Introductory Lecture
Topics: basics, resources, stl, bitwise tricks

League of Programmers

ACA, IIT Kanpur

October 21, 2012

Outline

Aim

- Discussion camp not a lecture series. You need to show motivation.

Aim

- Discussion camp not a lecture series. You need to show motivation.
- To help you introduce to the world of algorithms and competitive programming

Aim

- Discussion camp not a lecture series. You need to show motivation.
- To help you introduce to the world of algorithms and competitive programming
- Many useful algorithms, mathematical insights

Aim

- Discussion camp not a lecture series. You need to show motivation.
- To help you introduce to the world of algorithms and competitive programming
- Many useful algorithms, mathematical insights
- Useful for any programming contest that you may encounter

Aim

- Discussion camp not a lecture series. You need to show motivation.
- To help you introduce to the world of algorithms and competitive programming
- Many useful algorithms, mathematical insights
- Useful for any programming contest that you may encounter
- After this you can rock in job/internship interviews

Aim

- Discussion camp not a lecture series. You need to show motivation.
- To help you introduce to the world of algorithms and competitive programming
- Many useful algorithms, mathematical insights
- Useful for any programming contest that you may encounter
- After this you can rock in job/internship interviews
- It's fun too!

Aim

- Discussion camp not a lecture series. You need to show motivation.
- To help you introduce to the world of algorithms and competitive programming
- Many useful algorithms, mathematical insights
- Useful for any programming contest that you may encounter
- After this you can rock in job/internship interviews
- It's fun too!
- There are handsome rewards - prestige, joy of learning new things, and yes lots of money!

Language specifications

- Language we will stress upon: C, C++, Java
  ACM-ICPC Official languages. Allowed in almost every contests

Language specifications

- Language we will stress upon: C, C++, Java
  ACM-ICPC Official languages. Allowed in almost every contests
- But, Java is comparatively very slow, so sometimes an optimal algorithm might time out on the judge

Language specifications

- Language we will stress upon: C, C++, Java
  ACM-ICPC Official languages. Allowed in almost every contests
- But, Java is comparatively very slow, so sometimes an optimal algorithm might time out on the judge
- C has too restrictive and does not support stl/templates /classes

Language specifications

- Language we will stress upon: C, C++, Java
  ACM-ICPC Official languages. Allowed in almost every contests
- But, Java is comparatively very slow, so sometimes an optimal algorithm might time out on the judge
- C has too restrictive and does not support stl/templates /classes
- Use Library functions and Data Structures instead of writing your own every time

Programming competitions

- ACM-ICPC

Programming competitions

- ACM-ICPC
- Google Code Jam - Google's annual programming contest

Programming competitions

- ACM-ICPC
- Google Code Jam - Google's annual programming contest
- Facebook Hacker Cup - an easy gateway to facebook job

Programming competitions

- ACM-ICPC
- Google Code Jam - Google's annual programming contest
- Facebook Hacker Cup - an easy gateway to facebook job
- IOPC (IITK), Shaastra (IITM), Bitwise (IITKgp)

Websites for practice

- Compete against Indian coders in live contests: Codechef
- Short Programming Contests: Codeforces, Topcoder
- Problem set Archives: SPOJ, Project Euler, livearchive, acm.sgu.ru and many more

Outline

Overflow

```
#include<stdio.h>
int main()
{
int a, b;
scanf("%d %d", &a, &b);
printf("%d\n", a+b);
return 0;
}
```

Overflow

```
#include<stdio.h>
int main()
{
int a, b;
scanf("%d %d", &a, &b);
printf("%d\n", a+b);
return 0;
}
```

- What if the given numbers are HUGE?

Overflow

```
#include<stdio.h>
int main()
{
int a, b;
scanf("%d %d", &a, &b);
printf("%d\n", a+b);
return 0;
}
```

- What if the given numbers are HUGE?
- Not all the input constraints are explicit

Overflow

```
#include<stdio.h>
int main()
{
int a, b;
scanf("%d %d", &a, &b);
printf("%d\n", a+b);
return 0;
}
```

- What if the given numbers are HUGE?
- Not all the input constraints are explicit
- Always think about the worst case scenario, edge cases, etc.

Others

- Comparing doubles
  Always keep a cushion of ϵ

Others

- Comparing doubles
  Always keep a cushion of ϵ

```
double a, b;
a==b, not a very good idea
```

Others

- Comparing doubles
  Always keep a cushion of ϵ

  ```
  double a, b;
  a==b, not a very good idea
  ```

  instead, do the following

  ```
  #define EPS 0.0000001
  (a-EPS<b and a+EPS>b)
  ```

Others

- Comparing doubles
  Always keep a cushion of ϵ

  ```
  double a, b;
  a==b, not a very good idea
  ```

  instead, do the following

  ```
  #define EPS 0.0000001
  (a-EPS<b and a+EPS>b)
  ```
- Segmentation fault

Others

- Comparing doubles
  Always keep a cushion of ϵ

  ```
  double a, b;
  a==b, not a very good idea
  ```

  instead, do the following

  ```
  #define EPS 0.0000001
  (a-EPS<b and a+EPS>b)
  ```

- Segmentation fault
  - Invalid memory reference
  - Using too much memory than provided

Outline

Problem Solving Methodology

- Understand what the program is expected to do.

Problem Solving Methodology

- Understand what the program is expected to do.
- Understand the Input/Output format and use exactly that format

Problem Solving Methodology

- Understand what the program is expected to do.
- Understand the Input/Output format and use exactly that format
- Meaning of constraints

Problem Solving Methodology

- Understand what the program is expected to do.
- Understand the Input/Output format and use exactly that format
- Meaning of constraints
- What do time limit and memory limit mean??

Problem Solving Methodology

- Understand what the program is expected to do.
- Understand the Input/Output format and use exactly that format
- Meaning of constraints
- What do time limit and memory limit mean??
- Predict the order requirements of the given problem

Problem Solving Methodology

- Understand what the program is expected to do.
- Understand the Input/Output format and use exactly that format
- Meaning of constraints
- What do time limit and memory limit mean??
- Predict the order requirements of the given problem
- $1sec \approx (1-2) * 10^8$ operations

Problem Solving Methodology

- Understand what the program is expected to do.
- Understand the Input/Output format and use exactly that format
- Meaning of constraints
- What do time limit and memory limit mean??
- Predict the order requirements of the given problem
- $1sec \approx (1-2) * 10^8$ operations
- $xMB \approx x/4 * 10^6$ sized int arrays

Problem Solving Methodology

- Understand what the program is expected to do.
- Understand the Input/Output format and use exactly that format
- Meaning of constraints
- What do time limit and memory limit mean??
- Predict the order requirements of the given problem
- $1sec \approx (1-2) * 10^8$ operations
- $xMB \approx x/4 * 10^6$ sized int arrays
- For example, 10 test cases with $N = 10000$ means $O(N^2)$ is required

Problem Solving Methodology

1. Understanding the problem, mathematical formulation of the problem

Problem Solving Methodology

1. Understanding the problem, mathematical formulation of the problem
2. Categorizing the problem into one of the many types known

Problem Solving Methodology

1. Understanding the problem, mathematical formulation of the problem
2. Categorizing the problem into one of the many types known
3. Formulate a solution

Problem Solving Methodology

1. Understanding the problem, mathematical formulation of the problem
2. Categorizing the problem into one of the many types known
3. Formulate a solution
4. Verify with the sample test cases, make sure your solution atleast works for them

Problem Solving Methodology

1. Understanding the problem, mathematical formulation of the problem
2. Categorizing the problem into one of the many types known
3. Formulate a solution
4. Verify with the sample test cases, make sure your solution atleast works for them
5. Generate some small test cases of your own, the sample test cases may not be include some boundary cases.

Problem Solving Methodology

1. Understanding the problem, mathematical formulation of the problem
2. Categorizing the problem into one of the many types known
3. Formulate a solution
4. Verify with the sample test cases, make sure your solution atleast works for them
5. Generate some small test cases of your own, the sample test cases may not be include some boundary cases.
6. Coding the solution (the easiest part of all)

Problem Solving Methodology

1. Understanding the problem, mathematical formulation of the problem
2. Categorizing the problem into one of the many types known
3. Formulate a solution
4. Verify with the sample test cases, make sure your solution atleast works for them
5. Generate some small test cases of your own, the sample test cases may not be include some boundary cases.
6. Coding the solution (the easiest part of all)
7. Debugging (TLE: time limit exceeded, WA : incorrect solution etc.)

Some optimizations

- Not all operations are equally fast:
  operations on unsigned ints/long long are faster
  bitwise operators and shift operators (&ˆ $|\gg\ll$)
  Using too much memory ($> 10MB$) slows down programmes

Some optimizations

- Not all operations are equally fast:
  operations on unsigned ints/long long are faster
  bitwise operators and shift operators (&ˆ $|\gg\ll$)
  Using too much memory ($> 10MB$) slows down programmes
- Look at other people's code for reference and optimisations

Some optimizations

- Not all operations are equally fast:
  operations on unsigned ints/long long are faster
  bitwise operators and shift operators (&ˆ $|\gg\ll$)
  Using too much memory ($> 10MB$) slows down programmes
- Look at other people's code for reference and optimisations
- Read this: `http://www.codeproject.com/KB/cpp/C___Code_Optimization.aspx`

Some optimizations

- Not all operations are equally fast:
  operations on unsigned ints/long long are faster
  bitwise operators and shift operators (&ˆ $|\gg\ll$)
  Using too much memory ($> 10MB$) slows down programmes
- Look at other people's code for reference and optimisations
- Read this: `http://www.codeproject.com/KB/cpp/C___Code_Optimization.aspx`
- I/O: do NOT use cin/cout for large input output

Some Standard paradigms

- Sorting
- Searching
- Preprocessing
- Divide-and-Conquer
- Dynamic Programming
- Greedy Algorithms
- Graph
- Network Flow
- Backtracking
- Computational Geometry
- Pure maths
- Ad-hoc problems

Outline

Standard template library

Website: `http://www.cplusplus.com/reference`

1. Data Structures

Standard template library

Website: `http://www.cplusplus.com/reference`

1. Data Structures
   - vector

Standard template library

Website: `http://www.cplusplus.com/reference`

1. Data Structures
   - vector
   - stack

Standard template library

Website: `http://www.cplusplus.com/reference`

1. Data Structures
   - vector
   - stack
   - queue

Standard template library

Website: `http://www.cplusplus.com/reference`

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue

Standard template library

Website: http://www.cplusplus.com/reference

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set

Standard template library

Website: `http://www.cplusplus.com/reference`

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set
   - map

Standard template library

Website: http://www.cplusplus.com/reference

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set
   - map
2. Algorithms

Standard template library

Website: `http://www.cplusplus.com/reference`

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set
   - map
2. Algorithms
   - find

Standard template library

Website: http://www.cplusplus.com/reference

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set
   - map
2. Algorithms
   - find
   - max, min

Standard template library

Website: http://www.cplusplus.com/reference

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set
   - map
2. Algorithms
   - find
   - max, min
   - sort

Standard template library

Website: http://www.cplusplus.com/reference

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set
   - map
2. Algorithms
   - find
   - max, min
   - sort
   - reverse

Standard template library

Website: http://www.cplusplus.com/reference

1. Data Structures
   - vector
   - stack
   - queue
   - priority_queue
   - set
   - map
2. Algorithms
   - find
   - max, min
   - sort
   - reverse
   - swap

Stack

1. Last in, first out (LIFO)

Stack

1. Last in, first out (LIFO)
2. Supports three constant-time operations

Stack

1. Last in, first out (LIFO)
2. Supports three constant-time operations
   - Push(x): inserts x into the stack

Stack

1. Last in, first out (LIFO)
2. Supports three constant-time operations
   - Push(x): inserts x into the stack
   - Pop(): removes the newest item

Stack

1. Last in, first out (LIFO)
2. Supports three constant-time operations
   - Push(x): inserts x into the stack
   - Pop(): removes the newest item
   - Top(): returns the newest item

Stack

1. Last in, first out (LIFO)
2. Supports three constant-time operations
   - Push(x): inserts x into the stack
   - Pop(): removes the newest item
   - Top(): returns the newest item
3. C++ and Java have implementations of stack

Queue

1. First in, first out (FIFO)

Queue

1. First in, first out (FIFO)
2. Supports three constant-time operations

Queue

1. First in, first out (FIFO)
2. Supports three constant-time operations
   - Enqueue(x): inserts x into the queue

Queue

1. First in, first out (FIFO)
2. Supports three constant-time operations
   - Enqueue(x): inserts x into the queue
   - Dequeue(): removes the oldest item

Queue

1. First in, first out (FIFO)
2. Supports three constant-time operations
   - Enqueue(x): inserts x into the queue
   - Dequeue(): removes the oldest item
   - Front(): returns the oldest item

Queue

1. First in, first out (FIFO)
2. Supports three constant-time operations
   - Enqueue(x): inserts x into the queue
   - Dequeue(): removes the oldest item
   - Front(): returns the oldest item
3. C++ and Java have implementations of queue

Priority Queue

1. Each element in a PQ has a priority value

Priority Queue

1. Each element in a PQ has a priority value
2. Three operations:

Priority Queue

1. Each element in a PQ has a priority value
2. Three operations:
   - Insert(x, p): inserts x into the PQ, whose priority is p

Priority Queue

1. Each element in a PQ has a priority value
2. Three operations:
   - Insert(x, p): inserts x into the PQ, whose priority is p
   - RemoveTop(): removes the element with the highest priority

Priority Queue

1. Each element in a PQ has a priority value
2. Three operations:
   - Insert(x, p): inserts x into the PQ, whose priority is p
   - RemoveTop(): removes the element with the highest priority
   - Top(): returns the element with the highest priority

Priority Queue

1. Each element in a PQ has a priority value
2. Three operations:
   - Insert(x, p): inserts x into the PQ, whose priority is p
   - RemoveTop(): removes the element with the highest priority
   - Top(): returns the element with the highest priority
3. All operations can be done quickly if implemented using a heap

Priority Queue

1. Each element in a PQ has a priority value
2. Three operations:
   - Insert(x, p): inserts x into the PQ, whose priority is p
   - RemoveTop(): removes the element with the highest priority
   - Top(): returns the element with the highest priority
3. All operations can be done quickly if implemented using a heap
4. C++ and Java have implementations of priority queue

Heap

1. Complete binary tree with the heap property: value of a node $\geq$ values of its children

Heap

1. Complete binary tree with the heap property: value of a node $\geq$ values of its children
2. The root node has the maximum value

Heap

1. Complete binary tree with the heap property: value of a node $\geq$ values of its children
2. The root node has the maximum value
3. Constant-time: top()

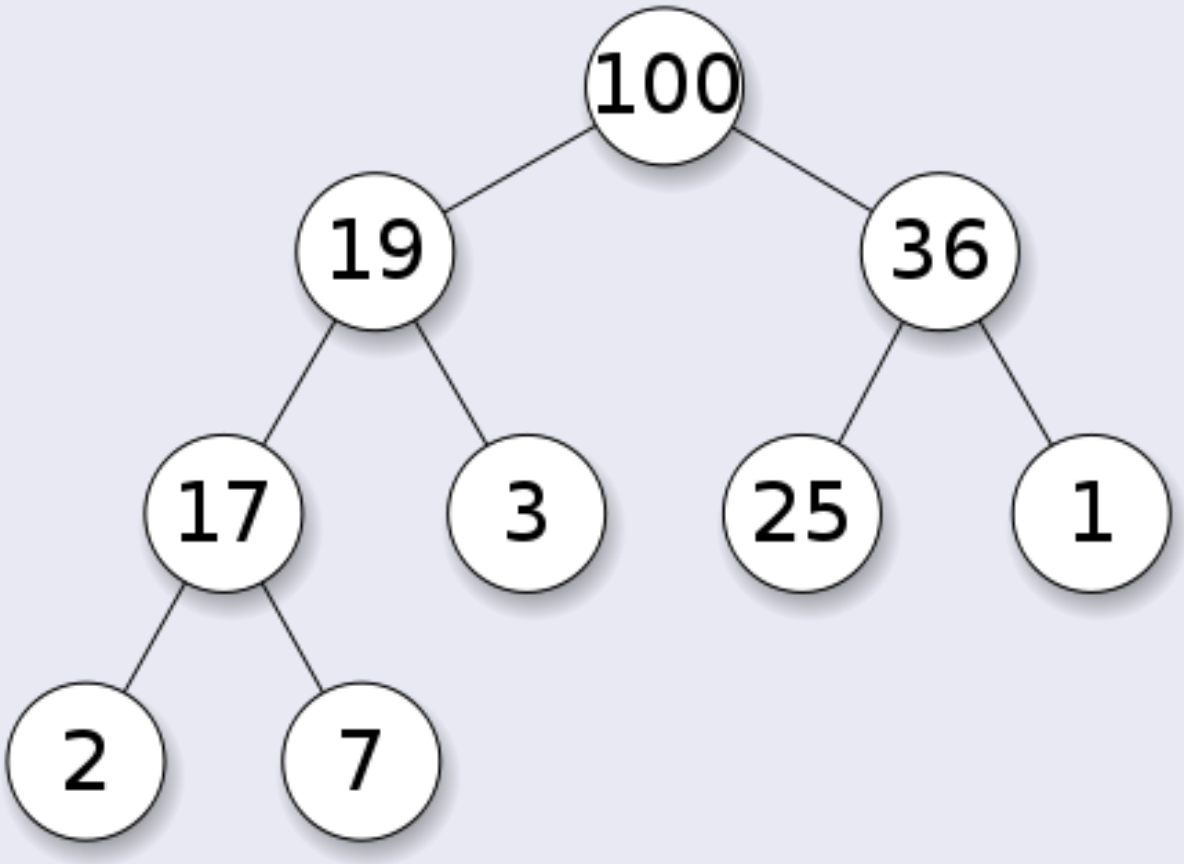
Heap

1. Complete binary tree with the heap property: value of a node $\geq$ values of its children
2. The root node has the maximum value
3. Constant-time: top()
4. Inserting/removing a node can be done in O(log n) time without breaking the heap property

Heap

1. Complete binary tree with the heap property: value of a node $\geq$ values of its children
2. The root node has the maximum value
3. Constant-time: top()
4. Inserting/removing a node can be done in O(log n) time without breaking the heap property
5. May need rearrangement of some nodes

Heap



Heap

Inserting a Node

1. Make a new node in the last level, as far left as possible. If the last level is full, make a new one

Heap

Inserting a Node

1. Make a new node in the last level, as far left as possible. If the last level is full, make a new one
2. If the new node breaks the heap property, swap with its parent node. The new node moves up the tree, which may introduce another conflict

Heap

Inserting a Node

1. Make a new node in the last level, as far left as possible. If the last level is full, make a new one
2. If the new node breaks the heap property, swap with its parent node. The new node moves up the tree, which may introduce another conflict
3. Repeat 2 until all conflicts are resolved

Heap

Inserting a Node

1. Make a new node in the last level, as far left as possible. If the last level is full, make a new one
2. If the new node breaks the heap property, swap with its parent node. The new node moves up the tree, which may introduce another conflict
3. Repeat 2 until all conflicts are resolved
4. Running time = tree height = O(log n)

Heap

Deleting the Root Node

1. Remove the root, and bring the last node (rightmost node in the last level) to the root

Heap

Deleting the Root Node

1. Remove the root, and bring the last node (rightmost node in the last level) to the root
2. If the root breaks the heap property, look at its children and swap it with the larger one. Swapping can introduce another conflict

Heap

Deleting the Root Node

1. Remove the root, and bring the last node (rightmost node in the last level) to the root
2. If the root breaks the heap property, look at its children and swap it with the larger one. Swapping can introduce another conflict
3. Repeat 2 until all conflicts are resolved

Heap

Deleting the Root Node

1. Remove the root, and bring the last node (rightmost node in the last level) to the root
2. If the root breaks the heap property, look at its children and swap it with the larger one. Swapping can introduce another conflict
3. Repeat 2 until all conflicts are resolved
4. Running time = O(log n)

Outline

Introduction to Bitwise Operators

1. Numbers are stored in binary and processing on bits is way faster.

Introduction to Bitwise Operators

1. Numbers are stored in binary and processing on bits is way faster.
2. Our weapons:
   << (left shift); >> (right shift); & (bitwise and);
   | (bitwise or); ^ (bitwise xor); ~ (bitwise not)

Introduction to Bitwise Operators

1. Numbers are stored in binary and processing on bits is way faster.
2. Our weapons:
   $<<$ (left shift); $>>$ (right shift); & (bitwise and);
   | (bitwise or); ^ (bitwise xor); $\sim$ (bitwise not)
3. Speed up the code by upto 100 times. Caution: try to use bitwise operations on unsigned integers only

Beauty of Bitwise

1. Example:

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.
   - Generate all subsets of S in $2^{|S|}$ time

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.
   - Generate all subsets of S in $2^{|S|}$ time
   - Generate all subsets of 1..n changing one bit at a time

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.
   - Generate all subsets of S in $2^{|S|}$ time
   - Generate all subsets of 1..n changing one bit at a time
   - Generate all subsets of S which have exactly t elements

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.
   - Generate all subsets of S in $2^{|S|}$ time
   - Generate all subsets of 1..n changing one bit at a time
   - Generate all subsets of S which have exactly t elements
   - Count the number of elements of elements in a set S

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.
   - Generate all subsets of S in $2^{|S|}$ time
   - Generate all subsets of 1..n changing one bit at a time
   - Generate all subsets of S which have exactly t elements
   - Count the number of elements of elements in a set S
   - Remove smallest element from S

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.
   - Generate all subsets of S in $2^{|S|}$ time
   - Generate all subsets of 1..n changing one bit at a time
   - Generate all subsets of S which have exactly t elements
   - Count the number of elements of elements in a set S
   - Remove smallest element from S
   - Check if $|S| = 1$

Beauty of Bitwise

1. Example:
   - Any subset of 0,1. . . 31 is a single int
   - Do set union/intersection/complement in one operation increment/decrement all elements by x in one operation
2. Even more:
   - Find if $x \in S$.
   - Generate all subsets of S in $2^{|S|}$ time
   - Generate all subsets of 1..n changing one bit at a time
   - Generate all subsets of S which have exactly t elements
   - Count the number of elements of elements in a set S
   - Remove smallest element from S
   - Check if $|S| = 1$
3. Never multiply or divide or take remainder modulo power of 2

Outline

Problems

Added on the contest on VOC http://ahmed-aly.com/voc/
Contest ID: 2578
Name: ACA, IITK LOP 01
Author: pnkjjindal
Links:

1. http://spoj.pl/problems/WEIRDFN
2. http://www.spoj.pl/problems/HOMO/
3. http://spoj.pl/problems/HISTOGRA
4. http://spoj.pl/problems/SUBSEQ
5. http://www.spoj.pl/problems/NGM2/
6. http://www.spoj.pl/problems/JOCHEF
7. http://www.spoj.pl/problems/SWTHIN/
8. http://www.spoj.pl/problems/LAZYPROG/