	'	7	G	W	w
1	0	0	0	0	0	0	0	8	BS	CAN	(	8	H	X	x
1	0	0	1	1	1	1	1	9	HT	EM	)	9	I	Y	y
1	0	1	0	0	0	0	0	10	LF	SUB	*	:	J	Z	z
1	0	1	1	1	1	1	1	11	VT	ESC	+	;	K	[	{
1	1	0	0	0	0	0	0	12	FF	FS	,	<	L	\	
1	1	0	1	1	1	1	1	13	CR	GS	-	=	M	]	}
1	1	1	0	0	0	0	0	14	SO	RS	.	>	N	^	~
1	1	1	1	1	1	1	1	15	SI	US	/	?	O	_	DEL

ASCII Indexes were selected with very specific properties.