Heaps + Tries Followup

① Question: Running Median

Median: Middle Element of Sorted Array

A = 9 8 5 6

Sort(A) = 5 6 8 9

0 1 2 3

N = 6

$\left[\frac{n}{2}, \frac{n}{2}+1\right]$

$3 \leq 4$

$\left[\frac{n}{2}-1, \frac{n}{2}\right]$

$\frac{6+8}{2} = 7$

Odd: $A\left[\frac{N}{2}\right]$

Even: $\frac{A\left[\frac{N}{2}-1\right] + A\left[\frac{N}{2}\right]}{2}$

7 8 9

$\frac{8+9}{2} = 8.5$

A = 9 8 7 3 6 4

Median = 9 8.5 8 7

A: 7 8 9 [Sorted Insert]

Brute Force:

n. 1) Sort the array when you get a new element → O(nlogn)

2) Return Middle Element.

T.C: $O(n^2 \log n)$

7 8 9

[3 6 7 8 9
3 4 6 7 8]

Approach2:

1) Sorted Insert: O(n) ✓

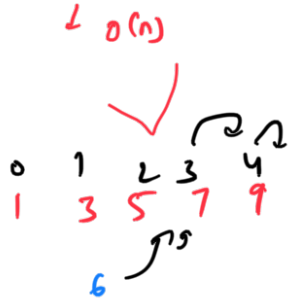
2) Return middle element.

Linear Search (4)

[1 3 4 5 6 7]

T-C : $O(n^2)$

$O(n)$

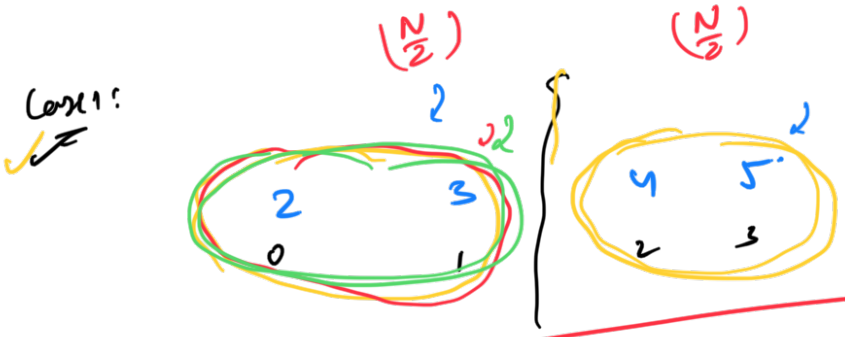


Approaches: Heaps

Observations

1) Only Interested in the middle element
2) Finding median is $O(1)$ if array was sorted
3) To maintain array in sorted order, $O(n)$
4) Do we really need array in sorted order?
   → Give middle element as last as possible

$\frac{3+4}{2} = 3.5$



Median: Max of Left Half + Min of Right Half

Left Right

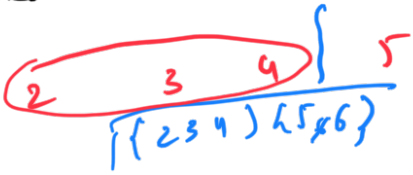
2 3 | 4 5 6

Min of Right Half

Median =

Maximum of Left Half.

N = 5



{2,3} {4,5,6}

Idea :

- Max of Left Half → Max Heap
- Min Half → Min Heap

Min of Right

4) All Elements of Left Half should be less than all elements of Right Half

size(maxHeap) − size(minHeap) = {0, 1}

size(maxHeap) → Left Half

size(minHeap) → Right half.

1) Even => 0
2) odd => 1

Left < Right

Dry Run

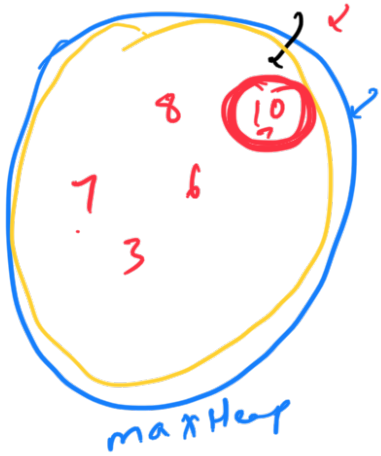
A: 9 8 7 3 6 20 30

Median: 9 8.5 8 7.5 7 7.5 8

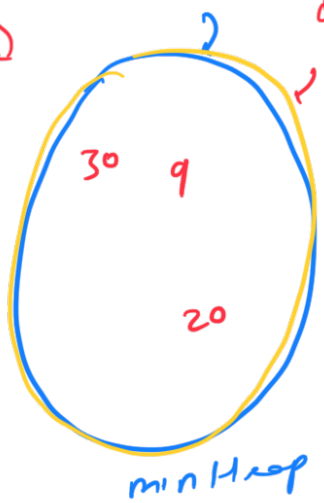
Left − Right = 2

else < maxHeap.top() MAX HEAP

else MIN HEAP



maxHeap



minHeap

4 − 32!

MaxHeap = 3
MinHeap = 4
5 − 4 = {−1, 1}
10, 14

Even = (Max(Left) + Min(Right)) / 2

Odd. Max(Left)

T.C : O(1) + O(log n) + O(1) + O(log n) + O(log n)

T.C: [illegible]

N operations => O(N log N)

Map of Left Hall

[L - R] : {0, 1}

Question: Merge K - sorted Linked Lists

[Frequently asked in interviews]

n . k

LL1 => (1) → (5) → (9)

LL2 => (2) → (4)

LL3 => (3) → (7) → (14) → (20)

x_1, x_2, x_3

LL => (1) → (2) → (3) → (4) → (5) → (7) → (9) → (14) → (20)

$x_1 + x_2 + x_3$

Brute force:

→ Take all elements into array => O(n.k)

→ Sort it O(n.k log(n.k))

→ Create a new Linked List -> O(n.k)

→ K Linked Lists => Each of size N

T.C: O(n.k log(n.k))

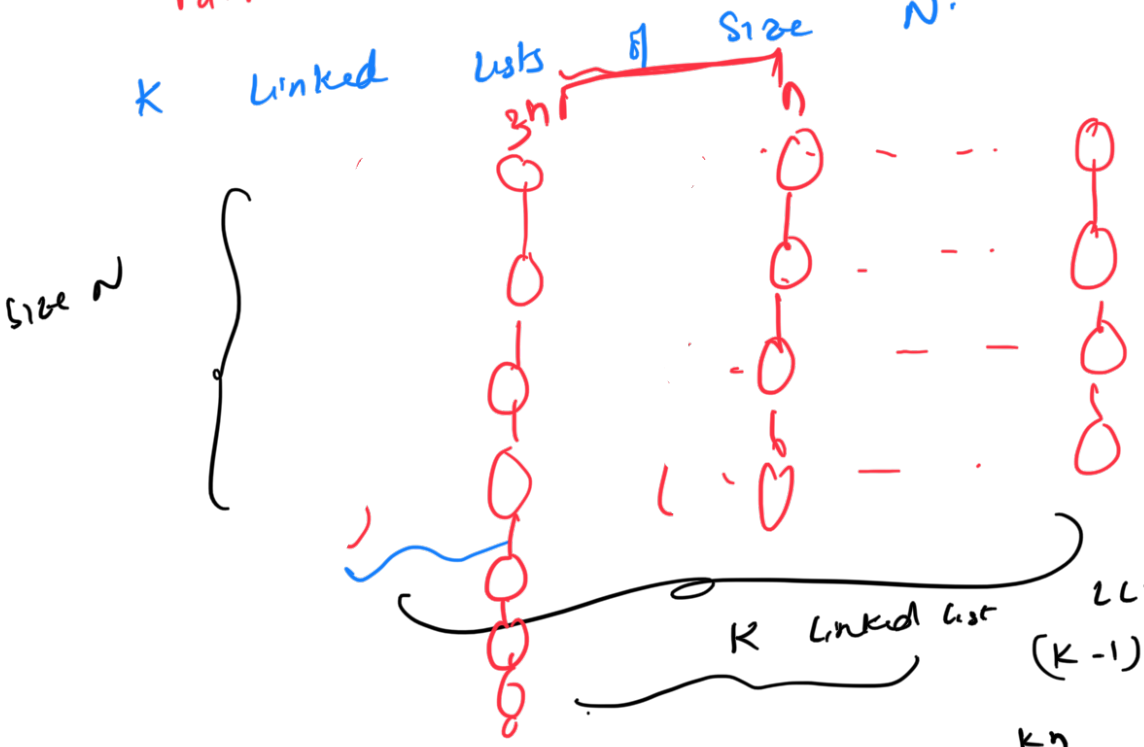
S.C: O(n.k)

LL1 : m

Approach 2 :

Pair wise Merge

K Linked Lists of Size N.



LL2 : n

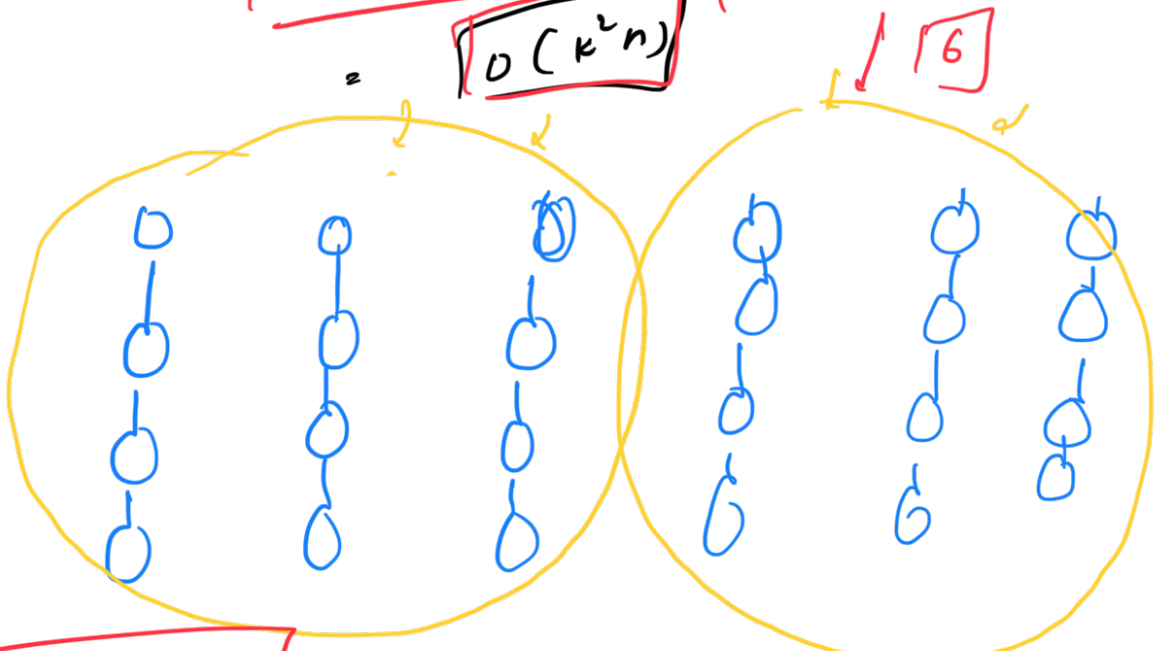
O(m+n)

T.C : $2n + 3n + 4n + \dots + kn$

$= n(2+3+4+\dots+k)$

$= n\left(\frac{K(K+1)}{2}\right)$

$= O(K^2 n)$



Merge k Lists ( )

Approaches:

- Insert all elements into a heap

Sol 1: Implement our own heap

1) Size = array N·K
2) Build a min Heap => O(N·K)
3) Extract min ele and add it to Linked List => log(NK)

T.C: $N \cdot K \log(N \cdot K)$

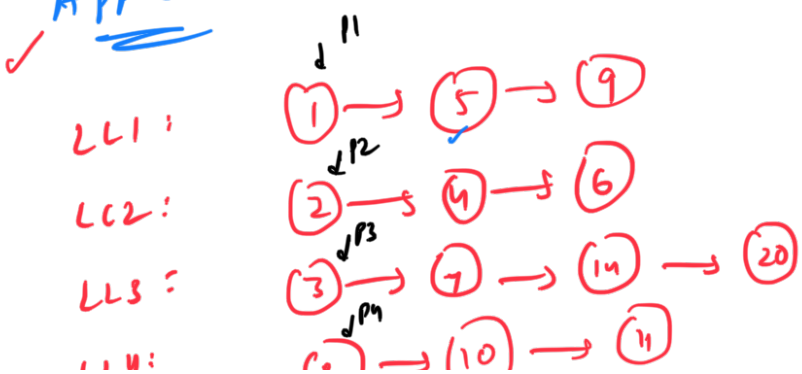
Sol 2: Priority Queue

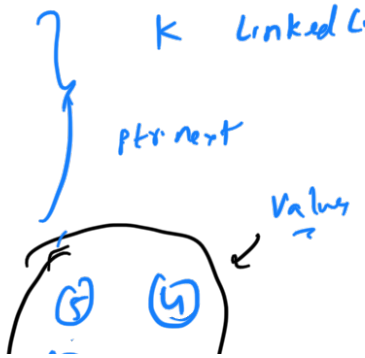
1) Build a min Heap: $(N \cdot K) \log(N \cdot K)$
   - $(N \cdot K)$ Insert operations in a priority queue
   - $\log(N \cdot K) \times N \cdot K$

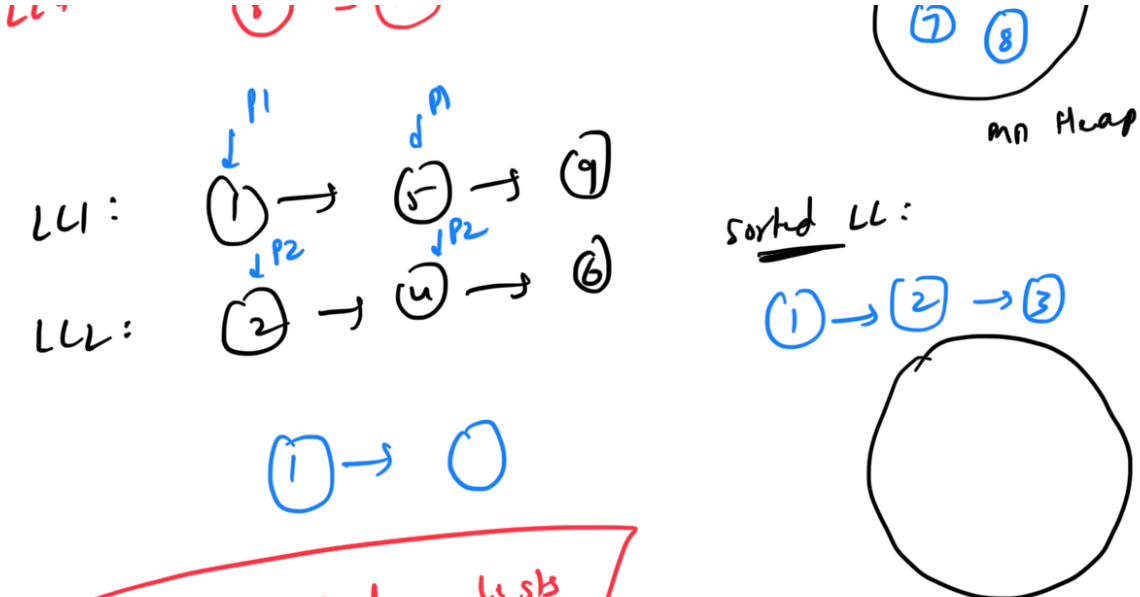
T.C: $O(N \cdot K \log(N \cdot K))$

S.C: $O(N \cdot K)$

Approaches:







K Linked Lists

→ K pointers

→ Minimum of all these K Pointers → O(K)

Min Heap

```
struct Node {
    Node* next;
    int val;
};
```

Custom Comparator

T.C :

Size of Heap

1) Appending min element to LL [O(1)]

2) Extract the min : O(log K)

3) Insert new element : O(log K)

1 iteration : O(log K)

T.C : O(n.k ^ log)

S.C : O(k)

Question : N Max Pair Combinations

N

	0	1	2	3
A =	3	1	4	0
B =	9	8	1	13

→ Return the first N pairs in order of decreasing sum

(Ai, Bi)

Ans : (4, 13), (3, 15), (1, 13), (4, 9)

N pairs

Brute Force:

→ Consider all pairs => O(n²)

→ Sort all the pairs => O(n² log n²)

→ Return the first N pairs =>

T.C : O(n² log n²)

S.C : O(n²)

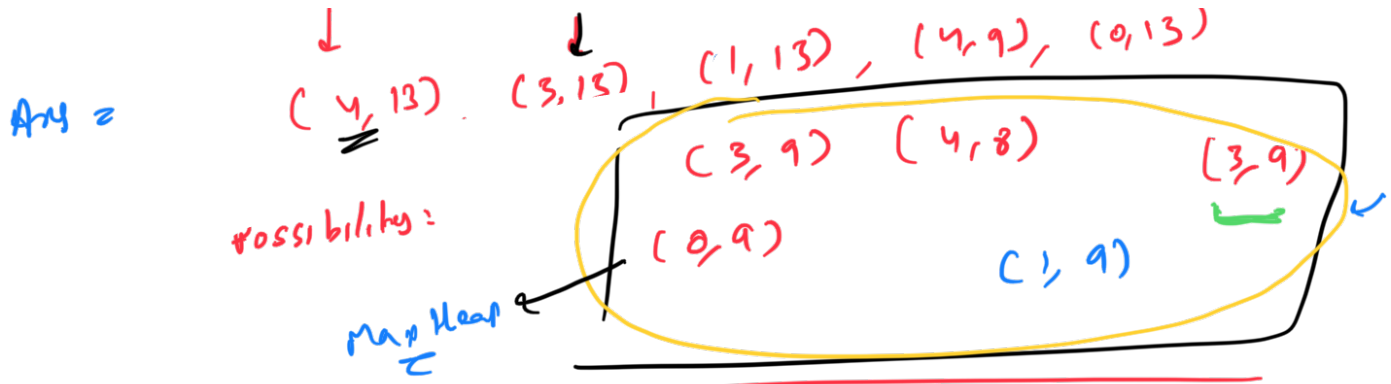
(4, 15)

(3, 15) (4, 9)

(3, 15) (4, 9)

Approach :

	0	1	2	3
A =	0	1	3	4
B =	1	8	9	13



(A[i], B[j]) => (A[i-1], B[j]) and (A[i], B[j-1])

→ class Node {
int i1;
int i2;
int sum;
}

i1, i2

a[i1] + B[i2]

A[5] + B[5]

A =

0	1	2	3
1	2	3	4

B =

0	1	2	3
1	2	5	6

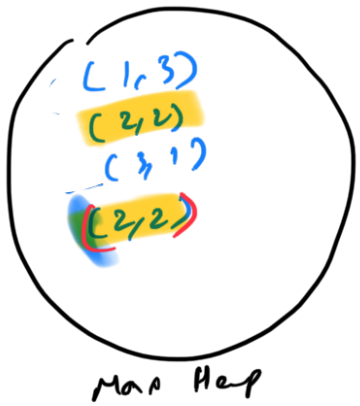
Ans = (4,6), (3,6), (4,5)

ind = (3,3) (2,3) (3,2)

(3,2)

(3,6)
(4,5)

(4,3)



(Hashset) <
pair<int, int>

(i1, i2)

Compare(Node n1, Node n2)?

N1 => A[n1.i1] + B[n1.i2]

N2 => A[n2.i1] + B[n2.i2]

$N_1 \geq N_2$

Max - Heap

T.C:
1) Sorting the 2 arrays : $O(n \log n)$
2) Extracting 2 Max Element : $O(\log n)$ } N times
3) 2 Insert operations : $O(\log n)$

T.C: $O(N \log N)$

$(i, j) \Rightarrow (i-1, j) \quad (i, j-1)$

Question: No of subarrays with XOR less than B

$N(N+1)/2$, $N = 4$

$B = 6$

	0	1	2	3
A =	9	4	3	2
	1001	0100	0011	0010

size 1 = [4] [3] [2] => 3

size 2 = [3, 2] => 1

size 3 = [4, 3, 2] => 1

size 4 =

3 + 1 + 1 = 5

1001
0100
0011
1110 = 14

0011
0010
0001 = 1

0100
0011
0010
0101 = 5

Brute force :

Consider all subarrays : $\frac{n(n+1)}{2}$ $[l, r]$

T.C to find XOR of subarray $[l, r]$: $O(n)$

T.C : $O(n^3)$

Approach :

Prefix XOR array :

T.C : $O(n^2 \times 1) : O(n^2)$

Subarray Sum/XOR

$[l, r] = ?$ (4)

Approach :

→ Prefix Sum array

A = [9, 4, 2, 3, 2]

prefixXOR = [0, 9, 13, 14, 12]

0000, 1001, 1101, 1110, 1100

(x, y)

→ (x, y)

$(9 \oplus 4 \oplus 3 \oplus 2) \oplus 9 = 4 \oplus 3 \oplus 2$

$(9 \oplus 4 \oplus 3) \oplus 0 = 9 \oplus 4 \oplus 3$

Tries

30 :

→ Find no. of pairs (x, y) from the prefix XOR array such that $x \oplus y < B$

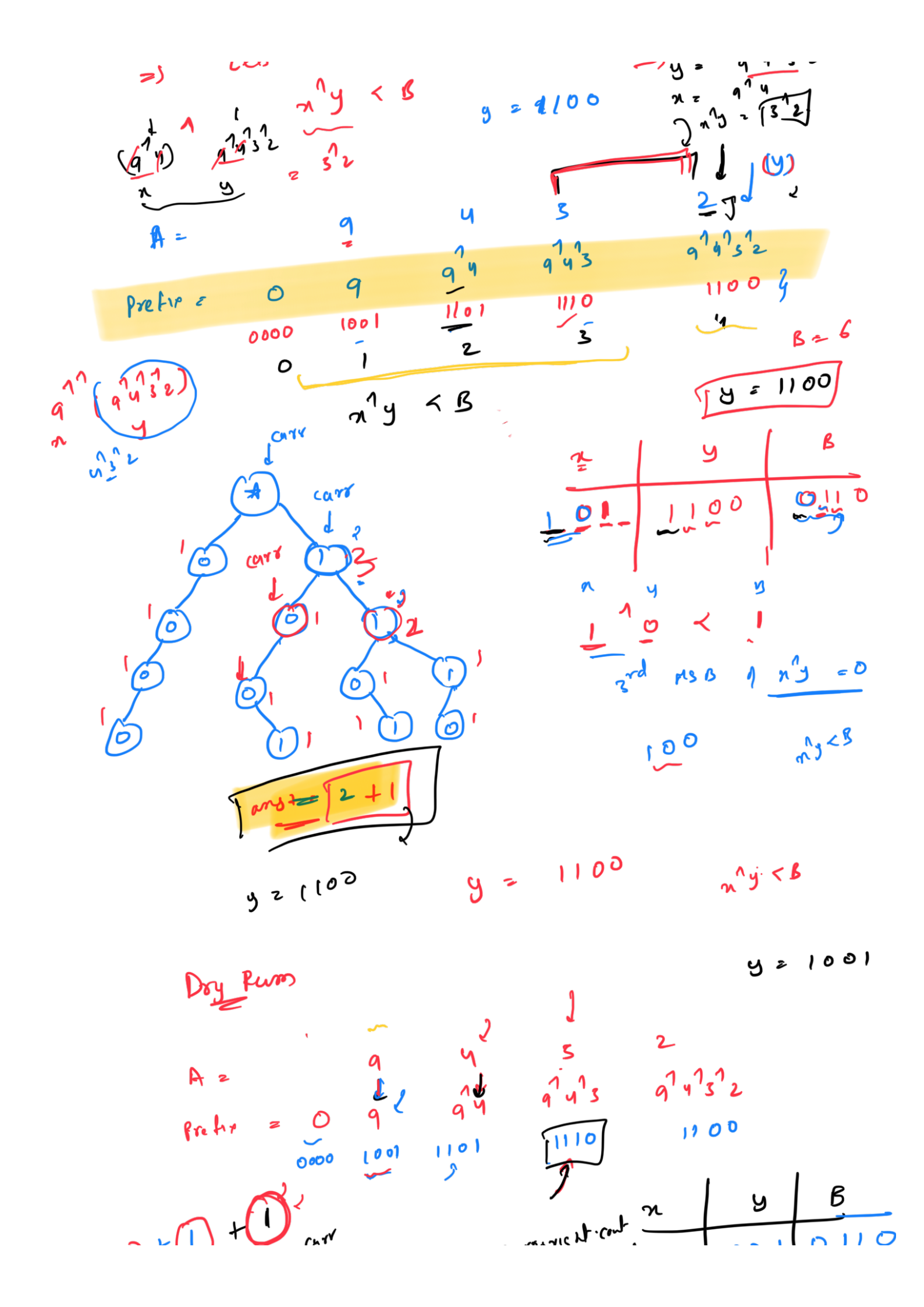
$x \oplus y = B$

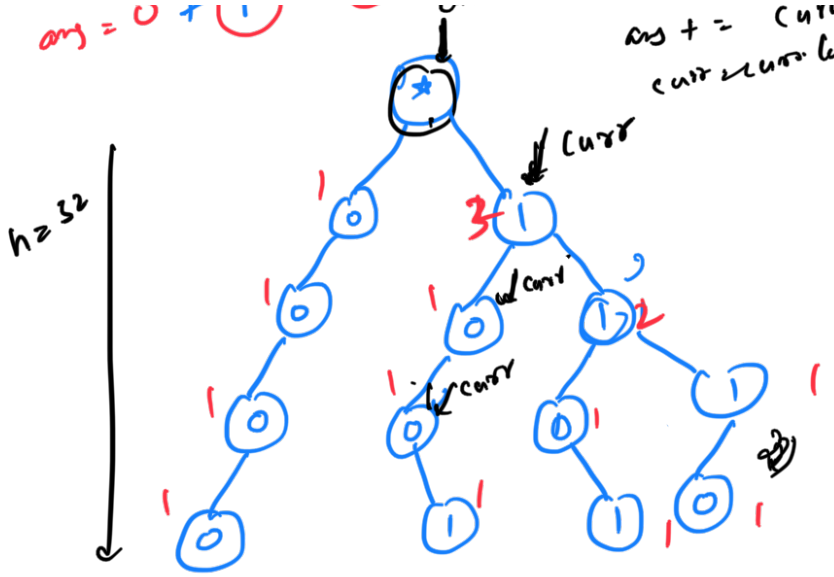
$\Rightarrow$ $x \oplus y < B$

Find no. of x's such that

Let fix y.

$9 \oplus 4 \oplus 3 \oplus 2$


y = 1100
B = 6
y = 1100
y = 1100
y = 1001
Dry Run
ans = 2 + 1
carry



1	1 0 0 1	
1 0 1	1 1 0 1	0 1 1 0
1 0 0 0	1 1 1 0	0 1 1 0
	1 1 0 0	0 1 1 0

$0 \wedge 0 < 0$

$n \wedge y = 0$

Lecture Notes ⇒ 1 hour ?

T.C: $O(32) \times (n)$

$= O(n \times 32) = O(n)$

$\log(\max(\text{pref}))$

S.C: $O(n \cdot 1) = O(n)$

$n \wedge y = B$

$B \wedge y$

$n \wedge y < B$

No. of subarray with sum $= B$?

No. of subarrays with XOR less than B

```
bool ithBit(int x, int i){
     if(x && (1 << i)) return 1;
     return false;
}

int getSubArraysEndingAtI(root, Y, B){
     temp = root;
     for(i = 31; i >= 0 && temp != NULL; i--){
          B_bit = ithBit(B, i);
          Y_bit = ithBit(Y, i);
          if(B_bit == 0){
               if(Y_bit == 0){      x_bit = 0
                    temp = temp->children[0];
               else       // X_bit = 1
                    temp = temp->children[1];
               }
          }
          else{      // B_bit = 1
               if(Y_bit == 1){      X_bit = 0
                    if(temp->children[1] != NULL)
                         ans += temp->children[1]->count;
                    temp = temp->children[0];
               }
               else{           X_bit = 1
                    if(temp->children[0] != NULL)
                         ans += temp->children[0]->count;
                    temp = temp->children[1];
               }
          }
     }
     return ans;
}
```

```
int countSubarraysXORLessThanB(int arr[], int n, int B){
     root = new trieNode;
     // Generate the Prefix XOR array
     insert(root, pre[0]); // Insert 0 into the trie

     ans = 0;
     for(int i = 1; i <= n; i++){
          ans += getSubArraysEndingAtI(root, pre[i], B);
          insert(root, pre[i]);
     }
     return ans;
}
```

Running Median

```
priority_queue<int, vector<int>, greater<int>> minHeap;
priority_queue<int> maxHeap;

// Median after the element x has been added to the stream
int median(int x){
      if(!maxHeap.size() || x <= maxHeap.top())
            maxHeap.push(x);
      else
            minHeap.push(x);

      if(maxHeap.size() - minHeap.size() > 1){
            temp = maxHeap.top();
            maxHeap.pop();
            minHeap.push(temp);
      }
      else if(maxHeap.size() - minHeap.size() < 0){
            temp = minHeap.top();
            minHeap.pop();
            maxHeap.push(temp);
      }

      if(maxHeap.size() == minHeap.size())
            return maxHeap.top() + minHeap.top() / 2;
      else
            return maxHeap.top();

}
```

Merge K sorted linked lists

```
struct Node{
      int data;
      Node* next;
}

struct compare{
      bool operator() (Node* n1, Node* n2){
            // Min heap
            return n1->data > n2->data;
      }
}

head = tail = NULL    // Global variables.

mergeK(vector<Node*> A){   // k pointers are given as input
      priority_queue<Node*, vector<Node*>, compare> pq;

      for(int i = 0; i <A.size(); i++){
            if(A[i] != NULL) pq.push(A[i]);
      }
      while(!pq.empty()){
            minmNode = pq.top();
            insertAtEnd(minmNode);
            pq.pop();

            if(minmNode->next)
                  pq.push(minmNode->next);
      }
}
```

N Max pair combinations comparator

```
struct Node{
      int i1, i2;
}

struct compare{
      bool operator() (Node n1, Node n2){
            // Max heap
            return A[n1.i1] + B[n1.i2] < A[n2.i1] + B[n2.i2];
      }
}
```