Team Members:
23K-0618 Areeba Arif
23K-0595 Wishmah Akhtar
23K-0642 Aqsa Zaib

Date ______ 20__
M T W T F S S

DAA PROJECT

Q1

a)

1. Pseudocode:

```
Algo FindMinMax(Arr, l, h)
Input: Array Arr with index l and h
Output: pair (min, max)
if l==h:
    return (Arr[l], Arr[l])
else if h==l+1:
    if Arr[l] < Arr[h]:
        return (Arr[l], Arr[h])
    else:
        return (Arr[h], Arr[l])
else:
    mid = (l+h)//2
    (min1, max1) = FindMinMax(Arr, l, mid)
    (min2, max2) = FindMinMax(Arr, mid+1, h)
    finalmin = min(min1, min2)
    finalmax = max(max1, max2)
    return (finalmin, finalmax)
```

2. $T(n)$ = no. of comparisions made for an array of size n

Base cases:

$n=1 \rightarrow 0$ comparisions

$n=2 \rightarrow 1$ comparisions

For $n>2$: divide into two subarrays of size $n/2$

$T(n) = T(n/2) + T(n/2) + 2 \rightarrow +2$ as two mins and two maxes compared

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

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right)+2$ since $T(n) = T\lceil n/2 \rceil + T\lfloor n/2 \rfloor + 2$

$\downarrow$

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right)+2$ bottom levels of recursion, when subarray have size 2, only 1 comparision happens per pair

23K-0618
23K-0595
23K-0642

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

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

$T(n) = 4\left[2T\left(\frac{n}{8}\right) + 2\right] + 4 + 2 = 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$

$\vdots$

$T(n) = 2^{k-1}T\left(\frac{n}{2^{k-1}}\right) + 2^{k-1} + 2^{k-2} + 2^{k-3} + \dots + 2$

$\frac{n}{2^{k-1}} = 2 \quad n = 2^k \quad k = \log n$

since $T(n/2^{k-1}) = T(2) = 0$ and $n = 2^k$

$T(n) = 2^{k-1}\,T(2) + 2^{k-1} + \dots + 2$

$T(n) = 2^{k-1} + 2^k - 2 = 3 \cdot 2^{k-1} - 2$

$T(n) \approx 1.5n - 2$

3. Brute-Force Algorithm:

```
Algo FindMinMax(Arr, n)
    min = Arr[0]
    max = Arr[0]
    for i=1 to n-1:              --> n-1
        if Arr[i] < min:         --> 1
            min = arr[i]
        if Arr[i] > max:         --> 1
            max = arr[i]
    return (min, max)
```

Loop runs $n-1$ times

Each iteration makes 2 comparisions

$T(n) = 2(n-1)$ = total comparisions.

Divide and conquer algorithm is more effecient than brute force bcz. $1.5n-2$ comparision are less than $2n-2$ comparisions.

23K-0618
23K-0595
23K-0642

Date ____ 20__
M T W T F S S

b)

1. Pseudocode

Algo Power (a,n)
Input: Base a, exponent n ($n>0$)
Output: a^n

```
if n==0:
    return 1
if n==1:
    return a
if n%2 ==0:
    half = Power(a, n/2)
    return half*half
else:
    half = Power(a, (n-1)/2)
    return a*half*half
```

2. $T(n)$ = no of multiplications used to compute a^n

$T(0) = 0$
$T(1) = 0$] Base cases

Recursive case:

If n is even: $T(n) = T(n/2) + 1$

If n is odd: $T(n) = T((n-1)/2) + 2$

$n = 2^k$

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

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

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

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

$$T(n) = T\left(\frac{n}{8}\right) + 1 + 1 + 1 = T\left(\frac{n}{8}\right) + 3$$

23K-0618
23K-0595
23K-0642

Date ____ 20 ___
M T W T F S S

$T(n) = T\left(\frac{n}{2^k}\right) + \log 2^k$ since $n = 2^k$ and $T(1) = 0$

$T(n) = T(1) + \log n$

$T(n) = 0 + \log n = \log_2(n) =$ no of multiplications

3. Brute Force Algorithm

Algo Power(a, n)
Input: Base a, power n (positive integer)
Output: a^n

```
result = 1
for i = 1 to n:          → n times
    result = result * a
return result
```

$T(n) = n$ times $= n$ number of ~~comparisons~~ multiplications

Divide and conquer algorithm is more effective since logn number of ~~comp~~ multiplications made by divide and conquer is less than n number of ~~compar~~ multiplications made by brute force algorithm.

23K0618
23K0595.
23K0642

Date ______ 20 ___
M T W T F S S

c) Algorithm Count Inversion (A[0 ⋯ n-1])

Input: Array A of n elements
Output: Number of Inversions in A

```
if n <= 1 then
    return 0

mid ← [n/2]
lefthalf ← A[0 ⋯ mid-1]
righthalf ← A[mid ⋯ n-1]

LeftInv ← countInv(lefthalf)
RightInv ← countInv(righthalf)
SplitInv ← Merge and Count (A, lefthalf, righthalf)

return LeftInv + RightInv + SplitInv
```

Algorithm Merge and Sort (A, Left[0 ⋯ p-1], Right[0 ⋯ q-1])
Input: Two sorted subarrays left and Right.
Output: Number of split inversions and merge sorted array in A

```
i ← 0 , j ← 0 , k ← 0
invCount ← 0
while i < p and j < q do
    if Left[i] <= Right[j] then
    A[k] ← Left[i]
       i ← i+1
else
     A[k] ← Right[j]
     j ← j+1
```

23K-0618
23K-0595
23K-0642

Date ______ 20
M T W T F S S

```
invCount ← invCount + (p- i)
  k ← k+ 1
whil i<p do

A[k] ← Left[i]
  i ← i+ 1
  k ← k+ 1
  while j<q  do

  A[k] ← Right [j]
        j ← j+1
        k ← k+1
return  inv Count.
```

23K-0618
23K-0595
23K-0642

Date 20
M T W T F S S

d) Best Case of Quick Sort.

$T(n) = 2T(n/2) + O(n)$ $\quad$ $f(n) = O(n^k \log^p n)$
$aT(n/b) + f(n)$

$k = 1$ $\quad$ $p = 0$

$\log_b a = \log_2 2$ $\quad$ $p > -1$

Case II $\log_b a = k$

$O(n^k \log^{p+1} n)$

$O(n \log n)$

Worst Case

$T(n) = T(n-1) + n$

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

$T(n-2) = T(n-3) + (n-2)$

$T(n-3) = T(n-4) + (n-3)$

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

$T(n) = T(n-3) + (n-2) + (n-1)$

$T(n) = T(n-4) + (n-3) + (n-2) + (n-1)$

$\vdots$

$T(n) = T(n-k) + (n-k+1) + (n-k+2) + (n-k+3)$

$n-k = 1$

$T(n) = T(1) + 2 + 3 + 4 + 5 - - - - + n$

$T(1) = 0$

$T(n) = 2 + 3 + 4 + 5 - - - - n$

$T(n) = \dfrac{n(n+1)}{2}$

$= O(n^2)$

23K-0618.
23K-0595
23K-0642

Date ____ 20
M T W T F S S

e) Pseudocode

Algorithm ClosestPair (A[1...n])

Sort A in increasing order
return closest (A,1,n)

Procedure closest (A, low, high)

if((high-low) = =1)

return |A[high] - A[low]|, |A[high - A[low]|,
A [low ⇒
|A[highh] - A low|]
~~[illegible]~~

mid = [(low+high/2]
d1 closest(A, low, mid)
~~closest(A, low, mid~~
d2 = closest (A, mid-1, high
d3 = | A [mid+1] - A[mid]

return min(d_1, d_2, d_3)

Time Complexity:
$$T(n) = 2T(n/2) + c$$
$$T(n) = aT(n/b) + f(n)$$
$a=2$, $b=2$ $f(n) = c = \Theta(\underline{1})$

$n^{\log_b a} = n^{\log_2 2} = n^1 = n$

Compare
$f(n) = \Theta(1)$ to $n^{\log_b a} = O(n)$
$f(n)$ is smaller.

23K0618
23K0595
23K0642.

Date ____ 20__
M T W T F S S

$T(n) = O(n)$

Sorting Step:- $O(n \log n)$

$T_{total} = O(n \log n) + O(n) = O(n \log n)$

2- Yes It is a good algorithm for this problem, it is efficient, correct and asymptotically optimal for finding the closest pair in one dimension.