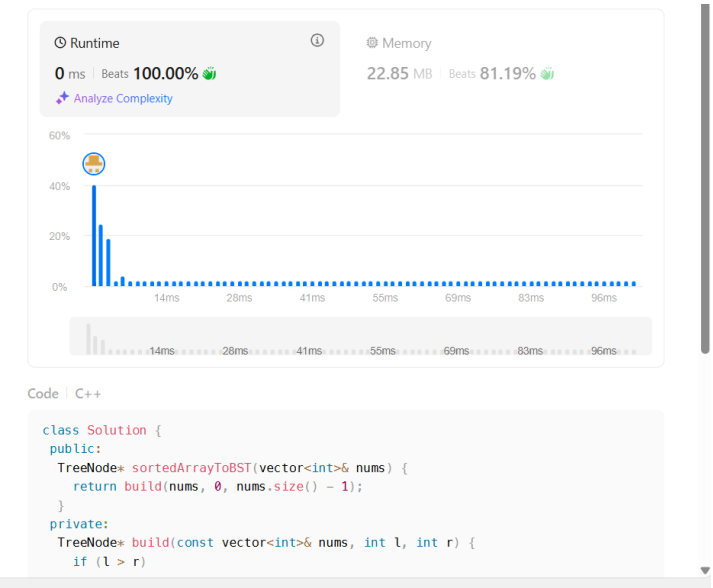
Name: Prince Sharma
UID : 22BCS14846
section: IOT-605 (B)

1.Convert Sorted Array to Binary Search Tree

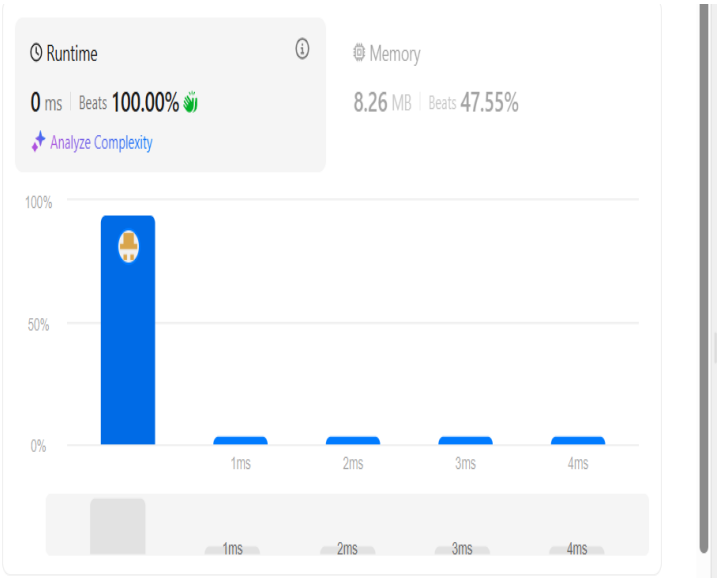
```
class Solution {
 public:
  TreeNode* sortedArrayToBST(vector<int>& nums) {
    return build(nums, 0, nums.size() - 1);
  }

 private:
  TreeNode* build(const vector<int>& nums, int l, int r) {
    if (l > r)
      return nullptr;
    const int m = (l + r) / 2;
    return new TreeNode(nums[m], build(nums, l, m - 1), build(nums, m + 1, r));
  }
};
```



2. Number of 1 Bits

```
class Solution {
 public:
  int hammingWeight(uint32_t n) {
    int ans = 0;
    for (int i = 0; i < 32; ++i)
      if ((n >> i) & 1)
        ++ans;
    return ans;
  }
};
```



Code | C++

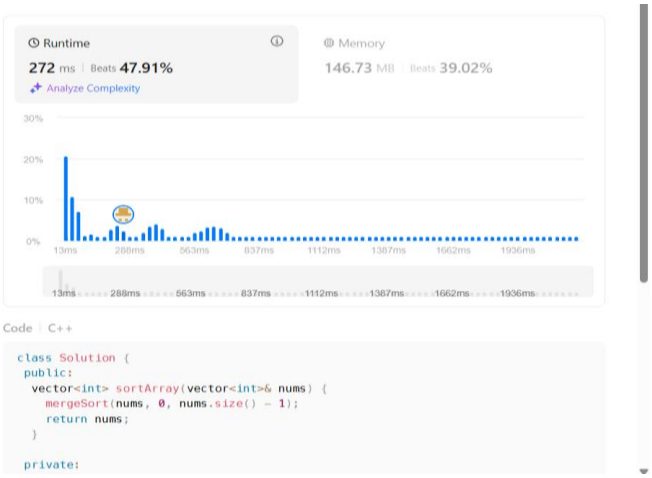
```
class Solution {
 public:
  int hammingWeight(uint32_t n) {
    int ans = 0;

    for (int i = 0; i < 32; ++i)
      if ((n >> i) & 1)
```

3. Sort an Array

```
class Solution {
 public:
  vector<int> sortArray(vector<int>& nums) {
    mergeSort(nums, 0, nums.size() - 1);
    return nums;
  }

 private:
  void mergeSort(vector<int>& nums, int l, int r) {
    if (l >= r)
      return;
    const int m = (l + r) / 2;
    mergeSort(nums, l, m);
    mergeSort(nums, m + 1, r);
    merge(nums, l, m, r);
  }
  void merge(vector<int>& nums, int l, int m, int r) {
    vector<int> sorted(r - l + 1);
    int k = 0;
    int i = l;
    int j = m + 1;
    while (i <= m && j <= r)
      if (nums[i] < nums[j])
        sorted[k++] = nums[i++];
      else
        sorted[k++] = nums[j++];
    while (i <= m)
      sorted[k++] = nums[i++];
    while (j <= r)
      sorted[k++] = nums[j++];
    copy(sorted.begin(), sorted.end(), nums.begin() + l);
  }
};
```



4. Maximum Subarray

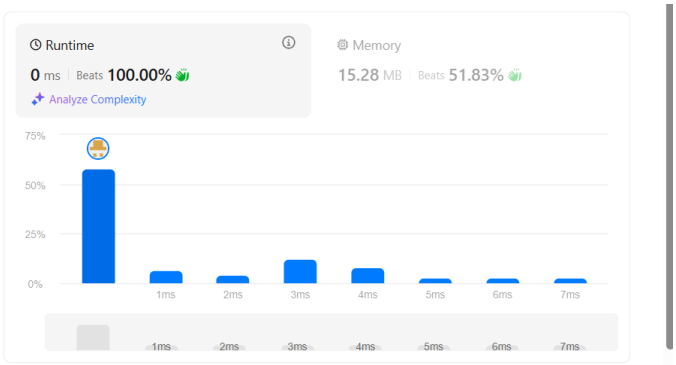
```
class Solution {
 public:
  int maxSubArray(vector<int>& nums) {
    vector<int> dp(nums.size());
    dp[0] = nums[0];
    for (int i = 1; i < nums.size(); ++i)
      dp[i] = max(nums[i], dp[i - 1] + nums[i]);
    return ranges::max(dp);
  }
};
```

5. Beautiful Array

```
class Solution {
 public:
  vector<int> beautifulArray(int n) {
    vector<int> arr(n);
    iota(arr.begin(), arr.end(), 1);
    divide(arr, 0, n - 1, 1);
    return arr;
  }
 private:
  void divide(vector<int>& arr, int l, int r, int mask) {
    if (l >= r)
      return;
    const int m = partition(arr, l, r, mask);
    divide(arr, l, m, mask << 1);
    divide(arr, m + 1, r, mask << 1);
  }
  int partition(vector<int>& arr, int l, int r, int mask) {
    int nextSwapped = l;
    for (int i = l; i <= r; ++i)
      if (arr[i] & mask)
        swap(arr[i], arr[nextSwapped++]);
    return nextSwapped - 1;
  }
};
```

6. Super Pow

```
class Solution {
 public:
  int superPow(int a, vector<int>& b) {
    int ans = 1;
    a %= kMod;
    for (const int i : b)
      ans = modPow(ans, 10) * modPow(a, i) % kMod;
    return ans;
  }
 private:
  static constexpr int kMod = 1337;
  long modPow(long x, long n) {
    if (n == 0)
      return 1;
    if (n % 2 == 1)
      return x * modPow(x % kMod, (n - 1)) % kMod;
    return modPow(x * x % kMod, (n / 2)) % kMod;
  }
};
```



Code | C++

```
class Solution {
 public:
  int superPow(int a, vector<int>& b) {
    int ans = 1;
    a %= kMod;
    for (const int i : b)
      ans = modPow(ans, 10) * modPow(a, i) % kMod;
    return ans;
```

7. The Skyline Problem

```
class Solution {
 public:
  vector<vector<int>> getSkyline(const vector<vector<int>>& buildings) {
    const int n = buildings.size();
    if (n == 0)
      return {};
    if (n == 1) {
      const int left = buildings[0][0];
      const int right = buildings[0][1];
      const int height = buildings[0][2];
      return {{left, height}, {right, 0}};
    }

    const vector<vector<int>> left =
        getSkyline({buildings.begin(), buildings.begin() + n / 2});
    const vector<vector<int>> right =
        getSkyline({buildings.begin() + n / 2, buildings.end()});
    return merge(left, right);
  }

 private:
  vector<vector<int>> merge(const vector<vector<int>>& left,
                            const vector<vector<int>>& right) {
    vector<vector<int>> ans;
    int i = 0;
    int j = 0;
    int leftY = 0;
    int rightY = 0;

    while (i < left.size() && j < right.size())
      if (left[i][0] < right[j][0]) {
        leftY = left[i][1];
        addPoint(ans, left[i][0], max(left[i++][1], rightY));
      } else {
        rightY = right[j][1];
        addPoint(ans, right[j][0], max(right[j++][1], leftY));
      }

    while (i < left.size())
```

```
    addPoint(ans, left[i][0], left[i++][1]);

  while (j < right.size())
    addPoint(ans, right[j][0], right[j++][1]);

  return ans;
 }

 void addPoint(vector<vector<int>>& ans, int x, int y) {
  if (!ans.empty() && ans.back()[0] == x) {
    ans.back()[1] = y;
    return;
  }
  if (!ans.empty() && ans.back()[1] == y)
    return;
  ans.push_back({x, y});
 }
};
```