Preparing for Olympiads

Bruce Merry

University of Cape Town

10 May 2012

Outline

Who Am I?

- South African competition programmer
- IOI 2006–2011
- ACM ICPC 2002–2003
- USACO, Topcoder, Code Jam, Challenge24, IPSC, ICFP, ...

Who Am I?

- Post-doctoral researcher at UCT
- Research in GPU computing

Image copyright Adrian Frith. Used under CC-BY-SA.

Why Am I Here?

Three talks today

- How I prepare for olympiads
- Turing Machines
- Geometry with complex numbers

Scope

I'll talk mostly about ICPC/Code Jam/TC Algorithm etc style:

- Computational problems
- Well-defined bounds on problem size
- Answers are either right or wrong
- Big-O complexity is important
- Micro-optimisation not important
- Objective testing using test data

The Secret To Success

The Secret To Success

There isn't one.

The Secret To Success

There isn't one.

- Natural ability
- Experience
- Training
- Practise
- Skills
- Strategy

Outline

Algorithms

- Bellman-Ford
- Biconnected components
- Binary search
- Breadth-first search
- Bresenham's algorithm
- Delaunay triangulation
- Depth-first search
- Boyer-Moore
- Dijkstra's algorithm
- Dynamic programming
- Euclidean algorithm
- Eulerian paths
- Finite state machines
- Fast exponentiation
- Floyd-Warshall
- Ford-Fulkerson
- Graham scan
- Knuth-Morris-Pratt
- Linear programming
- Matching
- Min-cost network flow
- Minimax
- Memoisation
- Prim's algorithm
- Rabin-Karp
- Sorting
- Stable marriage problem
- Strongly connected components
- Recursion
- Rotating calipers
- Sorting algorithms
- Kruskal's algorithm

Data Structures

- Arrays
- Balanced binary trees
- Binary heaps
- Binary indexed trees
- Binary trees
- Bit vectors
- Deques
- Hash tables
- Interval trees
- Graphs
- Linked lists
- Queues
- Range trees
- Stacks
- Suffix trees
- Tries
- Union-find trees

Mathematics

- Proof techniques
- Complexity analysis
- NP-completeness
- Problem transformations
- Binary representations
- Combinatorics
- Modular arithmetic
- Invariants
- Probabilities
- Prime factorisations
- Linear algebra

Tips

To master an algorithm/structure

- Learn it

Tips

To master an algorithm/structure

- Learn it
- Implement it

Tips

To master an algorithm/structure

- Learn it
- Implement it
- Teach it to someone else

Tips

To master an algorithm/structure

- Learn it
- Implement it
- Teach it to someone else
- Use it until you've memorised it

Tips

To master an algorithm/structure

- Learn it
- Implement it
- Teach it to someone else
- Use it until you've memorised it
- Be able to implement it fast

Outline

Do Contests

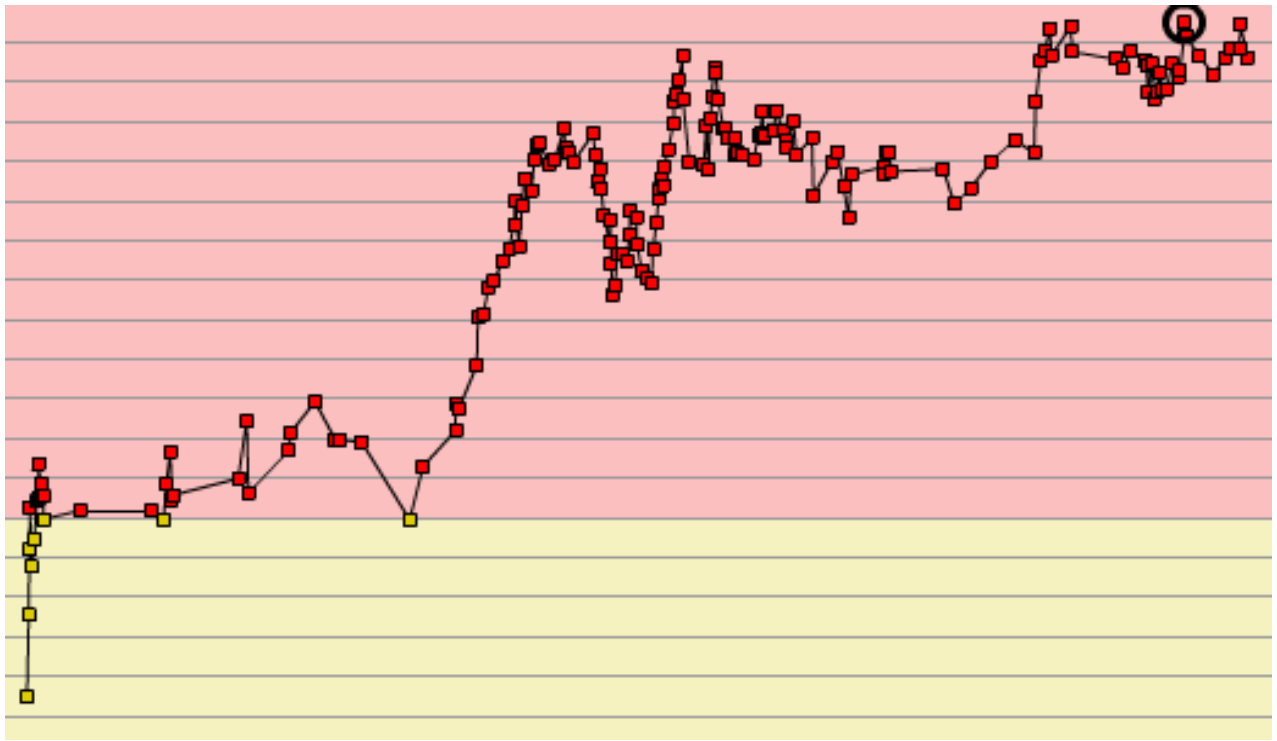
Top 10 Topcoder Algorithm competitors:

Rank	Handle	Matches
1	Petr	288
2	tourist	111
3	ACRush	178
4	bmerry	170
5	nika	139
6	tomek	171
7	wata	174
8	UdH-WiNGeR	96
9	marek.cygan	219
9	dzhulgakov	96

As of 15/04/2012

Do Contests

My Topcoder rating history



Learn From Mistakes

- Debug submissions that failed

Learn From Mistakes

- Debug submissions that failed
- Learn how to solve the problems you couldn't

Learn From Mistakes

- Debug submissions that failed
- Learn how to solve the problems you couldn't
- See how others solved it

Adapt To Avoid Mistakes

Binary Search

- Find largest i in $[0, N-1]$ s.t. $f(i)$ is true
- $f(0)$ is true, $f(i+1) \Rightarrow f(i)$

Adapt To Avoid Mistakes

Binary Search

- Find largest i in $[0, N-1]$ s.t. $f(i)$ is true
- $f(0)$ is true, $f(i+1) \Rightarrow f(i)$

```
int L = 0; // search [L, R]
int R = N - 1;
while (L != R) {
    int M = (L + R) / 2;
    if (f(M))
        L = M;
    else
        R = M - 1;
}
return L;
```

Adapt To Avoid Mistakes

Binary Search

- Find largest i in $[0, N-1]$ s.t. $f(i)$ is true
- $f(0)$ is true, $f(i+1) \Rightarrow f(i)$

```
int L = 0; // search [L, R]
int R = N - 1;
while (L != R) {
    int M = (L + R) / 2;
    if (f(M))
        L = M;
    else
        R = M - 1;
}
return L;
```

```
int L = 0; // f(L) is true
int R = N; // f(R) is false
while (R - L > 1) {
    int M = (L + R) / 2;
    if (f(M))
        L = M;
    else
        R = M;
}
return L;
```

Outline

Tools

Software tools can find errors

- Compiler flags
- Debugger
- Profiler
- Memory checker
- Library assertions

Libraries

Don’t reinvent the wheel

- Know the language containers in detail
- Know the complexity of operations

Outline

Keep it simple

Everything should be made as simple as possible, but no simpler. — Albert Einstein (paraphrased)

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. — Brian W. Kernighan

Simplification Examples

- Use the simplest algorithm that is fast enough.

Simplification Examples

- Use the simplest algorithm that is fast enough.
- If in doubt, use 64-bit integers.

Simplification Examples

- Use the simplest algorithm that is fast enough.
- If in doubt, use 64-bit integers.
- Allocate slightly more memory than you need.

Simplification Examples

- Use the simplest algorithm that is fast enough.
- If in doubt, use 64-bit integers.
- Allocate slightly more memory than you need.
- Don’t try to put everything on one line.

Simplification Examples

- Use the simplest algorithm that is fast enough.
- If in doubt, use 64-bit integers.
- Allocate slightly more memory than you need.
- Don't try to put everything on one line.
- Don't repeat yourself. Write a function.

Write Less Code

Part of a solution to a TC problem

```
public class EvenRoute {
    public String isItPossible(int[] x, int[] y, int wantedParity) {
      boolean hasFP = false, hasFU = false;
      for (int i=0; i<x.length; ++i)
        if ((1 & (x[i]+y[i])) == 0)
          hasFP = true; else hasFU = true;
      boolean hasN = false;
      for (int i=0; i<x.length; ++i) for (int j=0; j<x.length; ++j)
        if (((x[i]−x[j] + y[i]−y[j]) & 1) == 1) hasN = true;
        if (wantedParity == 0) {
          if (!hasFP && !hasN) return "CANNOT";
          else return "CAN";
        } else {
          if (hasFU || hasN) return "CAN";
          else return "CANNOT";
        }
    }
}
```

Write Less Code

Another solution:

```
public class EvenRoute {
   public String isItPossible(int[] x, int[] y, int wantedParity) {
      int n = x.length;
      for (int i = 0; i < n; ++i) {
         if (Math.abs(x[i] + y[i]) % 2 == wantedParity) {
            return "CAN";
         }
      }
      return "CANNOT";
   }
}
```

Be Suspicious

- If it seems too easy, it might be

Be Suspicious

- If it seems too easy, it might be
- If you can't prove your algorithm is correct, try to break it

Be Suspicious

- If it seems too easy, it might be
- If you can't prove your algorithm is correct, try to break it
- Use assertions in your code

Be Suspicious

- If it seems too easy, it might be
- If you can't prove your algorithm is correct, try to break it
- Use assertions in your code
- Assume your code has a bug

Summary

- Do a lot of contests

Summary

- Do a lot of contests
- Learn how to solve problems

Summary

- Do a lot of contests
- Learn how to solve problems
- Keep things simple

Summary

- Do a lot of contests
- Learn how to solve problems
- Keep things simple
- Avoid subtle errors

Summary

- Do a lot of contests
- Learn how to solve problems
- Keep things simple
- Avoid subtle errors
- Have fun