FENWICK #1: Properties

Which of the following is true about a Fenwick Tree?

- 0 It requires O(N) additional space for a 1D BIT.
- 1 Useful to answer queries of the form [1, r].
- 2 Segment Tree is a more powerful data structure compared to Fenwick Tree.
- 3 A Fenwick Tree has a very good running time as it uses fast bit manipulation operations.
- 4 A Fenwick Tree can be coded in fewer lines of code compared to a Segment Tree.
- 5 All of these

FENWICK #2: Time Complexities

For range sum queries, what is the time complexity of build, update, and query using a Fenwick Tree respectively?

- 0 O(N), O(logN), O(logN)
- 1 O(NlogN), O(logN), O(1)
- 2 O(NlogN), O(logN), O(logN)
- 3 O(N), O(1), O(logN)

FENWICK #3: LSOne Function

Let us define a function:

CopyEdit

```
LSOne(x) = (x & -x)
```

What does LSOne(x) represent?

- [0] This value has no such significance.
- [1] It is a number with all bits of x flipped.
- [2] It is a number with all bits of x turned off except the rightmost (least significant) set bit of x.
- [3] It is a number with all bits of x turned off except the leftmost (most significant) set bit of x.

FENWICK #4: Parent Node

Define parent(x): the smallest index `j > x` such that BIT[j] includes A[x]. What is the correct formula for parent(x)?

- [0] parent(x) = x + LSOne(x)
- [1] parent(x) = x ^ (1 << 2)
- [2] parent(x) = x - LSOne(x)
- [3] parent(x) = x & (1 << 2)

FENWICK #5: Building in O(N)

Given P[i] = A[1] + A[2] + ... + A[i], what is the correct formula to build BIT in O(N)?

- 0 It is not possible to build the BIT in O(N) time.
- 1 BIT[x] = P[x] - P[x + LSOne(x)]
- 2 BIT[x] = P[x] - P[x - LSOne(x)]
- 3 BIT[x] = P[N] - P[x ^ LSOne(x)]

FENWICK #6: Order Statistic Tree Ops

Which operations are supported by an order statistic tree (OST)?

- 0 Both operations are supported with O(logN) complexity.
- 1 Only one of these operations is supported with O(logN).
- 2 Find the ith smallest element in the tree.
- 3 Find the rank of element x in the tree.

{An order statistic tree is a specialized binary search tree that efficiently supports two additional operations beyond basic BST functionality: finding the k-th smallest element (Select) and finding the rank of a given element (Rank)}

FENWICK #7: BIT as OST

Using BIT to simulate an OST with freq[i] = count of i:

- 0 To insert i: update(i, +1)
- 1 To delete i: update(i, -1)
- 2 To find rank(i): sum(1, i - 1)
- 3 All of these

FENWICK #8: kth Smallest Element

Time complexity to find the kth smallest element using BIT with binary search?

- 0️⃣O(logN)
- 1️⃣O(log(N*N))
- 2️⃣O(NlogN)
- 3️⃣O(logNlogN)

FENWICK #9: Handling Large Values

How can we counter the limitation that BIT requires values in range [1, N]?

- 0️⃣This limitation can't be countered and AVL trees are preferred.
- 1️⃣Coordinate Compression.
- 2️⃣DP on Trees.
- 3️⃣Segment Tree is the correct option for an order-statistic tree.

FENWICK #10: Range Update Complexity

Time complexity to perform a range update (add x to [L, R]) using BIT on array of size N?

- 0️⃣O(NlogN)
- 1️⃣O(logN)
- 2️⃣O(R - L)
- 3️⃣O(log(R - L))

FENWICK #11: Update It (SPOJ) Logic

Time complexity of this algorithm (range update + prefix sum + point query):

- 0️⃣ Incorrect algorithm
- 1️⃣ $O(N^2)$
- 2️⃣ O(u + q + N)
- 3️⃣ O(u + N)

FENWICK #12: 2D BIT Query Time

For querying sum over subrectangle (x1, y1) to (x2, y2), what is optimal query time using 2D BIT?

- 0️⃣ O(logN + logM)
- 1️⃣ O(logNM)
- 2️⃣ O(logN)
- 3️⃣ O(logM)

✅ Answer Key

Q#	Answer
1	5️⃣ All of these
2	0️⃣ O(N), O(logN), O(logN)
3	2️⃣ Rightmost set bit
4	0️⃣ x + LSOne(x)
5	2️⃣ P[x] - P[x - LSOne(x)]
6	0️⃣ Both are supported

7 3 All of these

8 3 O(logNlogN)

9 1 Coordinate Compression

10 1 O(logN)

11 2 O(u + q + N)

12 0 O(logN + logM)