T2622 Chess Tournament

Daniel Hsieh {QwertyPi}

2026-05-24

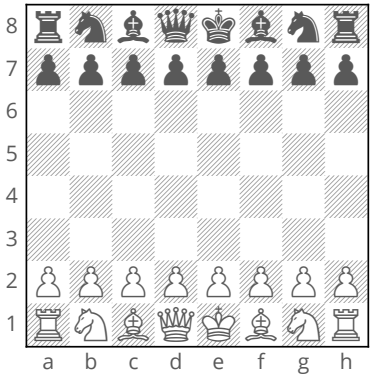
Table of Contents

Background

Problem Idea and Preparation by `QwertyPi`

Special Thanks to `gasbug`, `happypotato`, `hywong1`, `firewater`, `ITO` for testing!

One day I was watching a clip about the World Rapid Chess Championship, then the task "what is the fastest way to determine the best player" naturally pops up...



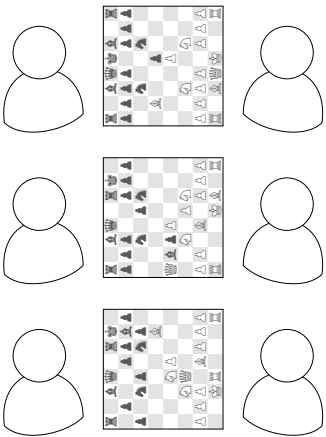
The Problem

Given N players of distinct ratings.

For each (two-player) game, the higher-rated player always wins without draw.

Each round consists of several games. Players cannot play two games simulatenously in the same round.

Find the top R players (with order) using minimal number of rounds.



Subtask Scoring

Subtask 1: $N = 8, R = 2$

K	Score
$K \leq 4$	30
$4 < K \leq 7$	$15 + 5 \times (7 - K)$
$7 < K \leq 15$	7

Subtask 2: $N = 4096, R = 8$

K	Score
$K \leq 19$	70
$19 < K \leq 22$	$61 + 3 \times (22 - K)$
$23 < K \leq 42$	$45 + 0.8 \times (42 - K)$
$42 < K \leq 80$	$26 + 0.5 \times (80 - K)$
$80 < K \leq 150$	$12 + 0.2 \times (150 - K)$

Statistics

#	Score	Predict #	Actual #
1	7	10	1
	15	10	1
	20	20	6
	25	15	29
	30	15	31

#	Score	Predict #	Actual #
2	$[12, 26)$	30	10
	$[26, 45)$	15	7
	$[45, 61)$	5	21
	$[61, 70)$	3	2
	70	3	4

First solved by **twg-210083** (Lo Ting Wai) at **1h 10m 41s**.

Table of Contents

Solution 1 (Repeat Find Max)

What is the simplest solution? Simply ignore the parallel part, and just use each round to compare two players!

For M players, we can find the highest-rated player among them using $M-1$ rounds. Therefore, the total number of rounds needed is

$$(N-1)+(N-2)+\cdots+(N-R)=RN-\frac{R(R+1)}{2}$$

	Sub 1	Sub 2	Total
K	13	32732	
Score	7	0	7

Solution 2 (Round-Robin)

What is the next simplest solution? Simply query for all pairs of players! The P-th (1-based) highest rated player should have won exactly $N - P$ times.

Since N is a power of 2, you can make round D ($1 \leq D \leq N - 1$) to be of games between players (X, Y) where X XOR Y is D.

	Sub 1	Sub 2	Total
K	7	4095	
Score	15	0	15

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

Table of Contents

Group Stage

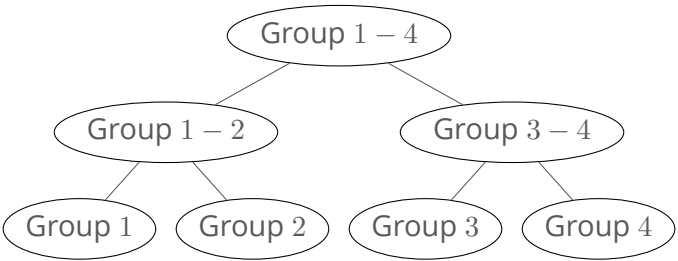
Querying for all pairs is clearly wasteful - since we only need top R players!

Observation 1

Once a player has lost to R other players, he/she cannot be in the top R players.

Simplest idea: Introduces *Group Stages* and eliminates players once they are no irrelevant to us.

Group Stage



Solution 3.1 (Group of 2R)

The idea is simple:

- Form groups of size $2R$. Then, query all pairs in the group.
- Eliminate all players who has lost at least R games within their group.

After each run, the number of remaining players is halved. The number of rounds needed is therefore

$$(2R-1) \times (\log_2 N - \log_2 R)$$

	Sub 1	Sub 2	Total
K	6	135	
Score	20	13.5	33.5

Solution 3.2 (Group of R)

After eliminating players, we still know the relative orders of the remaining players. Therefore, we can consider groups of R instead:

- Query all pairs in groups of R using $R-1$ rounds at the beginning.
- Merge two groups of R together into a group of $2R$ using R extra rounds, then eliminate R weakest players.

The total number of rounds needed is

$$(R-1)+R\times(\log_2 N-\log_2 R)$$

	Sub 1	Sub 2	Total
K	5	79	
Score	25	26.5	51.5

Merge Sort on Groups

Note that the above solution is exactly mimicing merge sort, except that whenever the group size exceeds R, we retain the largest R players only.

There are some natural questions you can ask:

1. Can we sort within the group of R using less than $R-1$ rounds?
2. Can we merge two groups together using less than R rounds?

Turns out the answer to both of the questions are *yes* for $R=8$!

Solution 3.3 (Group of R++)

Suppose the sort cost and merge cost are X and Y respectively, then the number of rounds required is

$$X + Y \times (\log_2 N - \log_2 R)$$

The optimal bound for X and Y are 6 and 4, both can be attained via *bitonic sorter* or decision tree. There exists heuristics to attain $Y = 5$ or 6 as well.

	Sub 1	Sub 2	Total
K	5	[42,79]	
Score	25	[26.5,45]	[51.5,70]

Table of Contents

Binary Search, revisited

"Binary Search"

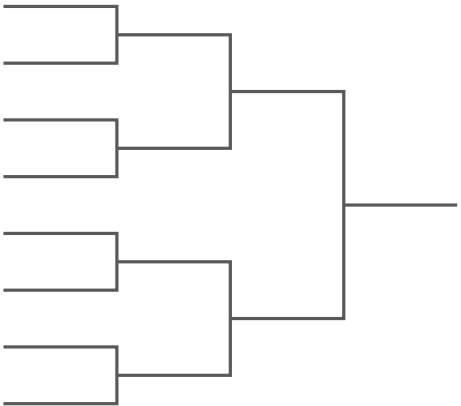
Suppose there are M candidates who potentially have the highest rating. Then we can find the highest-rated player in $\lceil \log_2 M \rceil$ rounds.

Idea: Whenever $M > 1$, then we

- Create $\lfloor \frac{M}{2} \rfloor$ games between the candidates.
- Eliminate those loses the game since they cannot have the highest rating.
- Update $M' := \lceil \frac{M}{2} \rceil$.

Binary Search, revisited

Actually it is just the tournament brackets you see in daily life:



Solution 4.1 (Repeat Binary Search)

We can simply find the highest-rated player for R times using binary search. The number of rounds is bounded by

$$R\lceil \log_2 N \rceil$$

	Sub 1	Sub 2	Total
K	6	96	
Score	20	17.4	37.4

Solution 4.2 (Binary Search, Candidate Reduction)

Observation 2

Candidates of the second highest-rated players are those players who have only lost to the highest-rated player in previous rounds.

Therefore, we can use $\lceil \log_2 N \rceil$ rounds to find the highest-rated player, then we binary search again on the $\lceil \log_2 N \rceil$ second-highest candidates. Repeating $\frac{R}{2}$ times the above steps, the number of rounds is bounded by

$$\frac{R}{2}(\lceil \log_2 N \rceil + \lceil \log_2 \lceil \log_2 N \rceil \rceil)$$

	Sub 1	Sub 2	Total
K	5	64	
Score	25	34	59

Solution 4.3 (Binary Search, Topological Sort)

Observation 3

Candidates of the P-th highest-rated players are those players who have only lost to the 1-st, 2-nd, ..., or $(P-1)$-th highest-rated player in previous rounds.

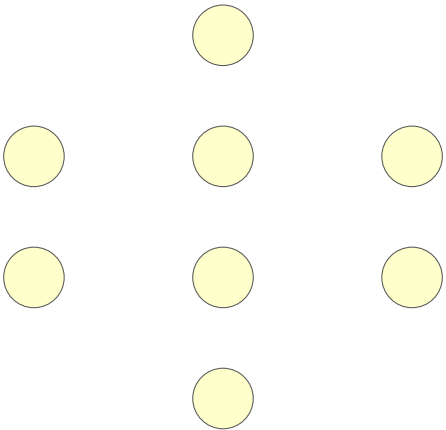
Thus, we can do binary search on the reduced set of candidates for each P, which is essentially topological sort. The number of rounds is bounded by

$$\lceil \log_2 N \rceil + \sum_{P=2}^{R} \lceil \log_2((P-1)\lceil \log_2 N \rceil) \rceil$$

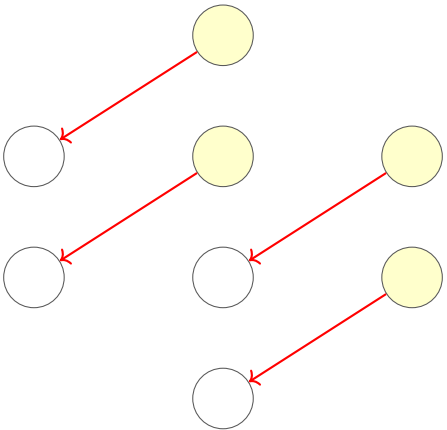
The actual number is lower since the bound is pretty loose.

	Sub 1	Sub 2	Total
K	5	42	
Score	25	45	70

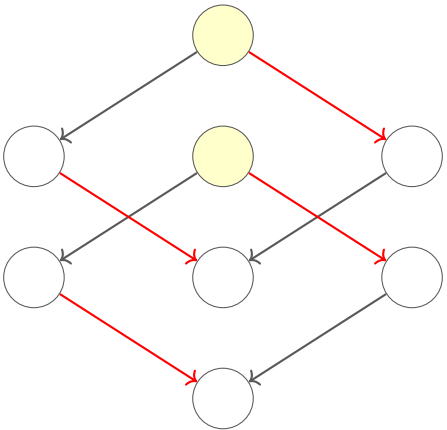
Topological Sort Example



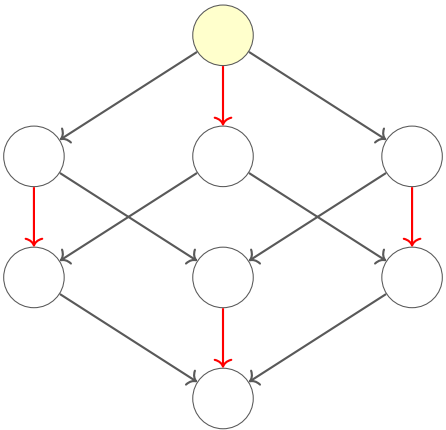
Topological Sort Example



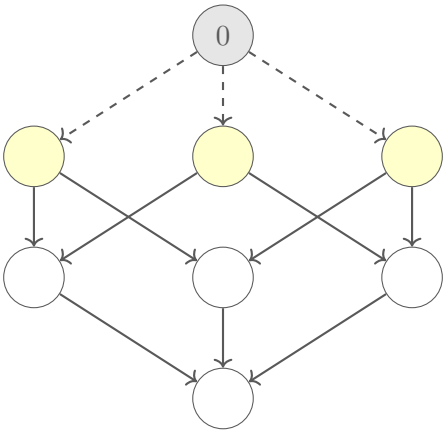
Topological Sort Example



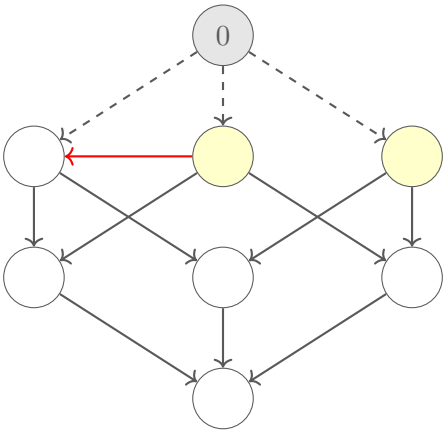
Topological Sort Example



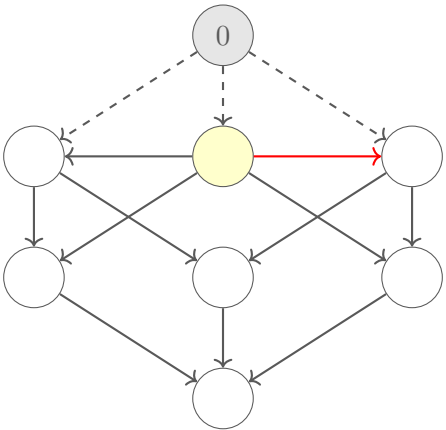
Topological Sort Example



Topological Sort Example



Topological Sort Example



Topological Sort Example

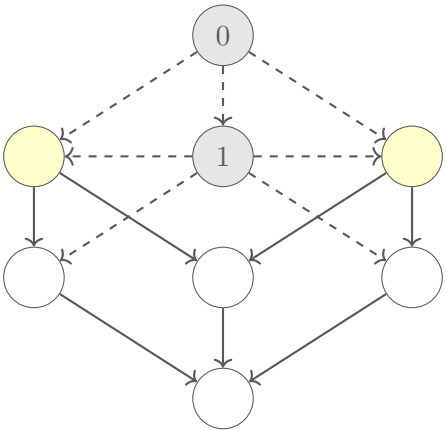


Table of Contents

Bitonic Sorter

Bitonic Sequence: A sequence of length $2N = 2^{k+1}$ which there exists a cyclic shift A where

$$A[0] \leq A[1] \leq \cdots \leq A[X] \geq \cdots \geq A[2N-1]$$

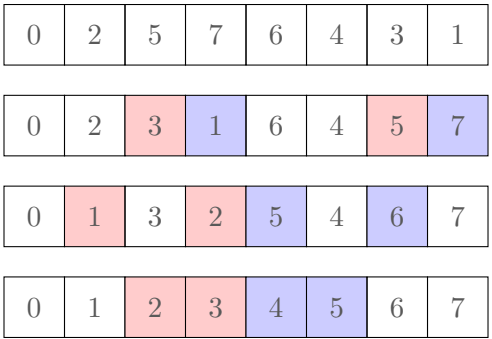
Turns out we can sort any bitonic sequence in $\log_2 2N = k+1$ rounds.

Suppose we pairwise compare $A[i]$ and $A[i+N]$ for $0 \leq i < N$, then swap them iff $A[i] > A[i+N]$. Then

1. $\max\{A[0], A[1], \ldots, A[N-1]\} \leq \min\{A[N], \ldots, A[2N-1]\}$
2. Both $\{A[0], \ldots, A[N-1]\}$ and $\{A[N], \ldots, A[2N-1]\}$ are bitonic.

Bitonic Sorter

We can thus sort any bitonic sequence in $\log_2 2N = k + 1$ rounds recursively:



Solution 5.1 (Bitonic Merge Sort)

Directly implementing the Bitonic Merge Sort (Merge Sort but merge with Bitonic Sorter), the number of rounds is

$$1 + 2 + \cdots + \log_2 N = \frac{\log_2 N(\log_2 N + 1)}{2}$$

	Sub 1	Sub 2	Total
K	6	78	
Score	20	27	47

Solution 5.2 (Bitonic Sorter, Group of R)

We can use the bitonic sorter to optimise base on the group of R solution:

- Sort within groups of R using $\frac{\log_2 R(\log_2 R+1)}{2}$ rounds at the beginning.
- Merge two groups of R into a group of $2R$ using $\log_2 R + 1$ extra rounds.

The total number of rounds becomes

$$\frac{\log_2 R(\log_2 R+1)}{2} + (\log_2 R + 1) \times (\log_2 N - \log_2 R)$$

	Sub 1	Sub 2	Total
K	5	42	
Score	25	45	70

Table of Contents

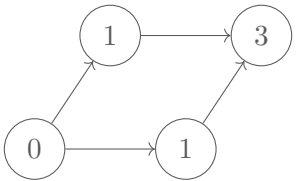
Model with DAG

Actually it makes much sense to model the task using DAG, since each game can be represented by an edge $u \to v$ when player u defeats player v.

We can further define the *rank* of node v be the number of nodes u which **transitively** defeats node v, that is, there exists a chain

$$u \to p_1 \to \cdots \to p_k \to v$$

For some $k \geq 0$.



Model with DAG

Observation 4

If the rank of a node is $\geq R$, then we can safely ignore it.

Implementation-wise, notice that you cannot naively compute all the ranks since that would be $O(NM)$ per round, M the number of edges.

Therefore, you should stop DFS once the rank is at least R, so the time complexity drops to $O(RN)$ per round, which is more than acceptable.

Now, our target is to construct good rounds, such that all the nodes has rank $\geq R$ except the top R nodes, as soon as possible.

Solution 6.1 (Maintain DAG, Random Pair)

The simplest solution is just randomly form pairs, repeat until there is an unique node of each of the rank 0 to $R-1$.

	Sub 1	Sub 2	Total
K	9	87	
Score	7	18.3	25.3

Solution 6.2 (Maintain DAG, Same Level)

How can we make sure the rounds has "contribution" over increasing the ranks? Turns out there is an interesting observation:

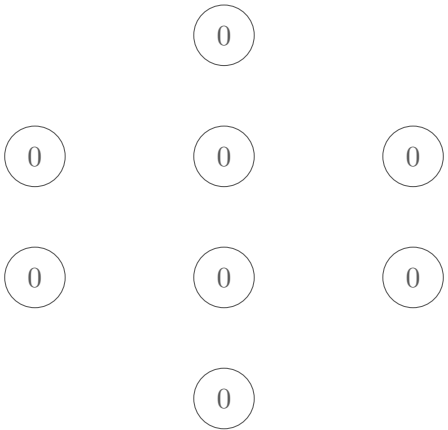
Observation 5

Two nodes of the same rank does not **transitively** defeat each other.

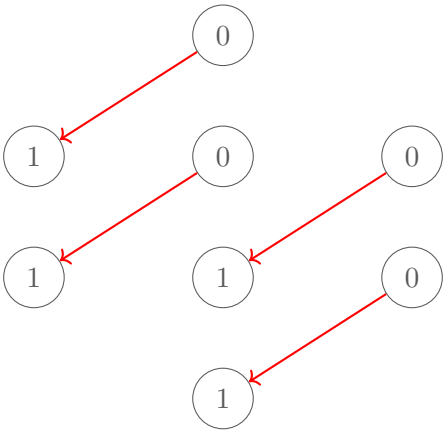
Thus, by comparing two nodes of the same rank, one of the ranks increases by at least one! Prioritising this, the performance is quite good.

	Sub 1	Sub 2	Total
K	5	22	
Score	25	61	86

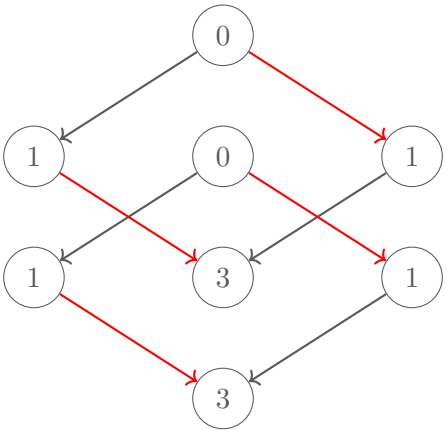
Computing Rank Example



Computing Rank Example



Computing Rank Example



Computing Rank Example

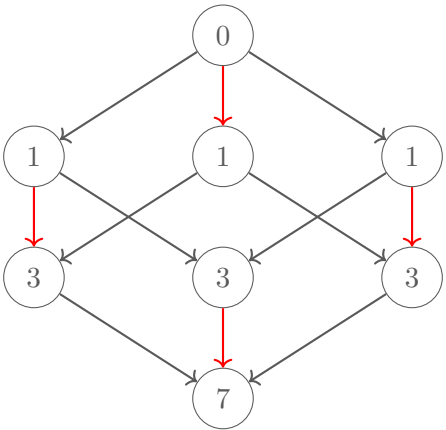


Table of Contents

Tree vs DAG

What if we maintain rooted trees (heaps) instead of DAG, such that A beats B if A is an ancestor of B?

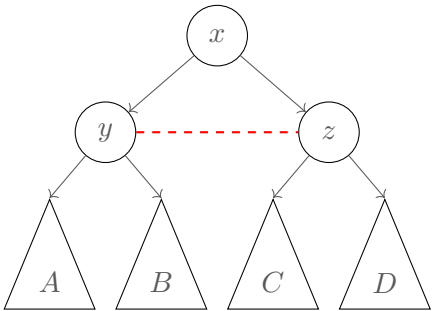
Image a supernode that beats all players. We want the ending tree to have an unqiue chain of $R + 1$ nodes starting from the root.

Idea: Chains conveys more information due to transitivity, so it is favourable to make the children count of nodes as small as possible.

How to do so? For a node with 2 children, we can compare them in a round. This allows us to move one of them as a child of the other, hence reducing the number of children!

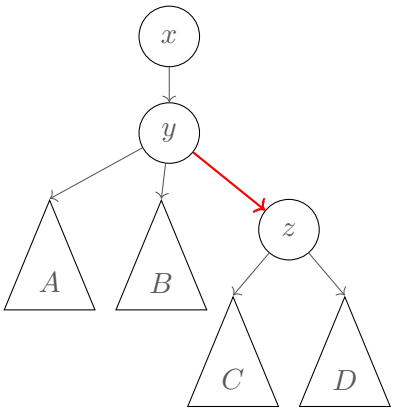
Modifying Binary Heap

Binary (Max) Heap: all nodes has ≤ 2 children, parent $\geq$ child for all nodes
Compare node y and node z:



Modifying Binary Heap

If node y is larger than node z, then the parent of node z is assigned to be node y.

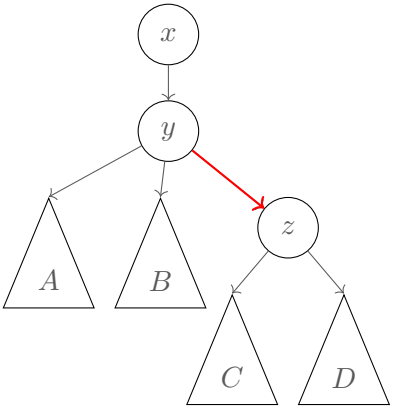


Modify Binary Heap in Parallel

The heap property (parent $\geq$ child) is maintained. However, node y now has 3 children, making the heap no longer binary. How to fix this?

Solution: Carry out comparison **in parallel** between all pairs of nodes with the same parent.

Either A becomes subtree of B or B becomes subtree of A - either case now node y only has two children!



Example

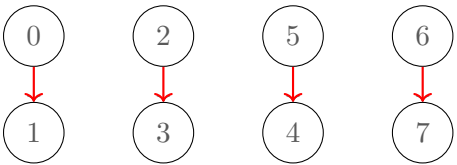
`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`

Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`

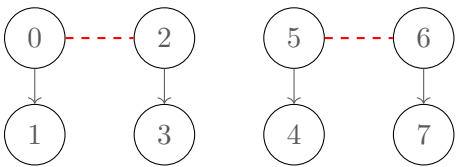
Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



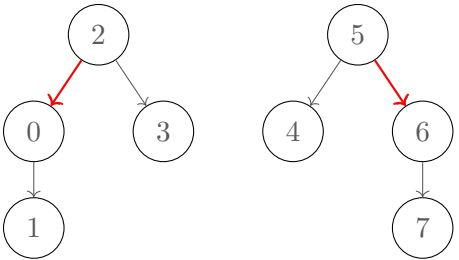
Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



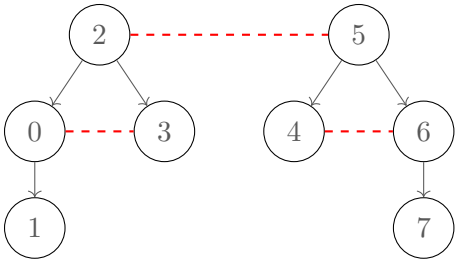
Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



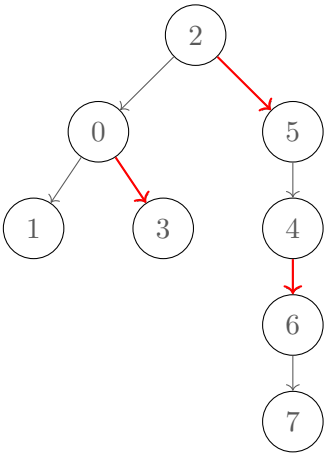
Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



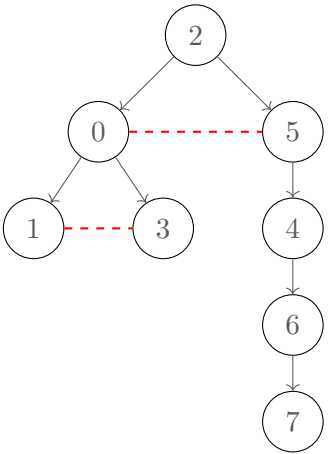
Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



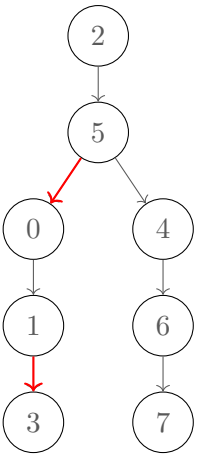
Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



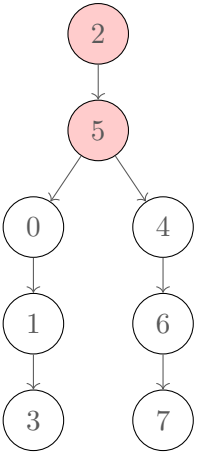
Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



Example

`N = 8, R = 2, A = [2, 5, 4, 0, 6, 7, 1, 3]`



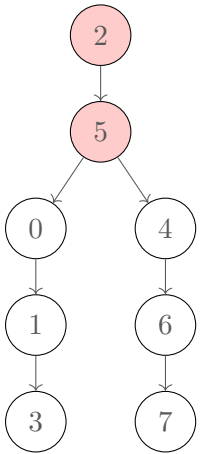
Solution 7 (Evolving Binary Heap)

It takes $\log_2 N$ rounds to form a single binary heap, which we get the maximum for free. In each subsequent round, we can find the next maximum.

Therefore, the total number of rounds is

$$\log_2 N + (R - 1)$$

	Sub 1	Sub 2	Total
K	4	19	
Score	30	70	100



Solution	Subtask 1		Subtask 2		Total
	K	Score	K	Score	Score
group-of-2R	6	20	135	13.5	33.5
group-of-R	5	25	79	26.5	51.5
binary-search-repeat	6	20	96	17.4	37.4
binary-search-reduction	5	25	64	34	59
binary-search-toposort	5	25	42	45	70
bitonic-merge-sort	6	20	78	27	47
bitonic-group	5	25	42	45	70
dag-random	9	7	87	18.3	25.3
dag-same-level	5	25	22	61	86
evolving-binary-heap	4	30	19	70	100

Takeaways

- Always try to attack the task using different perspectives. Although not all paths lead to full score, you can still get valuable insights via trying to use say sort, search, DAG and trees to solve the task.

- Do try to solve for small case; However, try your best to make your solution as general as possible, by using subroutines! Hard-coding for the small case won't get you very far.