* Minimizing ~~or~~ maximum or Maximizing minimum → we should think of 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.

28/10/2023

* A of size N: [5, 12, 17, 24, 36, 90] → 6
B of size M: [-3, 7, 14, 19, 21] → 5 } 11 elements

→ N + M = always odd.
→ No duplicates
→ Both sorted arrays
→ Median of both sorted Arrays. → (17) → middle elements

* **Brute Force**: Combine 2 arrays & find the mid.

$\boxed{T.C = N+M, S.C = 1}$

* **2nd Approach**: Create a new array using 2 pointer, & print $\left(\frac{N+M}{2}\right)$ element

$\boxed{T.C = N+M, \quad S.C = N+M}$

* **3rd Approach:** Count Array till $(\frac{N+M}{2})$, $P_1 = A[0]$, $P_2 = B[0]$, & $c = 0$, $P_1 \leq P_2$, $c += 1$, P_1++, P_2++, till count == $\frac{N+M}{2}$

Print the min element when reached $\frac{N+M}{2}$

$$\boxed{T.C = \frac{N+M}{2}, \quad S = 1}$$

* **Code:**

Rough idea, might have to include conditions

```
tot_size = (N+M)/2
P1 = 0
P2 = 0
while (P1 <= tot_size and P2 <= tot_size):
    if A[P1] <= B[P2]
        P1 += 1
    else
        P2 += 1
Print(min(A[P1], B[P2])
```

Incase, the array ends, & we need to still compare & reach $\frac{N+M}{2}$, we need to still move forward, so we can handle this condition for remaining which can be ; if P_1 < tot size :

```
while P1 < tot size:
    P1 += 1
Print( A[P1])
if P2 < tot_size :
    while P2 < tot_size:
        P2 += 1
    Print ( B(P2])
```

* 4 th Approach: min = -3, max = 90 → min(A[0], B[0],
↳ -3
max(A[-1], B[-1]) → 90

→ low = min (A[0], B[0]) = -3
hi = max (A[-1], B[-1]) = 90

→ mid = $\frac{-3+90}{2} = \frac{87}{2} = 43$

A_N: 5, 12, 17, 24, 36, 90
P_M: -3, 7, 14, 19, 21

* After we set 43, if we have equal elements on left of 43 & right of 43 → it is the median.

10 elements less than 43
1 element greater than 43

→ so this mid is invalid, since lower elements count is more, redue range → hi = mid -1

→ new mid = low = -3, hi = 42, $\frac{42-3}{2} = \frac{39}{2} = 19$

lower elements = 6
higher elements = 4

Since lower elements < higher elements → hi = mid -1

→ low = -3, hi = 18 → new mid = $\frac{18-3}{2} = \frac{15}{2} = 7$

lower elements = 2
higher elemts = 8
higher elemats > lower elements

low = mid + 1

→ low = 8, hi = 18 → $mid = \frac{18+8}{2} = \frac{26}{2} = 13$

→ low. ele = 4, hi. elemen = 6

hi > low

→ low = 14 hi = 18 → $mid = \frac{18+14}{2} = \frac{32}{2} = 16$

low e = 5, gc = 6

g.e > low.e

low = mid + 1

→ low = 17, hi = 18, mid = 17

low. e = 5, g.e = 5

* Tip → get low elements count, it will automatically give greater elements → (N + M - low)

→ B.S of A to get index → less elements = mid + 1.

B.S on B to get floor element → mid → mid + 1 elements

gc = (N + M) - low. count

steps:

```
1.  low = min (A[0], B[0]) ; hi = max( A[-1], B[-1])
2.  while ( low <= hi ):
              mid = (hi + low) / 2
main
BC
3.      int l. count  & int g.c
4.      if lc == gc  → return mid
5.      if lc < gc  →  low = mid + 1
6.      else if gc < lc  →  hi = mid - 1
```

* to get g.c (greater count) → (N+M) − l.c

→ But if mid . element is present in l.c we need to subtract 1

$(N + M - l.c - 1)$ → to find if mid ele is present return 1 else return 0

so → $(N+M) - l.c - (BS(A, mid, 0, N-1) \,||\, BS(B, mid, 0, M-1))$

↳ return 1 if mid == ele
else return 0

* For L.C : bsfloor(A, 0, N−1, mid) + bsfloor(B, 0, M−1, mid)
+2 (since it is 0 based index)

$$\boxed{T.C = \log(hi - low + 1) \times \overbrace{4 \times \underbrace{\log N}_{\log N + M}}^{4\ BS\ (2\ for\ L.C\ \&\ 2\ for\ G.C)}}$$

$$\boxed{\begin{array}{l} T.C = \log(hi - low + 1) * 2\log N * 2\log M \\ S.C = 1 \end{array}}$$

* Phone Number

	insert	search	delete
unsorted Array	O(1)	O(n)	O(n)
sorted Array	O(log n)+N ↳ shift	O(log n)	O(log n + N) ↳ shift elements
stack | Queue	O(1)	O(N)	O(N + N) ↳ remove ↳ place it back
unsorted Linked List	O(1)	O(N)	O(N)
sorted L.L	O(N)	O(N)	O(N)
B.S.Tree	(H) Height of Tree	H	H
Balanced B.S.T	log n	log n	log n
Count Array	(1) C[300] = T	if C[x] (1) in runtime	C[x] = False (1)

Hash Function → h(x) = x % S → size
h(x) ↓ hash fnctn, x ↓ hash value

203, 500, 707, 24, 84, 60

→ S = 10

500			203	24			707		
0	1	2	3	4	5	6	7	8	9

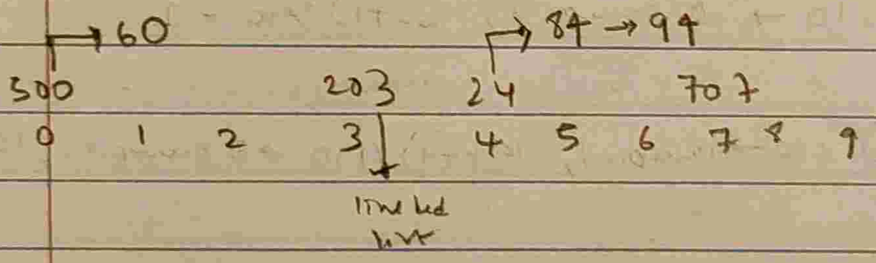
* **Collision** occurs when 2 diff numbers have same hash value → collision occurs

* **Collision Reduction techniques**:

1. Seperate chaining → L.L
   → B.B.ST

2. Open addressing → Linear Probing
   → Quadratic Probing
   → Double hashing.

* Incase of collision, if the index is already filled, we maintain a linked list, where we store the other connected elements.



* L is the length of the linked list (chain), travelling through the list we placing l times O(N)

→ Search → 84 % 10 → 4th index → traverse through and find it O(N)

→ same like search, search & delete O(N).

→ ~~[illegible]~~.

* 2nd way: Height of balanced B.S. Tree → log(n)

→ maint count for duplicates, just inc. count for the same value.

→ As the length of the list (L.L.) increases we maintain a cont valuated to

→ [illegible]

* Linear Probing in Open Addering:

$$\boxed{h(x) = (x+i) \% s \qquad , i = 0 \dots n}$$

6 → 203, 500, 707, 24, 84, 60, 94, 34.

s=10

500	60		203	24	84	64	707	34	14
0	1	2	3	4	5	6	7	8	9

$(84+0)\%10$ → occupied, so i is increased. $(84+1)\%10$ → 5th index.

$(60+0)\%10$ → 0 occupied, $(60+1)\%10$ → 1

94 → $(94+0)\%10$ → occupied, $(94+1)\%10$ → 5 occupied, $(94+2)\%10$ $(96)\%10$ → 6

34 → $(34+0)\%10$ → $(34+1)\%10$ → 5 occupied $(34+2)\%10$ = 6 occupied $(34+3)\%10$ → 7 occupied, $(34+4)\%10$ → 8 ✓

14 → $(14+0)\%10$ → $(14+1)\%10$, $(14+2)\%10$, $(14+5)\%10$

* Quadratic Probing : $\boxed{h = (x + i^2) \% s}$

$6 \rightarrow 203, 500, 707, 24, 84, 60, 34$

$- (34+0^2) \% 10, \quad (34+1^2) \% 10, \quad (34+4^2) \% 40$

* Linear Probing vs Quadratic Probing :

→ table : size 100 , 403, 503, 603, 703, 703, 404, 504, 704

	403	503	603	703	803	404	505	709
~~L~~Q : P :	3	4	~~3~~ 5	6	7	8	9	10
Probes :	0	1	2	3	4	4	5	6

	~~403~~	~~503~~	~~603~~	~~703~~	~~803~~	~~404~~	~~504~~	~~704~~
Q . P :	3	4	7	12	19	5	8	13
Probes :	0	1	2	3	4	1	2	3

→ total probes L.C = 25, Q.P = 16 steps.

→(always size for index in error index)

→ cluster size ∝ no. of probes

Quadratic.	Insert	Search	Deletion
Probes :	P	P	P

→ maintain linked array → 0, 1, -1, 1 occupied, 0 → empty, -1 Delete

search (n), Delete (n), Insert (n)

* Double Hashing : $\boxed{\begin{array}{l} h(x) = h_1(x) + h_2(x) \\ \qquad h_1(x) \times h_2(x) \end{array}}$

powered, [illegible]

$h_1(x) \rightarrow$ [illegible]

$h_2(x) \rightarrow$ [illegible] $(x+i^2) \% s$, ~~$(x^x) \% s$~~

$(x \times i^2) \% s, \quad (x+i) \% s, \quad$ ~~$(x^i) \% s$~~, $(x!) \% s$

* combine functions in double hashing.

* Good Hash Function:

→ Uniformly distribute keys over the entire table.

→ Function should be simple to calculate; (no powers, factorials)

→ Take double the [illegible] size as the table size, to have more empty spots & lower collisions.

```
        class Hashmap {
                int ht [10000]
                int availibility [10000] = {0}
                loop i = 0
insert  {       int index = (x + i^2) % 10000
                if (availibility [index] != 1):
                        ht [index] = x
                        availibility [index] = 1
                bool delete (int x):
Delete  {               int y = search(x)
                          if (y != -1):
                          availibiling [y] = -1.
                      else
                          print('Not found')
                def search (int x):
                    loop i = 0 ....
Search  {           int (index) = (x + i^2) % 10000
                    if (availibiling [index] == 0):
                          return -1
                    if (availibility [index] == 1) and
                          ht [index] == x)
                      return index.
```

29/10/2023

* Hashing avg. case → O(1) np worst case → O(logN).

* Hashing on strings : $h(x) = \left[\sum_{0}^{n-1} str(i)\right] \% M$.

→ abc = 97+98+99 == bac = 98+97+99 → it is same but the strings are different. the hash function gives True as the output while the answer should be false. This concludes that our hash function should be changed.

* $h(x) = \left(\sum_{i=0}^{N-1} str(i) \times i\right) \% M$

→ ab = (97×0) + (98×1) = 98 ab = (97×0) + (98×1) = 98

ba = (98×0) + (97×1) = 97 cb = (99×0) + (98×1) = 98

ab ≠ cb

* We can conclude that this hash function is also not optimal.

* In order to improve it we can start from (1 to N−1) for i

$h(x) = \left(\sum_{i=1}^{N-1} str(i) \times i\right) \% M$

aab = (97×1) + (97×2) + (98×3). aab = (97×1) + (97×2) + (98×3)

aac = (97×1) + (97×2) + (99×3). bba = (98×1) + (98×2) + (97×3)

aab ≠ bba

* The above change of i from 1 to N−1 is also not a good hash function as it gives 2 string as same when they are not.

* $h(x) = \left(\sum_{i=0}^{N-1} str(i) \times (i+1)^2\right) \% M$

ab = $97 \times (1)^2 + 98(2)^2 = 97 + 382$

ea = $101 \times (1)^2 + 97(2)^2 = 101 + 388$ ab ≠ ea

* The above hashmap function is also not the best.

* $h(x) = \left(\sum_{i=0}^{N-1} ctr(i) * P^{(i+1)} \right) \% M$; P = Prime Number → big number, not too big also, decent one = 31, 41 ....

* Finding Frequency → 7th solution covered hash map.

→ Hashmap → onrdered map (unsorted data) = $\overset{\text{insert}}{N(1)} + \overset{\text{search}}{Q(1)}$*, S.C = N

ordered map = $N(\underset{\text{insert}}{\log N}) + Q(\underset{\text{search}}{\log N})$, S.C = N

* ar_N : 5, 12, -6, 24, 39, 10, 15

k = 27

a + b = k

$i \neq j$

* **Brute Force**: 2 for loops, outer fixed element, inner to check

$$\boxed{T.C = N^2, \quad S.C = 1}$$

* **2 pointer**: sorting, P_1 & P_2 → move the pointers → till the sum is found

$$\boxed{T.C = \log N + N, \quad S.C = 1}$$

* **Binary Search**: sorting, start from 0, -(-6)+27 = 33, if we find 33 in the remaining array & if found return the pair

$$\boxed{T.C = \log N + N \times \log N, \quad S.C = 1}$$

* **Hash map**: store array in a hashmap, key: 5, → value: 1, 12: 1, pair = ar[i] - k or k - ar[i], find pair in hashmap, if yes return true else False

but, in case ar has 5 & k = 10, we need to return False, but pair value for 5 is 5, where it will return true, we need to check the value if > 1 return true.

* **code**:

```
hashmap = {}
for i in range(len(ar)):
    if ar(i) in hashmap:
        hashmap[ar(i)] += 1
    else
        hashmap[ar[i]] = 1
```

```
for j in hashmap :
    if (k-j) in hashmap :
        if j==(k-j) and hashmap[j] >=2 :
            return True
        elif j!=(k-j)
            return True
return False
```

T.C = 1
S.C = N

O(1) for insertion
O(1) for searching

* 5th Approach: Hashmap in a different way. Insert the element only if the pair value is present in the map.

a_N = 5, 12, -6, 24, 39, 10, 15 , k = 27

5 → pair = 22, hashmap empty 22 is not there insert 5

12 → pair = 15, hashmap = [5], 15 not there, insert 12

-6 → pair = 33, hashmap = [5, 12], -6 not there insert -6

24 → pair = 3, hashmap [5, 12, -6], insert 24

39 → pair , hashmap [5, 12, -6, 24], insert 39

10 → pair → 17, hashmap [5, 12, -6, 24, 39], 17 no there, insert 10

15 → pair → 12, hashmap [5, 12, -6, 24, 39, 10], 12 there → return True

* Code:

```
hashmap = {}
for i in range(len(a)):
    p = k - a[i]
    if p in hashmap:
        return True
    else:
        hashmap[a[i]] = 1
return False
```

check insert

$T.C = (1 + 1)$

S.C = N

The above can be done using a set also → same T.C for a single element

* Only if its a sorted array → T.C = logN + log N (logN for BST)

* **Problem:**

a_N : $\overset{0}{10}, \overset{1}{-12}, \overset{2}{13}, \overset{3}{6}, \overset{4}{4}, \overset{5}{-40}, \overset{6}{16}, \overset{7}{-9}, \overset{8}{2}, \overset{9}{15}, \overset{10}{13}$

→ print max sum subarray can generate.

→ sub array = continous part of an Array

→ $\boxed{\text{Max number of subarray} = \frac{n(n+1)}{2}}$ n = len(arr).

* subsequences & subsets = 2^N. [subset / subseq = not a continous Part of the array]

* **Generate Subarrays:** nested for loop → [outer loop (i) 0 → len(arr)]

innerloop for subset generations [j → i → len(arr)]

0 1 2 3

[0, (0,1), (0,1,2), (0,1,2,3)
1, (1,2), (1,2,3)
2, (2,3)
3]

* **code:**

```
for i in range(len(arr)):
    for j in range(i, len(arr)):
        sum=0
        for k in range( i , j+1 ):
            sum= sum + arr[k]
        ans = max(ans, sum)
return ans
```

$\boxed{T.C = N^2 (\text{subarrays}) \times \underset{\text{sum ser}}{N} \quad S.C = 1}$

* We can remove kth loop & just add the extra element to the sum value.
(last j element)

* Code:

```
ans = -(1<<31)
for i in range( len(arr)):
    sum=0
        for j in range(i, len(arr)):
                sum = sum+ arr[j]
                ans = max( ans, sum)
return ans
```

carried forward sum called carry forward method.

T·C = N^2
S·C = 1

* Dyamic Programming Solution: [UPCOMING CLASSES]

* **Problem:** $A_N : \overset{0}{12}, \overset{1}{-5}, \overset{2}{17}, \overset{3}{19}, \overset{4}{-3}, \overset{5}{5}, \overset{6}{16}, \overset{7}{24}$

→ Non- Decreasing Subsequences should be generated.

→ (12, 17, 19), (5, 16, 24), (-5, 5, 16, 24) etc....

* **Smaller Example:** $\overset{0}{12}, \overset{1}{-6}, \overset{2}{20}$

$$2^3-1 \left\{ \begin{array}{ccccl} & (2 & 1 & 0) & \\ & 0 & 0 & 1 & \rightarrow 12 \\ & 0 & 1 & 0 & \rightarrow -6 \\ & 0 & 1 & 1 & \rightarrow 12, -6 \\ & 1 & 0 & 0 & \rightarrow 20 \\ & 1 & 0 & 1 & \rightarrow 12, 20 \\ & 1 & 1 & 0 & \rightarrow -6, 20 \\ & 1 & 1 & 1 & \rightarrow 12, -6, 20 \end{array} \right.$$

* 2^3-1 → (1<<N), N = len(arr), (2^3-1 → combinations)

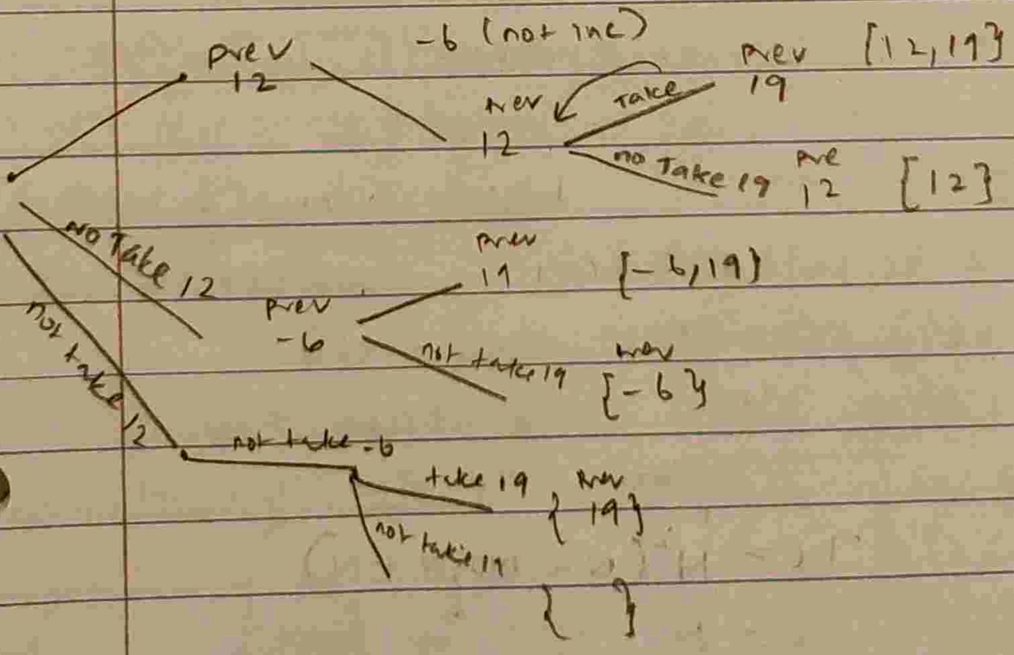
```
c=0
→ for i in range (1<<N):
      if subseq (i, arr, N)):      } T.C = 2^N
          c += 1
```

```
* bool subseq (i, arr, N):
          prev = -∞
          for j in range(len(arr)):  ──→ (i>>j)&1 == 0
     N        if (checkbit (i, j)):
                  if (prev <= arr[j]):
                      prev = arr[j];
                  else
                      return False

          return True.
```

→ $\boxed{\begin{array}{l} T.C = 2^N \times N \\ S.C = 1 \end{array}}$

* 2nd Approach: $(\overset{0}{12}, \overset{1}{-6}, \overset{2}{19})$ prev = -(1<<31)



* **Code:**

```
def subseq ( arr, N, idx →0 , prev →-∞ ):
    if (idx == N):
        return 1

    if ( arr [idx] > = prev ):
    return subseq ( arr, N, idx+1, arr [idx]) +
        subseq ( arr, N, idx+1, prev ) → (not take)
    else:
    return subseq ( arr, N, idx+1 , prev ):
```

$\boxed{\begin{array}{l} T \cdot C = 2^N \\ S \cdot C = 1 \end{array}}$

* **Problem:** a_N : 12, 14, 15, 13, 20, 9, 7, 5, 6, 12, 17

→ Find max. len of subarray which can be rearranged in cont. order

→ Diff b/w elements = 1 , No Duplicates.

* **1st Approach:** 2 loops for i & j & sort each subarray & find the diff. & keep count of len (temp array)

Code:

```
ans = -(1<<31)
for i in range( len(arr)):
    for j in range( i, len(arr)):
        temp = []
        for k in range( i, len(arr)):
            temp.append ( arr[k])
        temp.sort()
        if check ( temp):
            ans = max( ans , j-i+1)
```

```
Check ( temp):
  for i in range ( len(temp)-1, 0, -1):
      if temp(i) - temp[i-1] ! = 1:
            return False
  return True.
```

$$\boxed{\begin{array}{l} T \cdot C = \overset{\text{subarrays}}{N^2} (\underset{\text{storing in temp}}{N} + \underset{\text{sort}}{n \log N} + \underset{\text{check}}{N}) \\ S \cdot C = N \leftarrow \text{temparray} \end{array}}$$

* **2nd Approach:** instead of sorting the temp array for 1 element perform insertion sort in jth loop to insert in the correct position

carry forward temp array

* **Code:**

```
ans = -(1<<31)
for i in range(len(arr)):
    for j in range(i, len(arr)):
        temp=[]
        insertion(temp, arr[j])
        if check(temp):
            ans = max(ans, j-i+1)
```

$$T.C = \overset{subarray}{N^2}(\overset{insertion}{N}+\overset{check}{N})$$

$$S.C = N$$

* **3rd Approach:** Instead of sorting, we perform a step where max(temp) - min(temp) + 1 = len(temp) → if we get the equal answer, it means the dist. b/w each elements is 1, as per the question no duplicates → so this will validate the check function for us.

* **Code:**

```
ans = -(1<<31)
for i in range(len(arr)):
    for j in range(i, len(arr)):
        mini= ∞ , maxi= -∞
        for k in range(i, len(arr)):
            min=min(mini, arr[k])
            maxi=max(maxi, arr[k])
        if (maxi-mini+1 == j-i+1):
            ans = max(ans, j-i+1)
```

$$T.C = \overset{subarrs}{N^2}(\overset{\text{kloop for mini, maxi}}{N})$$

$$S.C = 1$$

* **4th Approach:** instead of having kth loop, we can carry forward the min & max elements

```
ans = -(1<<31)
for i in range(len(arr)):
    mini = ∞ ,  maxi = -∞
    for j in range(i, len(arr)):
        mini = min(mini, arr[j])
        maxi = max(maxi, arr[j])
        if (maxi - mini + 1 == j - i + 1):
            ans = max(ans, j - i + 1)
```

T.C = $N^2(1)$

S.C = 1