NOI 2015

Ken Sung

Scientific Committee

- Chang Ee-Chien (AskOneGetOneFree)
- Stephan Frank (Sudoku)
- Mark Theng (Radioactive)
- Ranald Lam Yun Shao (BananaFarm)
- Wing-Kin Sung, Ken
- Steven Halim
- Wu Xin Yu

Chang Ee-Chien

Associate Professor

Department of Computer Science

National University of Singapore

- Mother tongue: PASCAL, BASIC
- 1st “working language”: COBOL (used during a summer job)
- Languages learned and forget: SETL, ADA, PROLOG, LISP.
- Most used: C++ and MATLAB
- Favorite editor for programming: vi

- Research Interests: Information Security, Algorithm

Stephan, Frank

Professor

Department of Computer Science

Department of Mathematics

National University of Singapore

- Learnt at Secondary School: BASIC, MC6800 codes and PASCAL
- Languages learnt and almost forgotten: Fortran and APL
- Most used: C and Javascript
- Favourite editor for programming: vi

Research Fields: Mathematical Logic and Theory of Computation

Mark Theng

SG IOI Team 2013 - 2014

HCI NOI Team 2010 - 2013

HCI Alumni

- Mother tongue: C++
- Frequently used: C++, Python, MATLAB, Javascript
- Favorite editors: Code::Blocks (for C++) Notepad++

Ranald Lam Yun Shao

SG IOI Team 2012-2014

RI NOI Team 2011-2012

Raffles Institution Alumni

- Mother tongue: C++
- Frequently used: C++, Javascript, PHP
- Favorite editor(s): Sublime Text, Geany
- Bad Habits: Buying too much servers
- IOI/NOI Tip: Every mark counts ☺

Wing-Kin Sung, Ken

Professor

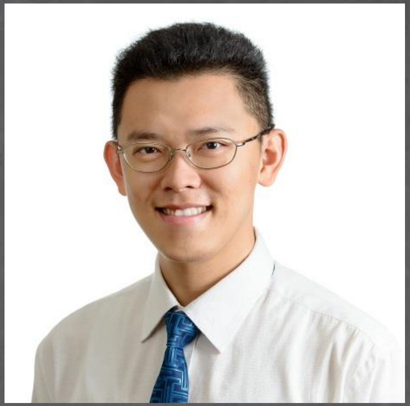
Department of Computer Science

National University of Singapore

- Mother tongue: BASIC and PASCAL
- 1st “working language”: FOXPRO (used during a summer job)
- Languages learned and forget: COBOL, PROLOG, LISP
- Most used: C, R and Java
- Favorite editor for programming: vi

- Research Interests: Computational Biology, Algorithm

Steven Halim



Lecturer, NUS ACM ICPC & SG IOI team leader
Department of Computer Science
National University of Singapore

- Languages most used: C/C++, Java, JavaScript, HTML5
- Other Languages: CSS3, PHP, SQL, MATLAB, C#, PASCAL, VB
- Favorite editor for programming: Sublime Text 2
- Research Interest: Algorithm Visualization (http://visualgo.net)
- Author of Competitive Programming textbook
  - Task bananafarm, up to 67 points (yeah, NOI15 bronze), can be easily solved with the SAMPLE CODE given in page 57-58 of CP3 + a bit extra ☺

Wu Xin Yu

SG IOI 2014

RI NOI 2013-14

Raffles Girls' School/

Raffles Institution Alumni

- Mother tongue: C++
- Most used: C++, JavaScript, Python
- Favorite editor(s): vim, sublime text
- Favorite moment in programming: Learn how to quit in vim

General statistics

- Number of partcipants: 111
- Number of >0 marks: 67 out of 111
- Number of $\geq$100 marks: 29 out of 111
- Max mark: 284 out of 400
- Most difficult question:
  - Sudoku (10 non-zero marks)
- Easiest question:
  - AskOneGetOneFree (58 non-zero marks)

AskOneGetOneFree

Two Unknown Problem

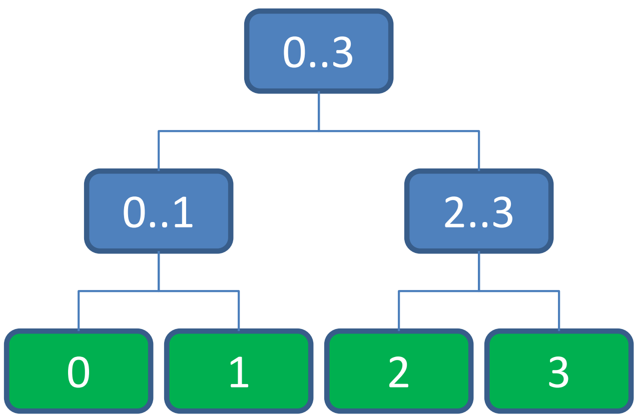
- Guess two integers x and y in 0..N-1.
- You can ask query(r), which tell you whether $x \geq r$ and whether $y \geq r$.
- By asking minimum number of queries, you need to discover x and y.

A simpler problem: One Unknown Problem

- Guess one integer x in 0..N-1.
- You can ask query(r), which tells you whether $x \geq r$.
- You need to discover x.

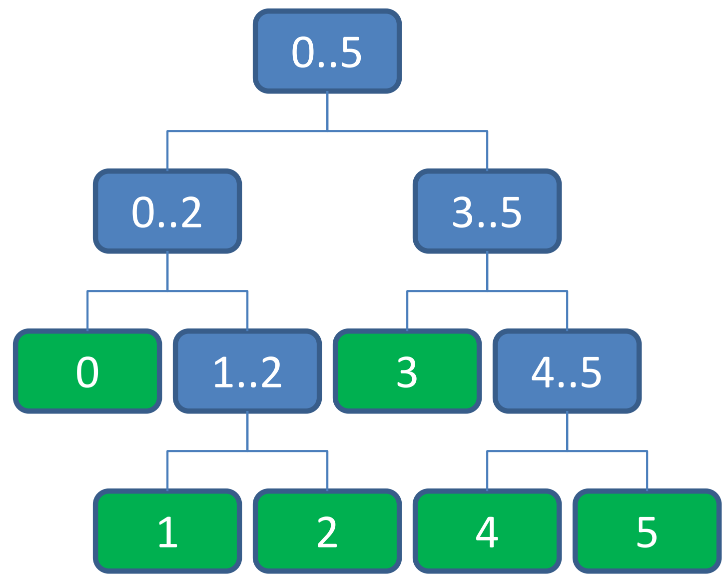
Binary search can solve the One Unknown Problem

N=4



2 queries

N=6



3 queries

In general, to guess x in 0..N-1, it requires $\lceil \log_2 N \rceil$ queries.

Why binary search is good?

- Let T(N) be the minimum number of queries to solve the One Unknown Problem.

- For some r, after we call query(r), there are two cases:
  - Case 1 (x<r): T(N) = 1 + T(r)
  - Case 2 (x≥r): T(N) = 1 + T(N-r)

- We have T(1)=0 and
  - $T(N) = \min_{1 \leq r < N} \left\{ max \begin{Bmatrix} 1 + T(r) \\ 1 + T(N-r) \end{Bmatrix} \right\}$ for N>0

- To minimize T(N), we should set $r = \left\lceil \frac{N}{2} \right\rceil$. Hence, binary search is good.

Two Unknown Problem

- Let T(N) be the minimum number of queries to solve Two Unknown Problem.
- After we call query(r), there are three cases:
  - Case 1 (Both x,y $\geq$ r): T(N) = 1+T(N-r)
  - Case 2 (Both x,y < r): T(N) = 1+T(r)
  - Case 3 (x < r and y $\geq$ r): T(N) = 1 + lg r + lg (N-r)

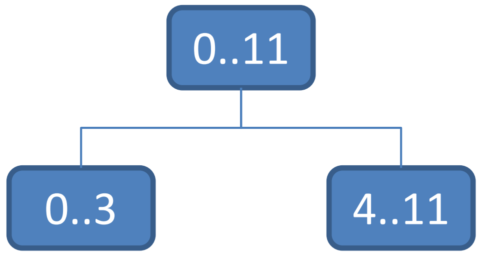
- To minimize the number of queries, we have
  - $T(N) = \min_{r}\left\{ max \left\{ \begin{array}{c} 1 + T(r) \\ 1 + T(N-r) \\ 1 + \lceil \log r \rceil + \lceil \log(N-r) \rceil \end{array} \right\} \right\}$
- By dynamic programming, we can compute T(N) and the corresponding r that minimizes T(N).

r and T(N) from DP

N	r	T(N)
1	-	0
2	1	1
3	1	2
4	1	3
5	1	4
6	2	4
7	1	5
8	2	5
9	1	6
10	2	6
11	3	6
12	4	6
13	1	7
14	2	7
15	3	7
16	4	7
17	1	8
18	2	8
19	3	8

Binary search may not be good for the Two Unknown Problem

- When N=12, the optimal solution calls query(4).



- The worst case is x<4 and y≥4. The number of queries is $1 + \lceil \log 3 \rceil + \lceil \log 8 \rceil = 1+2+3=6$.

- Instead, if we do binary search (i.e. call query(6)), the worst case is x<4 and y≥4. The number of queries is $1+2* \lceil \log 6 \rceil = 1+2*3=7$.

Sudoku

Problem

- Given the positions of 1's, find a Sudoku that has θ space entries and are human solvable.

			1					
						1		
	1							
				1				
							1	
		1						
					1			
								1

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	4	5	6
0	0	0	7	8	9	0	0	0
0	0	2	0	0	5	0	0	0
0	0	8	0	0	0	3	4	0
0	0	5	6	7	8	0	0	2
0	9	0	0	3	4	0	0	7
0	6	0	0	0	0	2	0	4
0	3	0	5	6	7	8	9	1

What is human solvable?

- For example, for the highlighted entry,
  - its row: 2, 4, 6
  - its column: 4, 5, 7, 8, 9
  - its quadrant: 3, 4, 5, 6, 7

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	4	5	6
0	0	0	7	8	9	0	0	0
0	0	2	0	0	5	0	0	0
0	0	8	0	0	0	3	4	0
0	0	5	6	7	8	0	0	2
0	9	0	0	3	4	0	0	7
0	6	0	0	0	0	2	0	4
0	3	0	5	6	7	8	9	1

- The row, column and quadrant contain all digits except 1.
- Hence, this entry should be 1.
- This is human solvable.

How to get a sudoku with many space entries?

- Method 1: Create it manually!
- E.g. below transformation gives 27 spaces.

1	2	3	4	5	6	7	8	9
7	8	9	1	2	3	4	5	6
4	5	6	7	8	9	1	2	3
9	1	2	3	4	5	6	7	8
6	7	8	9	1	2	3	4	5
3	4	5	6	7	8	9	1	2
8	9	1	2	3	4	5	6	7
5	6	7	8	9	1	2	3	4
2	3	4	5	6	7	8	9	1

0	0	0	4	5	6	7	8	9
0	0	0	1	2	3	4	5	6
0	0	0	7	8	9	1	2	3
9	1	2	0	0	0	6	7	8
6	7	8	0	0	0	3	4	5
3	4	5	0	0	0	9	1	2
8	9	1	2	3	4	0	0	0
5	6	7	8	9	1	0	0	0
2	3	4	5	6	7	0	0	0

Can we create empty spaces?

- Let A be a full sudoku. We aim to obtain θ empty entries.
- By greedy approach, identify an entry that is human solvable and make it as 0. Iterate this step until you have θ's zeros.
- Run createEmpty(A, θ).

- createEmpty(A, θ)
  - if A has θ empty entries, return A;
  - for each non-empty entry x in A
    - if A-x is human-solvable, then
      - return createEmpty(A-x, θ-1);
  - return fail;

Example to generate 51 spaces

- By greedy approach, identify an entry that is human solvable and make it as 0. Iterate this step until you have 51's zeros.
- This step can be done in offline.

1	2	3	4	5	6	7	8	9
7	8	9	1	2	3	4	5	6
4	5	6	7	8	9	1	2	3
9	1	2	3	4	5	6	7	8
6	7	8	9	1	2	3	4	5
3	4	5	6	7	8	9	1	2
8	9	1	2	3	4	5	6	7
5	6	7	8	9	1	2	3	4
2	3	4	5	6	7	8	9	1

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	4	5	6
0	0	0	7	8	9	0	0	0
0	0	2	0	0	5	0	0	0
0	0	8	0	0	0	3	4	0
0	0	5	6	7	8	0	0	2
0	9	0	0	3	4	0	0	7
0	6	0	0	0	0	2	0	4
0	3	0	5	6	7	8	9	1

How can we rearrange the 1’s?

- By swapping rows1-3, rows4-6 and rows7-9, the sudoku property is still valid.
- By swapping cols1-3, cols4-6 and cols7-9, the sudoku property is still valid.
- By swapping the rows/columns, we can place the 1’s in the correct positions.

1	2	3	4	5	6	7	8	9
7	8	9	1	2	3	4	5	6
4	5	6	7	8	9	1	2	3
9	1	2	3	4	5	6	7	8
6	7	8	9	1	2	3	4	5
3	4	5	6	7	8	9	1	2
8	9	1	2	3	4	5	6	7
5	6	7	8	9	1	2	3	4
2	3	4	5	6	7	8	9	1

Swap row1 & row2

7	8	9	1	2	3	4	5	6
1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
9	1	2	3	4	5	6	7	8
6	7	8	9	1	2	3	4	5
3	4	5	6	7	8	9	1	2
8	9	1	2	3	4	5	6	7
5	6	7	8	9	1	2	3	4
2	3	4	5	6	7	8	9	1

Swap col1 & col2

8	7	9	1	2	3	4	5	6
2	1	3	4	5	6	7	8	9
5	4	6	7	8	9	1	2	3
1	9	2	3	4	5	6	7	8
7	6	8	9	1	2	3	4	5
4	3	5	6	7	8	9	1	2
9	8	1	2	3	4	5	6	7
6	5	7	8	9	1	2	3	4
3	2	4	5	6	7	8	9	1

How to rearrange the 1’s?

1	2	3	4	5	6	7	8	9
7	8	9	1	2	3	4	5	6
4	5	6	7	8	9	1	2	3
9	1	2	3	4	5	6	7	8
6	7	8	9	1	2	3	4	5
3	4	5	6	7	8	9	1	2
8	9	1	2	3	4	5	6	7
5	6	7	8	9	1	2	3	4
2	3	4	5	6	7	8	9	1

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	4	5	6
0	0	0	7	8	9	0	0	0
0	0	2	0	0	5	0	0	0
0	0	8	0	0	0	3	4	0
0	0	5	6	7	8	0	0	2
0	9	0	0	3	4	0	0	7
0	6	0	0	0	0	2	0	4
0	3	0	5	6	7	8	9	1

Swap rows1&2, rows4&5, rows7&9

					1			
	1							
						1		
				1				
1								
							1	
								1
			1					
		1						

7	8	9	1	2	3	4	5	6
1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8
3	4	5	6	7	8	9	1	2
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7

0	0	0	0	0	0	4	5	6
0	0	0	0	0	0	0	0	0
0	0	0	7	8	9	0	0	0
0	0	8	0	0	0	3	4	0
0	0	2	0	0	5	0	0	0
0	0	5	6	7	8	0	0	2
0	3	0	5	6	7	8	9	1
0	6	0	0	0	0	2	0	4
0	9	0	0	3	4	0	0	7

How to rearrange the 1's?

7	8	9	1	2	3	4	5	6
1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8
3	4	5	6	7	8	9	1	2
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7

0	0	0	0	0	0	4	5	6
0	0	0	0	0	0	0	0	0
0	0	0	7	8	9	0	0	0
0	0	8	0	0	0	3	4	0
0	0	2	0	0	5	0	0	0
0	0	5	6	7	8	0	0	2
0	3	0	5	6	7	8	9	1
0	6	0	0	0	0	2	0	4
0	9	0	0	3	4	0	0	7

Swap cols1&2, cols4&6

					1			
	1							
						1		
				1				
1								
							1	
								1
			1					
		1						

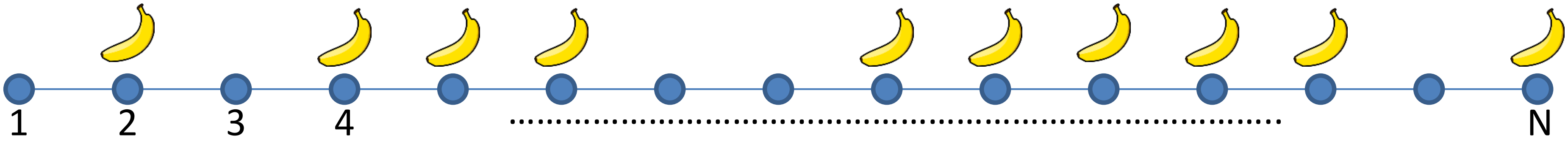
8	7	9	3	2	1	4	5	6
2	1	3	6	5	4	7	8	9
5	4	6	9	8	7	1	2	3
7	6	8	2	1	9	3	4	5
1	9	2	5	4	3	6	7	8
4	3	5	8	7	6	9	1	2
3	2	4	7	6	5	8	9	1
6	5	7	1	9	8	2	3	4
9	8	1	4	3	2	5	6	7

0	0	0	0	0	0	4	5	6
0	0	0	0	0	0	0	0	0
0	0	0	9	8	7	0	0	0
0	0	8	0	0	0	3	4	0
0	0	2	5	0	0	0	0	0
0	0	5	8	7	6	0	0	2
3	0	0	7	6	5	8	9	1
6	0	0	0	0	0	2	0	4
9	0	0	4	3	0	0	0	7

Radioactive

Radioactive Problem

- There are N houses in linear order.
- It is not possible for C consecutive houses to have bananas.
  - Otherwise, nuclear explosion would occur.
- Aim: Find number of possible ways to distribute the bananas.



Model as bit vector

- This problem is the same as finding number of len-N bit vectors that
  - don’t have runs of ones longer than C.
- Example: Below is an example of len-20 with no runs of ones longer than 3.
  - 01001110110100110011
- E.g. for N=3, C=1, there are 5 len-3 bit vectors with no runs of ones longer than 1:
  - 000
  - 100
  - 010
  - 001
  - 101
  - 011 ☹
  - 110 ☹
  - 111 ☹

Solution 1

- Let $S_C(N)$ be the number of combinations of len-N bit vectors where no runs of ones longer than C.
- $S_C(k) = 0$ for $k < 0$.
- $S_C(0) = 1$
- $S_C(1) = 2$
- $S_C(N) = \sum_{i=0..C} S_C(N - i - 1) \ mod \ p$

- By dynamic programming, $S_C(N)$ can be computed in O(NC) time.

You can get 31 marks!

Solution 2

- Lemma: $S_C(N) = \left(2S_C(N-1) - S_C(N-C-1)\right) mod\ p$
- Proof:
  - $S_C(N) = \left(S_C(N-1) + \ \dots + S_C(N-C-1)\right) mod\ p$
  - $\quad = \left(S_C(N-1) + \left(S_C(N-2) + \ \dots + S_C(N-C-2)\right) - S_C(N-C-2)\right) mod\ p$
  - $\quad = \left(S_C(N-1) + S_C(N-1) - S_C(N-C-2)\right) mod\ p$
  - $\quad = \left(2S_C(N-1) - S_C(N-C-2)\right) mod\ p$

- By dynamic programming, $S_C(N)$ can be computed in O(N) time.

- Algorithm
  - $S_C(0) = 1$
  - $S_C(1) = 2$
  - for i = 2 to N
    - $S_C(i) = \left(2S_C(i-1) - S_C(i-C-1)\right) mod\ p$
  - Report $S_C(N)$;

- This solution is good when N is small.

You can get 58 marks!

Solution 3

- Another way is to partition the len-N vector into two len-$\frac{N}{2}$ vectors.

- For example, for C=3, there are a few cases (so that the middle has at most C's ones):

0	0	01	0
0	10	01	10
0	110	01	110
0	1110	011	0
01	10	011	10
01	110	0111	0

How to compute $S_C(N)$

- Note that, the number of combinations for the following case is $S_{C=3}\left(\frac{N}{2}-1\right)+S_{C=3}\left(\frac{N}{2}-4\right)$.

0	1110

- For C=3, $S_C(N)$ is the sum of below 12 values.

$S_{C=3}\left(\frac{N}{2}-1\right)+S_{C=3}\left(\frac{N}{2}-1\right)$	0	0	01	0	$S_{C=3}\left(\frac{N}{2}-2\right)+S_{C=3}\left(\frac{N}{2}-1\right)$
$S_{C=3}\left(\frac{N}{2}-1\right)+S_{C=3}\left(\frac{N}{2}-2\right)$	0	10	01	10	$S_{C=3}\left(\frac{N}{2}-2\right)+S_{C=3}\left(\frac{N}{2}-2\right)$
$S_{C=3}\left(\frac{N}{2}-1\right)+S_{C=3}\left(\frac{N}{2}-3\right)$	0	110	01	110	$S_{C=3}\left(\frac{N}{2}-2\right)+S_{C=3}\left(\frac{N}{2}-3\right)$
$S_{C=3}\left(\frac{N}{2}-1\right)+S_{C=3}\left(\frac{N}{2}-4\right)$	0	1110	011	0	$S_{C=3}\left(\frac{N}{2}-3\right)+S_{C=3}\left(\frac{N}{2}-1\right)$
$S_{C=3}\left(\frac{N}{2}-2\right)+S_{C=3}\left(\frac{N}{2}-2\right)$	01	10	011	10	$S_{C=3}\left(\frac{N}{2}-3\right)+S_{C=3}\left(\frac{N}{2}-2\right)$
$S_{C=3}\left(\frac{N}{2}-2\right)+S_{C=3}\left(\frac{N}{2}-3\right)$	01	110	0111	0	$S_{C=3}\left(\frac{N}{2}-4\right)+S_{C=3}\left(\frac{N}{2}-1\right)$

Solution 3

- $S_C(N)$ {
  - For i = 0 to C
    - Set $u_i = S_C(\frac{N}{2} - i - 1)$;
  - Return $\sum_{i+j \leq C}(u_i \cdot u_j)$;
- }

- This algorithm takes O(C^2 log N) time.
- This solution is good when C is small but N is large.

You can get 100 marks!

BananaFarm

Problem

- Input:
  - N trees, i^{th} tree can harvest B[i] bananas.
  - There are P plans
    - The j^{th} plan tries to harvest C_j trees between $B[S_j..E_j]$.

- For the j^{th} plan, we need to compute the C_j largest integer in $B[S_j..E_j]$.

Abstract problem

- Consider an integer array arr[1..N].

- For each input (S, E, K),
  - Our aim is to find the K^{th} largest element in arr[S..E].
  - Denote this number as select(S..E, K).

Simple solution

- To find select(S..E, K),
  - We sort arr[S..E].
  - Then, report the K^{th} largest one.

- This takes O((E-S) log (E-S)) time.

But doing this… gets you 0 marks…

Subtask 1 Solution

- S = 1 and E = N
  - Implies all trees are harvested
- Sort arr[1..N] ascendingly into sorted[1..N]
  - **select(S, E, K) = sorted[N-K+1] (1-indexed)**

You can get 13 marks!

Subtask 2 Solution

- S = 1, K = 1
  - Implies only the maximum number is required
  - All ranges start from the first tree
- max_from_front[1] = arr[1]
- max_from_front[2] = max(arr[1], arr[2])
- max_from_front[3] = max(arr[1], arr[2], arr[3])
- max_from_front[i] = max(arr[1..i])
- **max_from_front[i] = max(arr[i], max_from_front[i-1])**
- **select(S, E, K) = max_front_front[E]**

You can get 13 + 12 = 25 marks!

Subtask 3 Solution

- K = 1
- Build data-structure (segment tree)
- Then, utilize the data-structure, answer the query select(S..E, K).
- Example: Segment tree for Arr[1..8]= (3, 6, 2, 8, 5, 1, 7, 4).

<table>
<tr><td colspan="8">8</td></tr>
<tr><td colspan="4">8</td><td colspan="4">7</td></tr>
<tr><td colspan="2">6</td><td colspan="2">8</td><td colspan="2">5</td><td colspan="2">7</td></tr>
<tr><td>3</td><td>6</td><td>2</td><td>8</td><td>5</td><td>1</td><td>7</td><td>4</td></tr>
</table>

1	2	3	4	5	6	7	8

Observation

- Any interval S..E can be partitioned into at most 2 log N intervals in the segment tree.

- Example, 2..8 can be partitioned into
  - 2..2, 3..4, 5..8

<table>
<tr><td colspan="8">8</td></tr>
<tr><td colspan="4">8</td><td colspan="4">7</td></tr>
<tr><td colspan="2">6</td><td colspan="2">8</td><td colspan="2">5</td><td colspan="2">7</td></tr>
<tr><td>3</td><td>6</td><td>2</td><td>8</td><td>5</td><td>1</td><td>7</td><td>4</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr>
</table>

So… how do I code it?

- NOI is open book ☺
- CP3 Supporting Material
  - https://sites.google.com/site/stevenhalim/home/material
  - ch2.zip → ch2_09_segmenttree_ds.cpp

You can get 13 + 12 + 21 = 46 marks!

By buying CP3 :O

Subtask 4

- K = 1 or 2
- Two options
  - Code a segment tree with updates
  - Code a segment tree that stores the two max values per range

You can get 13 + 12 + 21 + 21 = 67 marks!

Better solution

- Instead of storing the max, store all the elements in sorted order
  - Recall Subtask 1 solution and Subtask 3 solution
- Example: Segment tree for Arr[1..8]= (3, 6, 2, 8, 5, 1, 7, 4).

<table>
<tr><td colspan="8">1,2,3,4,5,6,7,8</td></tr>
<tr><td colspan="4">2,3,6,8</td><td colspan="4">1,4,5,7</td></tr>
<tr><td colspan="2">3, 6</td><td colspan="2">2, 8</td><td colspan="2">1, 5</td><td colspan="2">4,7</td></tr>
<tr><td>3</td><td>6</td><td>2</td><td>8</td><td>5</td><td>1</td><td>7</td><td>4</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr>
</table>

Observation

- Any interval S..E can be partitioned into at most 2 log N intervals in the segment tree.

- Example, 2..8 can be partitioned into
  - 2..2, 3..4, 5..8

<table>
<tr><td colspan="8">1,2,3,4,5,6,7,8</td></tr>
<tr><td colspan="4">2,3,6,8</td><td colspan="4">1,4,5,7</td></tr>
<tr><td colspan="2">3, 6</td><td colspan="2">2, 8</td><td colspan="2">1, 5</td><td colspan="2">4,7</td></tr>
<tr><td>3</td><td>6</td><td>2</td><td>8</td><td>5</td><td>1</td><td>7</td><td>4</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td></tr>
</table>

Rank(S..E, x)

- Denote rank(S..E, *x*) be the number of elements in arr[S..E] which are at least *x*.
- rank(S..E, *x*) can be found in O($\log^2$ N) time.
- Example: rank(2..8,4)
  - 2..8 can be partitioned into 3 segments: 2..2, 3..4 and 5..8.
  - Num of values ≥4 is rank(2..2,4)+rank(3..4,4)+rank(5..8,4)=5.

1,2,3,4,5,6,7,8							
2,3,6,8				1,4,5,7			
3, 6		2, 8		1, 5		4,7	
3	6	2	8	5	1	7	4
1	2	3	4	5	6	7	8

Select(S..E, K) takes O($\log^3$ N) time

- Our aim is to compute select(S..E, K)
  - That is, the K^{th} largest element in arr[S..E].
- select(S..E, K) can be computed by binary search *x* on 1..N such that rank(S..E, x) = K.

- We need to make at most log N queries rank(S..E, x).
- Hence, select(S..E, K) takes O($\log^3$ N) time.

Note

- There are better solutions to this problem
- Solution presented can pass time limit
- Time limit set such that the following can pass
  - $N \log N$
  - $N \log^2 N$
  - $N \log^3 N$
  - $N \operatorname{sqrt} N$

TRAINING SESSIONS

- Entire December → for NOI preparation and CS3233 (Competitive Programming) eligibility, **everybody can join**
- January – April, CS3233, Wednesday nights, for Singaporean/SPR, **by invitation only**
- June/July, intensive IOI trainings, **for Singapore top 4++ only**
- See: http://algorithmics.comp.nus.edu.sg/mediawiki/index.php/IOI_Workshop