Date:

Algorithm Assignment

Samad Amir
21K-3873
BSE-5A

Question: 01 Problem: 01

```
Boolean Distict week(arr, target)
{

    sorted arr = mergesort (arr);
for( i=0; i< sorted arr.length; i++)
    {

    temp = target - sorted arr[i];
if binary Search (sorted arr, i, temp)
        return true
}
    return false
}
                    ,e,s
Merge Sort (arr) {
    if (arr. length <=1)
        return arr
int mid = arr.length/2
left[ ] = mergesort (arr, 0, mid)
right[ ] = mergesort(arr, mid arr.length);
    return merge (left, right)
}
```

Date:

```
[ ] Merge (left[], right[]) {
    result[ ] = new int [left.length + right.length];
    i=0, j=0, K=0;
while ( i < left.length && j < right.length)
    if ( left[i] < right[j]
        result[K++] = left[i++]
    else
        result[K++] = right[j++]
    }
while (i < left.length)
    result[K++] = left[i++]
while (j < right.length)
    result[K++] = right[i++]
return result;
}
boolean binarySearch (sortedArr, temp, excludeIndex)
    low = 0, high = sorted.length-1;
while (low ≤ high) {
    mid = (low + high)/2;
if (sortedArr[mid] = temp)
    if (mid != excludeIndex)
        return true;
if (mid > 0 && sortedArr[mid-1] == temp)
        return true
if (mid < sortedArr.length-1 && sortedArr[mid+1] == temp)
        return true
return false
else if (sortedArr[mid] < temp) => low = mid+1
```

Date:

```
else
    high = mid - 1
}
}
return false;
}
```

Recurrence Relation:-

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

Solving by Iteration:-

$2\left[2^0 T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n$

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

$\Rightarrow 2^3 T\left(\frac{n}{2^3}\right) + n + n + n$

$\vdots$

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

$n.T(1) + n\log n \quad \because T(1) = c$

$\Rightarrow n + n\log n$

↓ dominating value

$\Rightarrow O(n \log n)$

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

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

$2^2 \cdot 2^1\left[\frac{n}{2^3} + \frac{n}{2^2}\right] + n + n$

$\frac{n}{2^K} = 1$

$n = 2^K$

$\log n = K$

Date:

Problem 2.

Algorithm:- Step 01:-

```
Has same Pair (bakery1[], bakery2[], target) {
    sorted Bakery1[] = mergesort (bakery1);
    for (int i=0 ; i< bakery2.length; i++)
    {
        temp = target - bakery2[i]
        if (binary search (sorted bakery1, temp))
                return true;
        }
        return false;
    }
```

Step 02:- Merge sort

1. test base condition.
   if beg < End
2. Calculate Midpoint
   Mid = (Beg + End) / 2 => 3. Sort first subtable : Merge_sort(arr, beg, mid)
4. Recursively sort the second subtable
   Merge_sort(Arr, Mid+1, End)
5. Merge two ordered subtables
   Merge (arr, Beg, ~~End~~ Mid, End)
6. Finsh
   return.

Date:

Step 03:- Binary search

1- Compare the middle element with key

2- If key is not founded

- If key < mid, iterate and search left side
- if key > mid, iterate and search right side

3- Finish

return.

Question : 02.

Divide -and Conquer approach.

Algorithm:-

```
Search (arr, low, high)
    if (low > high)
        return -1;
    if (low == high)
        return low;
    mid = (low + high) / 2
if (mid % 2 != 0)
    mid --;
}
if (arr[mid] == arr[mid+1]) {
    return search (arr, mid+2, high);
else
    return search (arr, low, mid)
}
find Single Character (arr)
    return search( arr, 0, arr.length-1);
```

Date:

Recurrence Relation:-

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

$\Rightarrow T(\frac{n}{2^2}) + 2c$

$\Rightarrow T(\frac{n}{2^3}) + 3c$

$\Rightarrow T(\frac{n}{2^k}) + k.c \quad \Rightarrow \frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \log n = k$

$T(1) + \log n . c$

↓ dominating value

$\Rightarrow O(\log n)$

~~Heaviest jar (arr[ ], start~~

Question 03

Divide - Conquer approach

Algorithm.

findHeaviestjar (arr[ ], start, end)

if (start == end)

return start

third = (end - start + 1) / 3.

firstThird = Sum (arr, start, start + third - 1)

secondThird = Sum (arr, start + third, start + 2*third -

Date:

```
if (FirstThird > secondthird)
        return findheaviestjar(arr, start, start+third-1)
else if (firstThird < secondThird)
    return findheaviestjar(arr, start+third, start+2*third-1)
else
    findheaviestJar(arr, start+2*third, end);
}

Sum(arr, start, end)
 total = 0
 for(i=0 start; i<= end; i+1)
     total += arr[i]
 }
 return total
}
```

~~Recursive~~ => $O(\log_3 n)$

Approach:-

1- Divide array into thirds

2- Compare weights to see which is heavier and recursively look there

3- Keep narrowing unless we find the heaviest jar.

Question : 04

. Let's take an input of 10 number (Ternary Search)

3, 5, 7, 9, 11, 13, 15, 17, 19, 20 ∴ Key = 20

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

1- Divide the array in three parts. m_1, m_2

$m_1 = l + (r-l)/3$ ∴ $0 + (9-0)/3 = 3$.

$m_2 = r + (r-l)/3$ ∴ $9 + (9-0)/3 = 6$.

~~$m_1 = 9$, $m_2 = 15$~~. $arr[m_1] = 9$, $arr[m_2] = 15$.

3, 5, 7, 9, 11, 13, 15, 17, 19, 20

(m_1 = 9, m_2 = 15)

Since 20 > 15, key lies in final third.

then recursively add m_2 value

$l = m_2 + 1 = 6 + 1 = 7$.

- $arr[l] = 17 < 20$ -> add again
- $arr[l] = 19 < 20$ -> add again

$arr[l] = 20$, $arr[r] = 20$.

Search completed.

=> Since we are dividing array in third we can analyze complexity as

=> $O(\log_3 n)$.

Pros: . Since it is dividing array in three halves, its time complexity is much better. . Works well for large dataset . Number of comparison reduced . Space efficient

Cons: . can only be used on ordered list . Not easy implementation . Requires in depth knowledge of Recursion function . Not suitable for non-continuous

Date:

Meta Search

Let's take an input of 10 numbers

A = [3, 5, 7, 9, 11, 13, 15, 17, 19, 20]. search = 15

Step 01 :- Divide array in half and then compute/compare mid-value.

- Here middle element is 11 (can be 13 too)

  11 < 15 so,

  $\frac{15 - 11}{2} = 2$, we will skip 2 values and compare again

- after two elements we find our desired value 15.

∵ - Let's take 13 as middle value.

  since 13 < 15

  $\frac{15 - 13}{2} = 1$ check next element and we find our desired value.

Since we are checking only half, the time it takes is O(log n)

Pros :-

fewer comparision in some cases when target element is beginning of list

Cons :-

More comparison when target element is at the end of the list.

Question : 05

Algorithm:-

```
freqCount (arr, n)
{
  int max = Math.max(a, n)
  temp = new int [max + 1]
for (i = 0; i ≤ max; i++)
  {
    temp[i] = 0
  }
 for (i = 0; i < n; i++)
  {
    temp [a[i]] ++;
  }
 for (i = 1 ; i <= max ; i++) }
    temp[i] += temp[i-1]
  }
getCount (int a, int b) {
    if (a == 0) {
      return temp [b];
    }
    return temp [b] - temp [a-1]
  }
}
```

Date:

Example

Arr = [1, 3, 6, 5, 5, 6, 8, 9, 4] Range = [4, 6]

num	0	1	2	3	4	5	6	7	8	9
count	0	1	0	1	1	1	3	0	1	1

temp = [0, 1, 1, 2, 3, 4, 7, 7, 8, 9] => cumulative frequency

Step 01)

Find numbers that are less than or equal to upper bound

Here b = 6

~~[0, 1, 2, 3, 4, 6] are less than or equal to~~, temp[6] = 7

total = 7

Step 02 Find number less than or equal to a-1 lower bound, here a = 4

∴ a - 1 = 3

temp[3] = 2 => less than or equal to 3

Step 03

Calculate numbers within Range [4, 6]

Numbers in [4, 6] = temp[6] - temp[4]

=> 7 - 2 = 5

There 5 number lie in O(1)

Date:

Question: 06.

Use of Radix - Sort (LMS) left most significant

Input.

ihab abid afif fadi adib hadi ibad.

Step 01:-

Take left most value and keep smallest at top

abid
adib
afif
fadi
hadi
ihab
ibad.

step 02.

Now separate all the letters in subarray.

abid	fadi	hadi	ihab
adib			ibad
afif			

Step 03:-

Now Sort second LSB of each subarray

abid	fadi	hadi	ibad
adib			ihab
afif			

~~now~~ step 04:- Merge all

abid adib afif hadi ibad ihab

Since we are going through every value and check => O(n).