Heap [min pq]

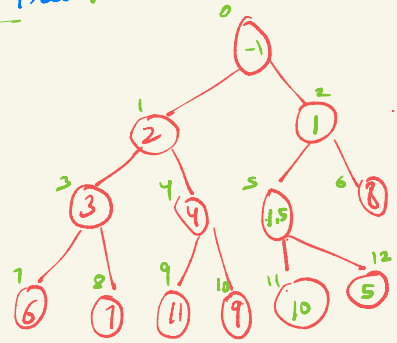
Complete Binary Tree (CBT)

Heap order property

why?
what?
how?

-1

upHeapify → every parent will be smaller than child



{-1, 2, 1, 3, 4, 1.5, 8, 6, 7, 11, 9, 10, 5}

pi = 4

lci = 2*pi+1;

rci = 2*pi+2;

lci = 9

rci = 10

pi = 12-1/2 ⇒ 5

pi = 5-1/2 ⇒ 2

pi = 2-1/2 ⇒ 0

swap(ci, pi)

ci

{pi = ci-1/2}

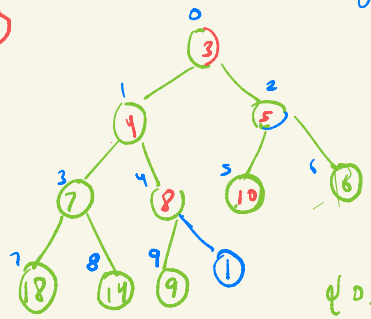
12

```
static class Heap {
    ArrayList<Integer> data;

    public Heap(){
        data = new ArrayList<>();
    }

    public void swap(int i, int j){
        int valAtI = data.get(i);
        int valAtJ = data.get(j);

        data.set(i,valAtJ);
        data.set(j,valAtI);
    }

    public void upHeapify(int ci){
        int pi = (ci-1)/2;

        if(data.get(pi) > data.get(ci)){
            swap(ci,pi);
            upHeapify(pi);
        }
    }
    // add
    public void add(int val){
        data.add(val);

        upHeapify(data.size()-1);
    }
```

data ⇒ {3, 4, 5, 7, 8, 10, 6, 18, 14, 9}

O(n)

-1



down Heapify

{0, 1} pi, minCi

pi = 0

lci = 2*pi+1 = 1

rci = 2*pi+2 = 2

pi = 1

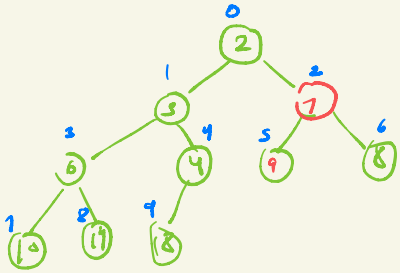
lci = 3

rci = 4

{1, 4}

```java
private void downHeapify(int pi){
    int lci = 2*pi + 1;
    int rci = 2*pi + 2;

    int mini = pi;

    if(lci < data.size() &&  data.get(lci) < data.get(mini)){
        mini = lci;
    }

    if(rci < data.size() && data.get(rci) < data.get(mini)){
        mini = rci;
    }

    if(mini != pi){
        swap(pi,mini);
        downHeapify(mini);
    }
}
// remove
public int remove(){
    if(data.size()==0){
        System.out.println(x: "There is no element in heap");
        return -1;
    }

    swap(i: 0,data.size()-1);
    int rv = data.remove(data.size()-1);

    downHeapify(pi: 0);

    return rv;
}
```

{2, 3, 7, 6, 4, 9, 8, 10, 19, 18}



break till 10:30

mini = 0 1 3 8

pi = 0 1 3

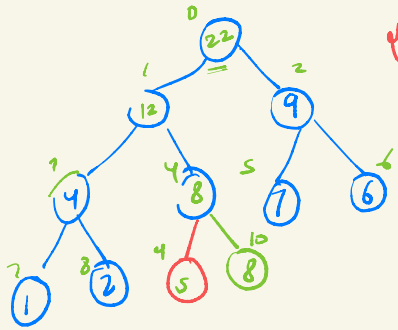
lci = 1 3 7

rci = 2 4 8

{22, 12, 9, 4, 8, 7, 6, 1, 2, 5, 8, 3}

ci = 10 1 0

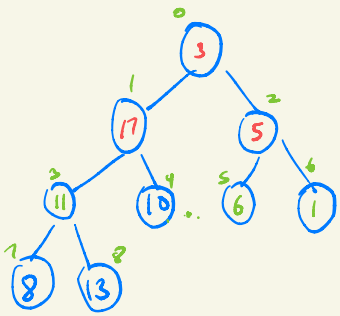
pi = 1 0 -1



```java
private void swap(int i, int j){
    int valAtI = data.get(i);
    int valAtJ = data.get(j);

    data.set(i,valAtJ);
    data.set(j,valAtI);
}

private void upHeapify(int ci){
    int pi = (ci-1)/2;

    if(pi>=0 && data.get(pi) < data.get(ci)){
        swap(ci,pi);
        upHeapify(pi);
    }
}
// add
public void add(int val){
    data.add(val);

    upHeapify(data.size()-1);
}
```

6, 4, 1, 3, 10, 9, 17, 8, 13

HOP



arr => {3, 17, 5, 11, 10, 6, 1, 8, 13}
0 1 2 3 4 5 6 7 8

```
void swap(int[] data, int i, int j){
    int temp = data[i];

    data[i] = data[j];
    data[j] = temp;
}

void upHeapify(int[] data, int ci){
    int pi = (ci-1)/2;

    if(pi>=0 && data[pi] > data[ci]){
        swap(data,pi,ci);
        upHeapify(data,pi);
    }
}

void buildHeap(int arr[]) {
    for(int i=0; i<arr.length; i++){
        upHeapify(arr,i);
    }
}
```

PQ

Priority Queue

PriorityQueue<Integer> pq = new PriorityQueue<>();
↳ min pq {min Heap}

pq.add(3)
pq.add(4)

pq.add(2);

pq.peek();

pq.remove()
pq.remove

class Student {
int roll_no;
String name;
}

Comparable
↓
Interface
comparator

51, 52

```
public int compareTo(Student o){
    return this.roll_no - o.roll_no;
}
```

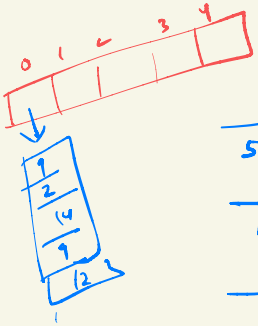
s1 = {1}
s2 = {2}

1 - 2 ⇒ -1

this.roll_no > 0. roll_no
⇓
(+) ve

this.roll_no < 0.
⇓
(-) ve

smaller on the top ⇒ this - other
greater on the top ⇒ other - this



matrix

5 10 averages

matrix

→ 5	9	2	14	9	12	8
→ 1	1	5	10	11	7	13
→ 4	5	7	15	6	17	18
→ 2	3	1	19	16	5	4
→ 3	4	8	9	1	2	3

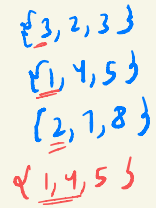
PQ < int[] > pq
= new PQ <>();

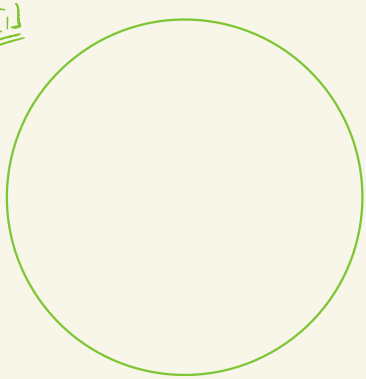
pq

{3, 2, 3}
{1, 4, 5}
{2, 7, 8}

```java
int[][] matrix={{3,2,3},{1,4,5},{2,7,8}};

PriorityQueue<int[]> pq = new PriorityQueue<>((int[] t, int[] o) ->{
    // return t[0] - o[0]; // min pq
    return o[1] - t[1];
});

for(int i=0; i<matrix.length; i++){
    pq.add(matrix[i]);
}

int[] top=pq.remove();
for(int i=0; i<top.length; i++){
    System.out.print(top[i]+" ");
}
```

{3,2,3}





```java
public int compareTo(Student other){
    return this.roll_no - other.roll_no; // min pq
    // return other.roll_no - this.roll_no //=> max pq
}
```

```
6
7 10 4 3 20 15
4
```

3 smallest elements

③

5, 1, 10, 4, 9, 3, 11, 4, 12

```java
public static int findKthLargest(int[] arr, int n, int k){
    PriorityQueue<Integer> pq = new PriorityQueue<>();

    for(int i=0; i<n; i++){
        pq.add(arr[i]);

        if(pq.size() > k){
            pq.remove();
        }
    }

    return pq.peek();
}
```

```java
public static int findKthSmallest(int[] arr, int n, int k){
    PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

    for(int i=0; i<n; i++){
        pq.add(arr[i]);

        if(pq.size() > k){
            pq.remove();
        }
    }

    return pq.peek();
}
```

10:30

break

aaabbcdefegefee

sb a= "Samreet"

```java
public String frequencySort(String s) {
    HashMap<Character,Integer> map = new HashMap<>();

    for(int i=0; i<s.length(); i++){
        char ch = s.charAt(i);
        map.put(ch,map.getOrDefault(ch, defaultValue: 0)+1);
        // if(map.containsKey(ch)){
        //     map.put(ch,map.get(ch)+1);
        // } else {
        //     map.put(ch,1);
        // }
    }

    ArrayList<Character> keys = new ArrayList<>(map.keySet());

    PriorityQueue<Character> pq = new PriorityQueue<>((Character t, Character o) -> {
        return map.get(o) - map.get(t);
    });

    for(char ch : keys){
        pq.add(ch);
    }

    StringBuilder sb = new StringBuilder();
    while(pq.size()>0){
        char ch = pq.remove();
        int fre = map.get(ch);

        while(fre -- >0){
            sb.append(ch);
        }
    }

    return sb.toString();
}
```

a=3
b=2
c=1
d=1
e=5
f=2
g=1

map.get(other)
map.get(this)

a>b

ch=b
fre=2

b f d c g

sb.append

sb=eeeeeaaabbfdcg

⇒ eeeeeaaaafbbgdc

6 7
1 2 3 9 10 12

k=10

pq

13 29 12

t1=9
t2=10

1×9+2×10

steps = 0+1+2 = 3

```java
static int solve(int a[],int n,int k){
    PriorityQueue<Integer> pq = new PriorityQueue<>();

    for(int e:a){
        pq.add(e);
    }

    int steps = 0;
    while(pq.size()>1){
        if(pq.peek()>=k) return steps;

        int t1 = pq.remove();
        int t2 = pq.remove();

        int newSweetness = 1*t1 + 2*t2;
        pq.add(newSweetness);

        steps++;
    }

    if(pq.peek()<k) return -1;
    return steps;
}
```

10 < 20

ans = { 10, 14, 16, 18, 20 }

x = 15

k = 4

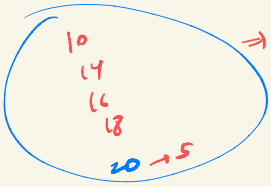
n = 10^8

k = 10, 20

n log k

14 15 18

element with smaller dis with x



⇒ n log n

n log k

ans = { 14, 16, 18, 10 }

{ 10, 14, 16, 18 }

```java
public List<Integer> findClosestElements(int[] arr, int k, int x) {
    PriorityQueue<Integer> pq = new PriorityQueue<>((Integer t, Integer o) -> {
        if(Math.abs(t-x) == Math.abs(o-x)){
            return t - o;
        }

        return Math.abs(t-x) - Math.abs(o-x);
    });

    for(int e:arr){
        pq.add(e);
    }

    List<Integer> ans = new ArrayList<>();
    while(ans.size()<k){
        ans.add(pq.remove());
    }

    Collections.sort(ans);
    return ans;
}
```

this - other // 8

abs(this - x) - abs(other - x)

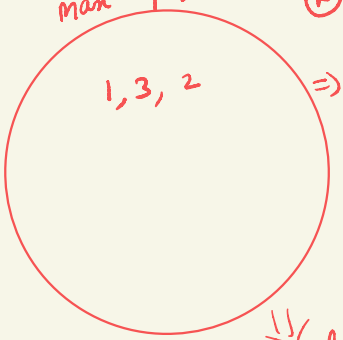
k = 3

{ 5, 1, 3, 7, 2, 4, 11 }

{ 1, 2, 3 }

k smallest elements → n log n (min pq)

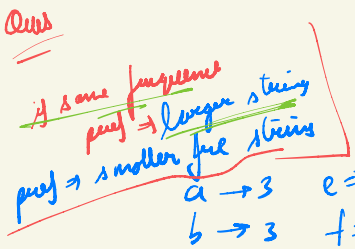
n log k (max pq)

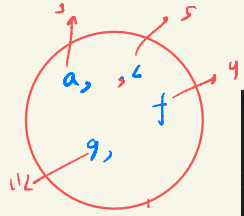


⇒ log k

n log k

n log n





{a, f, c, g}
{g, c, f, a}

```java
public List<String> topKFrequent(String[] words, int k) {
    HashMap<String,Integer> map = new HashMap<>();

    for(String s : words){
        map.put(s,map.getOrDefault(s, defaultValue: 0)+1);
    }

    PriorityQueue<String> pq = new PriorityQueue<>((String t, String o)->{
        if(map.get(t) == map.get(o)){
            return o.compareTo(t);
        }
        return map.get(t) - map.get(o);
    });

    for(String key:map.keySet()){
        pq.add(key);

        if(pq.size()>k){
            pq.remove();
        }
    }

    List<String> ans = new ArrayList<>();

    while(pq.size()>0){
        ans.add(pq.remove());
    }

    Collections.reverse(ans);
    return ans;
}
```

⇒ o - t
⇒ smaller first on top
n log k

— bricks for smaller changes
— ladders for bigger changes

[4,12,2,7,3,18,20,3,19]

ladders = 2, bricks = 10

{2, 3, 4, 7, 18, 19, 20}

```java
public int furthestBuilding(int[] heights, int bricks, int ladders) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();

    for(int i=1; i<heights.length; i++){
        int diff = heights[i] - heights[i-1];

        if(diff<=0) continue;
        pq.add(diff);

        if(pq.size()>ladders){
            int min = pq.remove();
            bricks -= min;
        }

        if(bricks < 0) return i-1;
    }

    return heights.length - 1;
}
```

{4, 12, 24, 48, 50, 52, 31, 37, 40}

b = 10 8 6 0 -3
l = 2

min pq
12, 24

addNum(2)
findMedian()

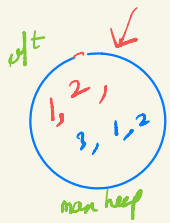
2, 4, 1, 5, 6, 3, 8, 5, 1, 2

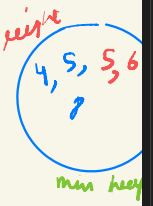
```java
public void addNum(int num) { // log(n)
    if(left.size()==0 || num<=left.peek()){
        left.add(num);
    } else {
        right.add(num);
    }

    int ls = left.size();
    int rs = right.size();

    if(ls - rs == 2){
        right.add(left.remove());
    } else if(rs > ls){
        left.add(right.remove());
    }
}

public double findMedian() { // O(1)
    if(left.size() == right.size()){
        return (1.0*left.peek() + right.peek()*1.0 )/(2.0);
    } else {
        return left.peek()*1.0;
    }
}
```

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

break tie

10:40





3 < 4

1 < 2

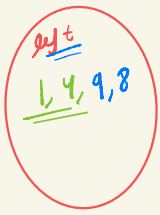
ls == rs

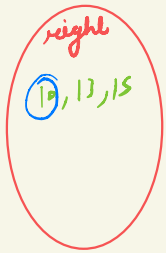
ls = = rs + 1

left.peek ⇒ 4

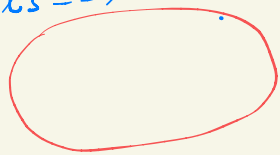
left ⇒ smaller numbers

right ⇒ larger numbers





ls == rs

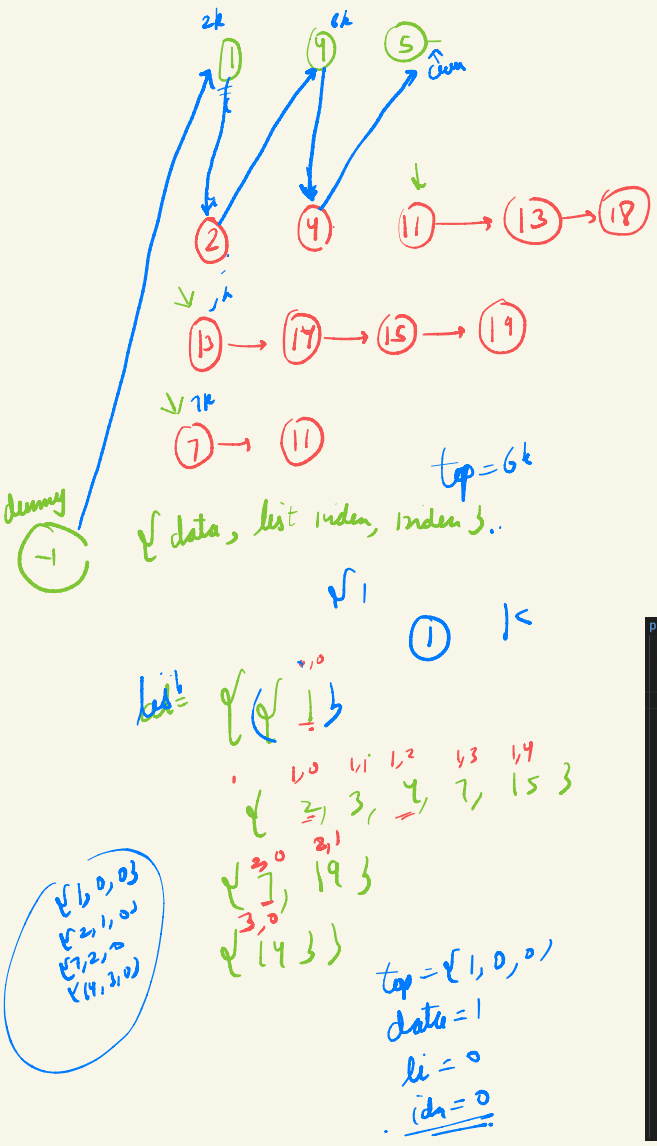


ls = rs + 1

↓

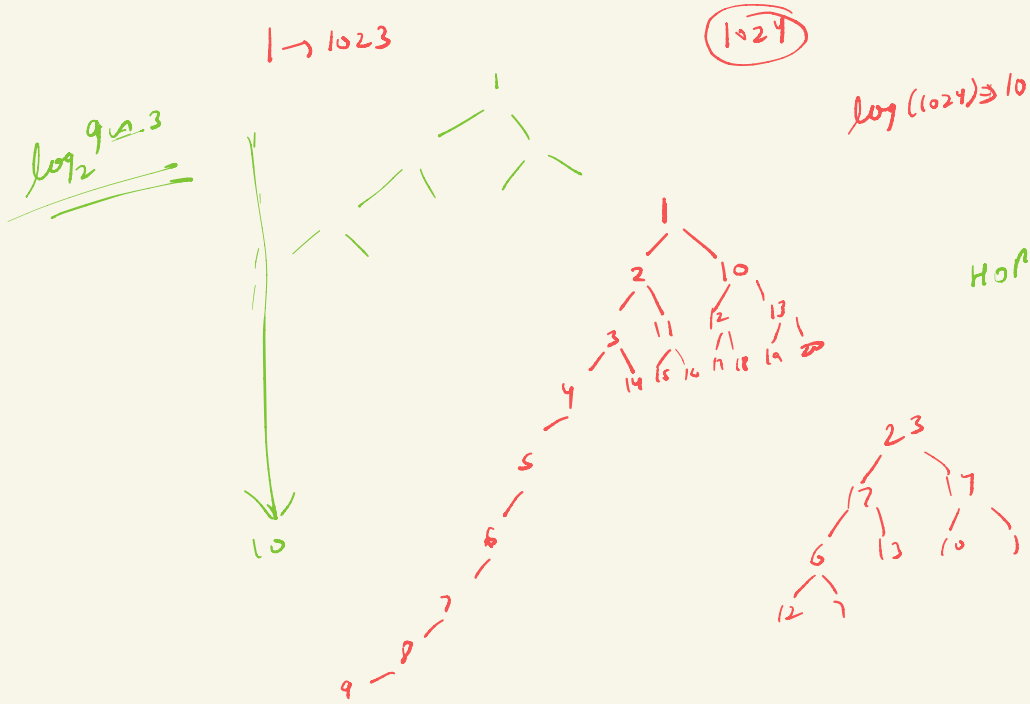
left.peek

1, 4, 8, 9, 10, 13, 15




```java
public ListNode mergeKLists(ListNode[] lists) {
    PriorityQueue<ListNode> pq = new PriorityQueue<>((ListNode t, Li
        return t.val - o.val;
    });

    for(ListNode ln : lists){
        if(ln!=null)
            pq.add(ln);
    }

    ListNode dummy = new ListNode(-1);
    ListNode curr = dummy;

    while(pq.size()>0){
        ListNode top = pq.remove();

        curr.next = top;
        curr = curr.next;
        if(top.next!=null){
            pq.add(top.next);
        }

    }

    return dummy.next;
}
```

```java
public static ArrayList<Integer> mergeKSortedLists(ArrayList<ArrayList<I
    ArrayList<Integer> ans = new ArrayList<>();

    PriorityQueue<int[]> pq = new PriorityQueue<>((int[] t, int[] o)->{
        return t[0] - o[0];
    });

    for(int i=0; i<lists.size(); i++){
        int data = lists.get(i).get(index: 0);
        pq.add(new int[]{data,i,0});
    }

    while(pq.size()>0){
        int[] top = pq.remove();

        int data = top[0];
        int li = top[1];
        int idx = top[2];

        ans.add(data);
        if(idx + 1 < lists.get(li).size()){
            data = lists.get(li).get(idx+1);
            pq.add(new int[]{data,li,idx+1});
        }
    }

    return ans;
}
```


1 → 1023
1024
log (1024) ≥ 10
log2 9 ≈ 3
Hop
10