day / date:

Algo Assignment 2
21K-3153

(Q1) ~~assuming that values~~

(a) (i) use merge sort (O(nlogn)) to sort the list
merge_sort(list, target)

(ii)

(1) position 2 pointers at the beginning and end of the list
left = 0
right = n-1

(3) add values/elements of the pointers. ~~If that sum is greater than~~
Make sure that positions of pointers are distinct.
sales[left] != sales[right].

add the values/elements
If the sum of elements is equal to the target, return true.

if arr[left] + arr[right] == target { return true }

if sum < target, increment left pointer
left ++

if sum > target, increment right pointer
right ++

Recurrence Relation:

$$T(n) = 2T(n/2) + O(n) + O(1) \rightarrow \text{pointer}$$

$2T(n/2)$ → merge sort

$2T\left(\frac{n}{2}\right) + n$

$n + 2T\left(\frac{n}{2}\right)$

$n + 2\left(\frac{n}{2} + 2T\left(\frac{n}{4}\right)\right)$

$2n + 4T\left(\frac{n}{4}\right) \rightarrow 2n + 4\left(\frac{n}{4} + 2T\left(\frac{n}{8}\right)\right)$

$3n + 8T\left(\frac{n}{8}\right)$

$kn + 2^k T\left(\frac{n}{2^k}\right)$

$2^k = n$

$k = \log n$

$n\log n + 2^{\log n}(1)$

$n\log n + n$

belongs to $O(n\log n)$

day / date:

6

(b) ① Use merge sort to sort ONE of the lists in ascending order (O(nlogn))

merge sort (list2, begin, end)

② Traverse every list 1 and subtract value of element from target (O(n))

for[i:n]:
    x = target - list1[i]

③ attempt to find "x" in list2 through binary search (O(logn)) which makes v traversing every element (n) * binary search (logn)
complexity =

→ nlogn

for[i:n]:
    x = target - list1[i] O(O(n))
binary search (list2, begin, end, x) O(logn)

if x is found, return true.
else return false.

Recurrence Relation

traversing

day / date:

$2T(\frac{n}{2}) + O(n) + O(n)\left(1 + T(\frac{n}{2}) \right)$ → binary search

merge sort:

$1 + 1 + T(\frac{n}{n})$

$2 + T(\frac{n}{n})$

$kn + 2^k T(\frac{n}{2^k})$

$cn\left(k + T(\frac{n}{2^k}) \right)$

$2^k = n$

$k = \log n$

$n \log n + n(1) + cn\left(\log n + T(\frac{n}{n}) \right)$

$n \log n + n + n \log n + c$

$2n \log n + n + c$

belongs to $\boxed{n \log n}$

day / date:

(Q 2)

It is understood that the word will always have odd elements as there can only be 1 unique letter

~~An approach similar to binary search will be used.~~

~~find mid.~~
~~Make sure mid is odd.~~
~~If mid even, move back~~

② ~~Compare [mid] to adjacent values.~~
~~If they match, move~~

① Find mid. Have 2 left & right pointers.
If mid even, move back to make it odd.

left = 0, right = n-1

```
findWord():
    if left < right:
        mid = ltr/2
        if mid % 2 == 1
            mid --
```

compare mid to adjacent values.
if left value is a match, increment left pointer by 2.
if right value is a match, decrement right pointer by 2.
isn't found.
if no match, return position of element

```
if arr[mid] == arr[mid-1]
    left += 2
if arr[mid] == arr[mid+1]
    right -= 2
else return mid
```

day / date:

Ans.

similar to binary search.

$T(n) = T(n/2) + 1$

$1 + (1 + T(\frac{n}{4}))$

$2 + T(\frac{n}{4})$ $=$ $2 + (1 + T(\frac{n}{8}))$

$3 + T(\frac{n}{8})$

$k + T(\frac{n}{2^k})$ $\quad 2^k = n$

$k = \log_2 n$

$\log n + T(1) = \boxed{\log n + 1} \rightarrow$ belongs to $\log n$

day / date:

(13) Divide Jars into 3 groups.

~~Start~~ ① func (arr, start, end):
if start == end
return start // heaviest
else:
p = (end - start + 1) / 3

② sum the weights of group 1 and 2, 2 and 3
Compare weights

x = sum (arr, start, start + p - 1)
y = sum (arr, start + p, start + 2p - 1)
z = sum (arr, start + 2p, start + 3p - 1)
Compare weights:
~~if x > y , heaviest in group 1 or 3~~

if x > y & x > z :
func (arr, start, start + p - 1) // 1st group
if y > x & y > z :
func (arr, start + p, start + 2p - 1) // 2nd group
else
func (arr, start + 2p, start + 3p - 1) // 3rd group.

by determining which group has heaviest jar,
we will recursively find it.

day / date:

$T(n) = T\left(\frac{n}{3}\right) + 1 \cdot T(1)$

↓ division in 3

$1 + \left(1 + T\left(\frac{n}{9}\right)\right)$

$2 + \left(1 + T\left(\frac{n}{27}\right)\right)$

$3 + T\left(\frac{n}{27}\right)$

$k + T\left(\frac{n}{3^k}\right)$

$3^k = n$

$k = \log_3 n$

$\log 3^n + T(1)$

↓

belongs to $O(\log_3 n)$

day / date:

Ternary:

0	1	2	3	4	5	6	7	8	9	10

mid1 = l + (r-l)/3

mid2 = r - (r-l)/3

key = 10

mid1 = 0 + (9-0)/3 = 3

mid2 = 9 - (9-0)/3 = 6

~~mid1 = 0 + (10-0)/3 = 4~~

~~mid2 = 10 - (10-0)/3 = 10-4 = 6~~

key > mid1 & key > mid2

10 > 3 10 > 7

newleft index = 7

left = mid2+1

right = 9

① newmid1 = 7 + (9-7)/3 = 7+1 = 8

newmid2 = 9 - (9-7)/3 = 9 - 1 = 8

key > mid1 & key mid2

③ newleft = mid2+1 = 8

right = 9

mid1 = 8 + (9-8)/3 = 8

mid2 = 9 - (9-8)/3 = 9 - 0 = 9

key > mid1 & key == mid2

return mid2

Meta binary:

day / date:

0	1	2	3	4	5	6	7	8	9
0	1	2	3	5	6	7	8	9	10

Key = 10

pos & val

① 0 000 A[0] != 10 & A[1]

pos

② 0001 ~~A[1] != 10~~ A[1] < 10 & ~~A[9]~~ 1 < 9

pos

③ 0011 ~~A[3] != 10~~ A[3] < 10 & 3 < 9

pos

④ 0111 ~~A[7] != 10~~ A[7] < 10 & 7 < 9

⑤ 1111 ~~A[9]~~ A[9] == 10 ✓

Ternary search reduces the time complexity of binary search as the search space is divided into 3 parts instead of 2 (hence $O(\log_3 N)$)

However, if target at extreme end, in worst case, could perform about the same amount of computations as Binary search

Meta Binary search perform fewer comparison if the target is close to 0 beginning of list.

If target close to the end, Meta Binary does more comparisons as you saw above

even more than ternary (5 iterations)

day / date:

Q5

① make an array of k+1 size, array X

② take n integers of 0 - k range as input.
With every integer, increment X[input] by 1.
This will count how many integers of value 'x' are in the inputted data.

0	1	2	3	. . . .	k
.	+1	+1			+1

③ do a prefix sum of the count array. This means add each previous element to current element for 0 . . k

0	1	2	3	. .	k
n	n+n	n+2n	n+3n		n+kn

④ for retrieval in O(1), simply take a and b and subtract their respective elements in the counting array (X[])

[b] - [a] will give numbers between b and a, b included.

([b] - [a]) - [b] will give numbers between a & b, with a and b excluded

day / date:

Traversing list of inputted integers is $O(n)$.

calculating prefix sum is $O(k)$

thus $O(n+k)$

day / date:

(b) ihad, abid, afif, fadi, adib, hadi, ibad

We can use bucket sort with our auxillary array being B[a ... i]
for our sorting algorithm, using insertion sort will be O(n) since probability of random variable falling into a bucket is O(1).

1	2 & 3	4 & 5	6 & 7
a	a → abid, afif	a → abid → afif → adib	a → abid, afif, adib
b	b	b	b
c	c	c	c
d	d	d	d
e	e	e	e
f	f	f → fadi	f → fadi
g	g	g	g
h	h	h	h → hadi
i → ihad	i → ihad	i → ihad	i → ihad → ibad

Sort B[i] using insertion (O(n))

a → abid, adib, afif
b
c
d
e
f → fadi
g
h → hadi
i → ibad, ibad

concatenate B[i]:

→ abid, adid, afif, fadi, hadi, ibad, ihad

O(n) for traversing original array
O(n) for insertion sort of B[i] lists
thus: O(n) + O(n) → O(n)