21/10/2023

* Binary Search:

$\infty = 1 << 31$ OR $\infty =$ sys.maxint $-\infty = -(1<<31)$ $-\infty = -($sys.maxint$)$

	sorted		unsorted	
	Q=1	Q=Q	Q=1	Q=Q
LS	1 x N, 1	Q x N, 1	1 x N	Q x N, 1
BS	log N, 1	Q log N, 1	N log N + log N → N	N log N + Q(log N) → N

* $array_N$: 15, 24, 19, -31, 42, -10, 16, 19

→ O/P → get the element(x) <= to the given element
↳ Floor of the element

Q queries:

* Brute Force: Linear search, look for each element, if x (1st element), if it is less than element, replace min as x,

for i in range (Queries):
    ans = -sys.max or -(1<<31)
    for j in range(len(arr)):
        if arr[j] <= x:
            ans = max[ans, arr[j]]
    return ans

T.C = Q x N
S.C = 1

* 2nd Approach: Sort the data, use Binary search

low = 0; hi = n-1, ans = -(1<<31)
while(low <= hi):
    mid = (hi + low) / 2
    if ar[mid] <= ele:
        ans = ar[mid]
        low = mid + 1
    else
        hi = mid - 1
    return ans

T.C = N log N + Q log N
S.C = N

* Problem: a_N: 20, 30, 20, 19, -16, 24, 20, 30, 19, 40

Queries: 20 → Freq. of 20 → 3 (count of 20 is 3)
-16 → 1
19 → 2
-16 → 1
35 → 0

* Brute Force: Go through each element & update count as per the frequency of the given element. T.C = Q × N, S.C = 1

cnt = 0 for i in arr: if i == ele, cnt = 1, return cnt

* 2nd Approach: Count Array, Fill the count array with the elements count & ~~for each~~ ~~update the~~ return the element at count(a[i] - a) → return count. T.C = N + Q(1), S.C = R

Range can only be 10^5, → count Array max len. R = range

* 3rd Approach: Sort the array,

-16, 19, 19, 20, 20, 20, 24, 30, 30, 40 → sorted Array

→ Apply binary search to get 1 element (that is given)
→ Once we get the element, traverse to the left & right to get the remaining count, P_1 → ele - 1, P_2 → ele + 2, if matched P_1 += 1, check P_1 → if matched P_2 += 1, stop when condition failed → for failed condition, P_1 should be incremented & checked same with P_2 → so frequency = $P_2 - P_1 + 1$

* Code:

```
sort.array()
def BS(arr, low, hi, ele):
    while (low <= hi):
        mid = (low + hi) // 2
        if arr[mid] == ele:
            P1 = mid - 1, P2 = mid + 1
```

T.C = NlogN +

```
while (P1 >= 0 and arr[P1] == ele):
    P1 -= 1
while (P2 <= n-1 and arr[P2] == ele):
    P2 += 1
count = P2 - P1 - 1
elif arr[mid] > ele:
    hi = mid - 1
    Bs(arr, low, hi, ele)
else:
    low = mid + 1
    Bs(arr, low, hi, ele)
return count
```

* 4th Approach: Instead of 2 pointer, we can do B.S on the array multiple times to get the index of the last occurrence of first occurrence of my element.

```
Last Occurrence
low = 0, hi = n-1, lo = -1
while (low <= hi):
    mid = (low + hi) // 2
    if arr[mid] == ele:
        lo = mid
        low = mid + 1
    elif arr[mid] > ele:
        hi = mid - 1
    else:
        low = mid + 1
return lo
```

```
First Occurrence
low = 0, hi = n-1, fo = -1
while (low <= hi):
    mid = (hi + low) // 2
    if arr[mid] == ele:
        fo = mid
        hi = mid - 1
    elif arr[mid] > ele:
        hi = mid - 1
    else
        low = mid + 1
return fo
```

T.C = $N\log N$ (sorting) + $Q(\log N + \log N)$ (to, to)

S.C = 1 - inbuilt library meye - 1000 extra space taken

* 5th Approach:

T.C = $N\log N + N + Q(\log N + 1)$,

S.C = $N + N$

compressed array: unique elements

frequency array: freq. of those unique elements.

→ arr = 20, 30, 20, 19, -16, 24, 20, 30, 19, 40

compressed arr = (-16, 19, 20, 24, 30, 40) (0, 1, 2, 3, 4, 5)

frequency arr = [1, 2, 3, 1, 2, 1]

→ Approach: Apply B.S on compressed array, we will get the index of that element, (mid), & we return, frequency(mid) to get the frequency.

* comparison Binary: we take compressed array, frequency arr, we traverse through main array, for this we first we take compressed array = [arr[0]], freq. arr = [1] → this is for the first element, from 2nd we can use pointers to add elements

compressed arr = [-16] $P_1 = 1$ → compressed arr

frequency arr = [1] $P_2 = 0$ →

→ arr[P_1] == compressed arr[P_2]

→ if this same, then frequency array(P_2] += 1 to inc. count

P_1 += 1

else we just add the element to the compressed arr & initialize frequency[P_1] = 1

→ now the compressed array will have the first element of the sorted original array & freq array will have 1's, now $P_1 = 1$, $P_2 = 0$

→ we need to check if arr[P_1] == compressed array[P_2], if it is same we need not inc. freq array[P_2] += 1 else we need to move to the next element check, if not we need to add that element into the compressed array & also add 1 to frequency array append 1.

→ Once we have all the frequency array & compressed array

→ B.S on the frequency array to ele for each query

in B.S we need to take (arr, low, len(compressed array or P_2, ele)

→ if arr[mid] == ele, mid = idx & if idx != -1, return (idx) → return.

*Code:

```
arr.sort()
ca = [arr[0]], fa = [1], P1 = 1, P2 = 0
while (P1 <= len(arr)):
    if ca[P2] == arr[P1]
        fa[P2] += 1
        P1 += 1
    else:
        ca.append[arr[P1]]
        fa.append[1]
        P1 += 1
```

```
idx = -1
hi = P2 or len(compressed arr)
low = 0
BS(arr, low, hi, ele)
    mid = (hi + low) // 2
    if arr[mid] == ele:
        idx = mid
    elif arr[mid] > ele:
        hi = mid - 1
    else:
        low = mid + 1
else
    if idx != -1
        return freq(idx)
```

* 6th Approach: arr_N = 20 30 20 19 -16 24 20 30 19

sorted: arr_N : -16, 19, 19, 20, 20, 20, 24, 30, 30, 40

Queries

20
-16
19
-16
35

Queries = 20, -16, 19, -16, 35

sorted queries = -16, -16, 19, 20, 35

freq arr · Q =

1	1	2	3	0

* 7th Approach: Original array, store the elements in a hashmap; maintain the unique elements as its counters in hash map.

arr = 20, 30, 20, 19, -16, 24, 20, 30, 19, 40

hashmap: {20: 3, 30: 2, 19: 2, -16: 1, 24: 1, 40: 1}

T.C = N x 1 + Q x 1

S.C = N

Print the corresponding value.

* code:

```
hashmap = { }
for i in arr:
    if arr[i] in hashmap:
        hashmap[i] += 1
    else:
        hashmap[i] = 1
print (hashmap[ele])
```

T.C. = n
S.C = n

22/10/2023

* Problem: I/P: 25 O/P: 5
  64 8

→ Find the square root of the given number. Given number is a perfect square root

* Brute force solution: for loop from 1 to N & if i x i is N return N

T.C = $\sqrt{N}$
S.C = 1

25 → will run till 5 so $\sqrt{36}$
36 → will run till 6. $\sqrt{25}$

* 2nd Approach: we can apply B.S, low = 1, hi = N (25), mid = $\frac{hi + low}{2}$ ⇒ if arr[mid] x arr[mid] == N, return mid, else if arr[mid] x arr[mid] > k → hi = mid - 1, else low = mid + 1

* Code:

```
low = 0, hi = N
def BS(low, hi, N):
    while (low <= hi):
        mid = (hi + low) // 2
        if mid*mid == N:
            return mid
        elif mid*mid > N:
            hi = mid - 1
        else:
            low = mid + 1
```

T.C = $\log_2 N$
S.C = 1

* In case, we need to calculate pth root, we can simply apply pow(mid, P) ==N, power inbuilt takes log P T.C
So total **T.C = $\log_P \times \log N$**

→ In case of squareroot, we can check pow(mid, 2) → $\log_2 2$ which equates to 1, so **T.C = 1 x log N** | **S.C = 1** → This is why with pow function also has same **T.C → log N, 1**

* **TO-DO → Plot graph sqrt vs log n**

$\sqrt{N}$	log n
4 = 2	$\log_4 = 2$
8 = 4	$\log_8 = \log_2 4 = 4$
$10^{100} = 10^{10}$	log

* a_N: 5, 12, -6, 24, 19, 10, 7 **[a+b=K]** $i \neq j$
K = 29

* Solutions: Brute Force: nested for loop & calculate sum of each value with i+j to get the ans. **T.C = N^2, S.C = 1**

* 2 pointer Approach: sort away, 2 pointers, $P_1=0$, $P_2=N-1$, get the ans by adjustments **T.C = N log N + N** (log N: sort, N: 2 pointers) **S.C = 1** → inbuilt merge

* Binary search:

sorted: -6 5 7 10 12 19 24
0 1 2 3 4 5 6

→ For loop for fixing the element & calculating the complement.
→ B.S from low i+1 to hi (N-1) & find the complement (k-a)

* Code:

```
arr.sort()
for i in range(len(a)):
    b = k - a[i]
    if bs(ar, i+1, N-1, b):   # low, hi
        return True
    else:
        False
```

$T.C = (N \times \log N) + N \times \log N$ (sorting + for loop → B.S)
$S.C = 1$

Reverse B.S also works:

```
for i in range(1, len(ar)):
    b = k - a[i]
    if Bs(ar, 0, i-1, b)   # low, hi
        return True
    else:
        False
```

-6 5 7 10 [12] 19 24 (b, i=4)
0 1 2 3 4
→ B.S from 0 (low) to 3 (hi) for k - a[i]

* a_N: 2, 2, 5, 5, 12, 12, 17, 24, 24, 30, 30
→ array is sorted & all elements are repeated twice except 1
→ Find the unique element → Find '17'

* Brute force: Nested for loop → count != 2, return i
$T.C = N^2$, $S.C = 1$

* 2nd Approach: XOR: for loop for all elements to XOR & ans = unique element
$T.C = N$, $S.C = 1$

* 3rd Approach: for (i, len(arr), 2), jump 2 steps, if a[i] != a[i+1] → return a[i]
$T.C = \frac{N}{2}$, $S.C = 1$

* 4th Approach: a_N: 0 (2), 1 (2), 2 (5), 3 (5), 4 (12), 5 (12), 6 (13), 7 (24), 8 (24), 9 (30), 10 (30)

low=0, hi=10 → mid = $\frac{10+0}{2}$ = 5 → with mid check even or odd

* even case → mid=5, (a) a[mid] = a[mid+1] → low = mid+2

(b) a[mid] = a[mid-1] → hi = mid-2.

* odd case → (a) a[mid] == a[mid+1] → hi = mid-1

a[mid] == a[mid-1] → low = mid+1

→ initial check → if (a[mid] != a[mid-1] && a[mid] != a[mid+1])
return a[mid]

→ if len(a) == 1 → return a directly

→ if mid != 0, mid is n-1, cannot check (i+1) or (i-1) → if conditions.

* MIN-Max Problem: Minimizing the max result:

Workers, need to complete the work in minimal time up from all the possibilities return the min value of the max.

Time taken = 30, 132, 15 → 132 max time → } 75 min time.
= 62, 75, 10 → 75 max time → }

* N elements → partitions N-1 | 1 | 2 | 3 | 4 | 5 | 6 [1-5 Partitions]

→ Combinations → $\boxed{{}^{n-1}C_{k-1}}$ N-1 = Places
K-1 = partitions.

→

10 | 20 | 5 27 40 60 15

10, 20, 147 = 147

10 | 20, 5 | 27 40 60 15

10, 25, 142 = 142

10 | 20 5 27 | 40 60 5

10, 57, 105 → 105

10 | 2 5 27 40 | 60 5

10, 74, 65 → 74

10 | 2, 5, 27, 40, 60 | 5

10, 134, 5 → 134

..... 10, 2 | 5 | 27, 40, 60

12, 5, 127 → 127

..... 10, 2 | 5 27 | 40, 60

12, 32, 160 → 160.

continues till the end. return the min(all sums).

150
2+
15
142

2+
45
52

27
40
7
74

27
100
7
134

→ Back Tracking Approach: req. parameters: arrays, size of arrays, idx, no. of partitions, current ans [the sum of partitions], int max = ans (min. int → sum of partitions → max.) to get the min. value before taking it in the ~~array~~ function, but in the array to get the max of sum of all partitions → we take int min

cans = -0, idx = 0 5, 10, 20, 40, 90, 15

```
void part(arr[], N (size), K (partitions), idx, cans)
    for i in range(idx, N):
        x = x + arr[i]
        part(arr, N, k-1, i+1, max(cans, x))
```

Base condition → when K == 1, 3 size, partitions 2, k = 2, k = 1

```
for i in range(idx, N):
    y = y + arr(i)
```

cans = max(cans, y)

ans = min(ans, cans)

return

* Code:

```
ans = ∞
void part( ar[], N, K, idx, cans):
    if K == 1 or idx == n
        for i in range( idx, N):
            y += ar[i]
            cans = max( cans, y)
            ans = min( ans, cans)
        return
    x = 0
    for i in range( idx, N):
        x = x + a[i]
        part( ar, N, k-1, i+1, max( cans, x))
```

TODO:
Trace it own

* 2nd Approach: max sum that we get is 1 worker → sum of all elements in array

min sum is the max(arr) with N-1 workers, each worker, 1 element → max element is the time.

low = max(ar), hi = sum(ar).

ar: 10, 20, 5, 27, 40, 60 | 15

low = max(ar) = 60

hi = sum(ar) = 177.

mid = (177+60)/2 = 118

a_N - | 10 | 20 | 5 | 27 | 40 | 60 | 15 |

mid = 118

sum=0 → sum = 10, sum = 10+20, sum = 10+20+5 < 118 → can add, sum = 10+20+5+27 < 118 → 10+20+5+27+40 < 118 next +60 > 118 → no partition after +0 → ans = 118 (assume)

→ Task is to minimize so hi = 117 (mid-1)

low = 60, hi = 117, mid = $\frac{117+60}{2}$ = 88 = mid

→ hi = mid-1 → low = 60, hi = 87, mid = $\frac{87+60}{2}$ = $\frac{147}{2}$ = 73

sum = 0, 10, 30, 35, 62, 27 → partition

sum = 0 → +0 → 2nd partition

sum = 60 + 75 → 75 → max → but mid = 73 → False

so, if we're not able to minimize 73 mins → we cannot minimize 72, 71 .... no

→ so the range will be from 74 ..... 118.

now low = 74 hi = 87, mid = 80

low = 74, hi = 79, mid = 76

low = 74, hi = 75 → mid = 74

now low = mid+1

low = 75, hi = 75, mid = 75 → valid

* Code!

Next Page → → →

int low = max(arr), hi = sum(arr), ans = hi → worst case

while (low <= hi):

mid = (hi + low) // 2

if (valid(arr, N, k, mid)):

ans = mid

hi = mid - 1

else:

low = mid + 1

return ans

**

range = hi - low

T.C = log(hi - low + 1) × N

= N log(sum - max + 1)

S.C = 1

def valid(arr, N, k, val): k cuts → k-1 partitions

sum = 0, partitions = 0

for i in range(N-1):

sum += arr[i]

if (sum > val):

p + 1 = 1 → count as partitions

sum = arr[i] → current element after partition

return p <= k-1 → valid true / false

* TODO: Problem: Houses: 5, 20, 40, 47, 70, 90, 107, 140

→ K = 3, Houses are vacant, Houses should be far from each other

→ maximize the gap b/w houses

diff b/w 5 to 20 is 15

so 5, 20, 40, 47, 70, 90, 107, 140

15 30 20

everyone is atleast 20ft apart → min (elem)

42, 43, 50 → 42 min distance

TODO

→ maximizing minimum → max ∩ all min elements

→ start from 1st house as prev house, house[0] then we need to find next position can be placed (greedy)

→ arr[i] - prev , prev = 0 → arr[0]

prev = arr[i]

* Binary search: hi = arr[n-1] - arr[0] → [last - first] (elem)

low = min (diff b/w adj houses) → all taken accepted

= min [arr[i] - a[i-1])

5, 20, 40, 47, 70, 90, 107, 140

15 20 7 23 20 19 31 → low = 7

*

Code:

```
low = min(arr[i] - arr[i-1], hi = arr[n-1] - arr[0],
while ( low <= hi):
    mid = (hi+low)/2
    if valid (arr, N, k, mid):
        ans = mid;
        low = mid + 1 → increasing the distance b/w houses to low ↑ to inc. the diff.
    else
        hi = mid - 1
```

valid (arr, N, k, mid, prev = arr[0]): → h = 1

```
for i in range(1, N):
    if ( a[i] - a[prev] ) >= mid:
        prev = i
        h += 1
```

→ return h >= k

T.C = $\log(hi - low + 1) \times N = \log(hi - low - 1) \times N$
(B.S) (valid func?)

S.C = 1

* Minimizing or maximum or Maximizing minimum → we should think BS.

→ now for maximizing the ans → low = mid + 1
minimizing the ans → hi = mid − 1.

* In the above problem, we check if P >= k in the valid function, because, we need to pick **atleast** k families but not less.