Assignment 3

3.-1 a)

→ if array is sorted by relation

i.e $A[0 \ldots n-1]$ is true $\forall\ n \geq 0$.

Let break the array in two parts from i:

$A[0 \ldots i-1]$ & $A[i \ldots n-1]$.

$i = \frac{n}{2}$.

a Substitute the values.

$A[0 \ldots \frac{n}{2}-1]$ & $A[\frac{n}{2} \ldots n-1]$

function Minimum value $A[0 \ldots n]$ = Min Val. → $A[i]_{min}$

function Maximum value $A[0 \ldots n]$ = Max Val. → $A[i]_{max}$.

~~Procedure find $A[i]_{max}$.~~

Case 1.

if $A[i]_{min}$ & $A[i]_{max}$ both lie in either $A[0 \ldots \frac{n}{2}-1]$ or.

in $A[\frac{n}{2} \ldots n-1]$ then that part is not sorted

$A[i]_{min}, A[i]_{max} \in A[0 \ldots \frac{n}{2}-1]$ or $A[\frac{n}{2} \ldots n-1]$

then the array with $A[i]_{min}$ & $A[i]_{max}$ is unsorted.

ex → A 1 2 3 4 5 6 7 8

Same array ← A[7 & 2] ~~A[3,4,5,6]~~ A[3,4,5,6] → Sorted

In any case we can see that: $i-1$ i

$\forall n > 1, 0$ atleast one of the function $A[0 \ldots \frac{n}{2}-1]$ or $A[\frac{n}{2} \ldots n]$ will always be sorted

⇒ Condition when both the arrays are sorted.

$A[i]_{min} \in A[0 \ldots (\frac{n}{2}-1)]$

$A[i]_{max} \in A[\frac{n}{2} \ldots (n-1)]$

If the condition when $A[i]_{min}$ & $A[i]_{max} \in$ different Arrays that were sub divided, both the arrays would be sorted.

Eg: 1 2 3 4 5 6 7 8 → $A[i]_{max}$.

$A[1\ 2\ 3\ 4]$ $A[5\ 6\ 7\ 8]$

$A[i]_{min}$.

Basically the limits should belong to different Arrays.

3-1

(a) Algorith

```
Procedure : find min (A[l ... r])
    mid = (l+r)/2.
    if A[mid] > A[mid+1] then
        ans = a[mid+1]
    else if A[mid+1] > A[r] then
        ans = find min [(mid+1 ... r)]
    else if A[mid] < A[l] then
        ans = Find min (A[l .... mid])
    else
        ans A[l]
    end if
    return ans.
end procedure.
```

Recurrence Part of Algorith

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

Complexity to divide $O(1)$ i.e. computation of mid.

Complexity of Conquer $(n/2)$

$$T(n) = O(\log n)$$

By apply Case 2 of Master theorem $f(n) = n^{\log_2 1}$

$= O(n^0)$

$= O(1)$

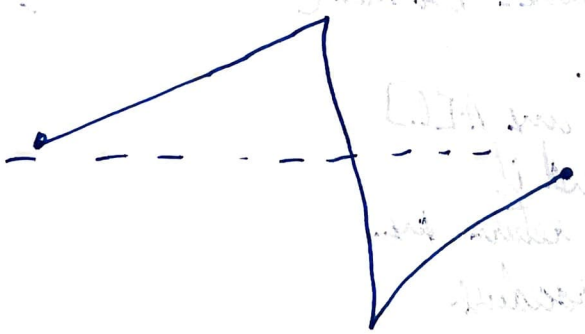
Cont.

Or it can be easily calculated by recursive tree method. ☆

There are $O(\log n)$ levels with complexity $O(1) = T_n = \log n$

Edge case :- If given array A. is of size 1. then the answer is only that element, else you can use find min (A[0 ... 6-1)]

Analysis :- In $C=0$ or $n/2$. algorithm runs in $O(1)$ & in other cases $O(\log n)$.



Use full property use

After dropping in all the values that are smaller than the drop.

Ex : 6 7 8 1 2 3 4 5

After 1 come after 8 all elements from 1 to 5 are smaller than 6.

☆ Used this property to divide the problem to other sub problems.

Question 3-2

(i) $z_i = \sum_{j=0}^{i} x_j y_{i-j}$

Now,

$$z = \sum_{i=0}^{2m-2} z_i \, 2^{\frac{in}{m}}$$

$$z = z_0 + z_1 2^{n/m} + z_2 2^{2n/m} + z_3 2^{3n/m} \ldots\ldots + z_{2m-2} 2^{(2m-2)n/m}$$

Now, z is a summation of $2m-1$ quantities. So; Summation takes $\theta(2m-1)$.

We need to pre-store $2^{n/m}$ which can be done in $\theta(\frac{n}{m})$

So the cost of combining the values is $\theta\left(2m-1+\frac{n}{m}\right)$

$$f(m,n) = \theta\left(2m-1+\frac{n}{m}\right)$$

∴ This function will be asymptotically equal to :-

$$\boxed{f(m,n) = \theta\left(2m+\frac{n}{m}\right)}$$

(ii) $T_m(n) = a_m T_m\left(\frac{n}{b}\right) + f_m(n)$

Now we need to calculate this function to z_i, $0 \leq i \leq 2m-2$ in $m \log n$ multiplications and additions can so produce z.

$$a_m = m \log n. \quad (\forall \text{ of } 2. \in R)$$

In the question, it is stated that the given problem is the sub problems $\frac{n}{m}$.

- In standard Karatsuba.
  3 multiplications, when $n=2$. $T_2(n) = 3T_2\left(\frac{n}{2}\right) + O(n)$

So. for all values of X & Y represented below.

where $X_0 \ldots X_{m-1}$ & $Y_0 \ldots Y_{m-1}$ to calculate z_0 $\forall c > 0$.

Also from (a) we know that the work of $CF=0 = f_m(n)'$.

$$f_m(n)' = \Theta\left(m + \frac{n}{m}\right) \quad \text{---- (1)}$$

Here to Calculate a function $z_0 \ldots z_{m-2}$, denoted by $Z = XY$ we can have the below equation.

$$T_m(n) = m \log_m T_m\left(\frac{n}{m}\right) + f(n).$$

for K bits in part $\frac{n}{m}$, deducing the value from (1)

$$\boxed{T_m(n) = m \log_m T_m\left(\frac{n}{m}\right) + \Theta\left(m + \frac{n}{m}\right).}$$

(c.) $\epsilon > 0$. (there exist a constant m independent of n)

From the equation, we know that

$$T(n) = n \log n \, T\left(\frac{n}{n}\right) + \Theta\left(n + \frac{n}{n}\right)$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$a = n \log n, \quad b = n.$$

$$f(n) = \Theta\left(n + \frac{n}{n}\right)$$

Now. $\log_b a = \log_n (n \log n)$

$$\log_b a = \log_n n + \log_n \log n.$$

$$= 1 + \log_n \log_2 n$$

Calculating for a base value at $f(n) = O(n)$

$$\text{So}, \quad f(n) = O\left(n^{\log_b a - k}\right), \quad k > 0$$

Applying Masters theorem

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

$$T(n) = \Theta\left(n^{1 + \log_n \log n}\right)$$

Let $\xi(n) = \log_n (\log n) \longrightarrow (I)$

Also, $\lim_{n \to \infty} \varepsilon(n) = 0$.

So we can deduce the below function at this via from I.

where $\boxed{T_n = \Theta\left(n^{1+\varepsilon(n)}\right)}$ for $\forall\, n > 0$.

(d) We will divide X into $\frac{n}{m}$ parts of size m bits each.

Problem statement

$$X = \sum_{i=0}^{\frac{n}{m}-1} x_i 2^{im}$$

$$Y = Y_0$$

Now $z_i = \sum x_i y_{i-j}$

but $y_k = 0$ for $k \geq 1$

So, $z_i = x_i y_0$.

$$Z = \sum_{i=0}^{\frac{n}{m}-1} z_i 2^{im}$$

Now, z_i, $0 \leq i \leq \frac{n}{m} - 1$ can calculate 2 in $\frac{n}{m}$ multiplication & addition

Also, we are dividing a problem of size n into subproblem of size m or $\frac{n}{n/m}$

So $Z = x_0 y_0 2^0 + x_1 y_0 2^m + x_2 y_0 2^m + \ldots + x_{\frac{n}{m}-1} y_0 2^{\left(\frac{n}{m}-1\right)m}$.

2 is a summation of $\frac{n}{m}$ terms

quantities which takes $O\left(\frac{n}{m}\right)$

Also, we need to pre calculate 2^m which will take $O(m)$

So, cost of combining the values will be $O\left(m + \frac{n}{m}\right)$

Therefor, we have.

$$T_n = \frac{n}{m} \cdot T\left(\frac{n}{n/m}\right) + O\left(m + \frac{n}{m}\right) \quad \text{---(I)}$$

$$T_n = a \log\left(\frac{n}{b}\right) + f(n)$$

$$a = \frac{n}{m}, \quad b = \frac{n}{m}, \quad f(n) = O\left(m + \frac{n}{m}\right)$$

$$\log_b a = \log_{\frac{n}{m}} \frac{n}{m} = 1.$$

Now, $f(n) = O(n) = O\left(n^{\log_b a}\right)$

So, by master theorem

$$T_n = O\left(n^{\log_b a} \log n\right)$$

$$T_n = O(n \log n)$$

This specific example can hence be taken to have a runtime better than directly applying Karatsuba, where runtime was $O(n^{\log_2 3}) \simeq n^{1.58}$

3-3

Q3(a) Let subarray A[start, ..., end] is a form $a^{x+1} b^{end-start-1-x-y} c^{y+1}$ where $x \geq 0$; $y \geq 0$ & atleast one of [x and $y \neq 0$] & $end - start - 1 - x - y > 0$ are some integers.

Then it is clearly visible that we can form x 'a's & y 'c's from from the corners to obtain a shorter span but it is given that [start, end] is the shortest span and hence our assumption is wrong. & there is only one number at corners. —(1)

Now we still have to prove that apart from the corners.

Arg [minSpan] can't contain numbers in between them

Assume. A[start, end] = a b .. b a b .. b c .. —(2)

From eq 2 it is clear that it contains some 'a' that it can be argued shorter span exists.

a b .. b [a b . b c] -> Span.

Similarly we can show existance of some c between $b^* (b^* c b^*)$ → (3).

Answer: ∴ our assumption is wrong that the given span is the shortest one.

3 (b). Note that $n/2$ is just a single point and span is the entire interval

Also, we know that this interval may or may not contain this point

So there only excists 3 possibilities only geometrical

1. If end $\leq \frac{n}{2}$ then interval starts before $\frac{n}{2}$ and ends on or before $n/2$.
2. If start $> n/2$ then interval starts after $n/2$. Hence this set of interval is completely disjoint from one above.

This implies that the interval completely lies on one side of point and don't contain point as intermediate or starting point.

3. So our third case follows such cases where start $\leq n/2$ & end $> \frac{n}{2}$.

Hence the above mentioned 3 cases cover all possible intervals that can excist and our span is in one of them interval

3(c) Procedure Part c(A[l.....m])

```
n =. m - l + 1.
if n ≤ 3 then
    return(∞, ∞).
end if
mid =. (l-1) + n/2;
i = mid - 1.;
j = mid + 2;
while i ≥ l do.
    if A[i] = A[mid] then
        break;
    end if
    i = i - 1.
end while.
while j ≤ m do;
    if A[j] != A[mid+1] then
        break;
    end if.
    j =. j + 1.
end while.
if A[mid] == A[mid+1] then
    if i == l-1. or j == m+1 then
        return (∞, ∞)
end if
    if A[i] == A[j] then
        return (∞, ∞)
end if
    return(i, j)
end if
```

① c —

② —

```
if i == mid + and a.(i) != A[mid+1] then
        return (i, mid+1)
end if.
if j = mid+2 and. A[j] != A mid then
        return (mid, j).
end if
return (∞,∞)
end procedure
```

③

∴ All cases covered.

Running time = O(N).

This is because. in total i & j pointer shift by maximum of n units during while loop & all the other statements are O(1).

Correctness of the argument above ↘.

if. A[mid] == A[mid+1]. is then

then we find the corner values. for making our structure and algorithm. ensure that i. & A[i] & A[j] exists and are other two number than A[mid]. ⟶ ①

if A[mid]a! =A[mid+1] then these point (a, b) must itself be the corner values. because multiple values cannot repeat at the centers. the above algorithm ensures by finding & forcing each of them corners. once & find corresponding corners, if they exist.

(d) Procedure Minimum Span (A[l ... m])

```
n = m - l + 1.
if n < 3 then
    return (∞, ∞)
end if
    mid = (l-1) + n/2.
X = Minimum Span (A[l ... mid])
Y = Minimum Span (A[mid+1 ... m)]
Z = Part C (A[l ... m])
```

return valid span with minimum cardinality from X, Y, Z.

end Procedure.

Correctness of Algorith.

⇒ Base Case. if $n < 3$ then there can be any sub arrays with all 3 elements.

⇒ Recursive case

Since from part (a) we know that based on mid we can separate set of all possible interval into disjoint set. → ①

Using ① we calculate our answer for X & Y for first & second halves. respectively.

Z is interval belonging to 3rd type of part (a) which is calculated by algorith for C.

Cont.

We return minimum span of all X, Y, and Z.

① Since our method division of subproblem is correct ⟹ (Part b)

② And our method for Conquer is correct by part c. for Z and for X and Y; also because the base case holds.

③ Finally, method to merge the result from recursive calls is correct

Hence we can prove that the overall algorithm is correct.

(e). $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$ → work of CEO (order of n).

- Complexity to divide is O(1)
- Conquering a subproblem is $2T(n/2)$ → (2 V.P & 2 subproblems)
- Complexity to merge needs to be calculated by the algorithm used in part c.

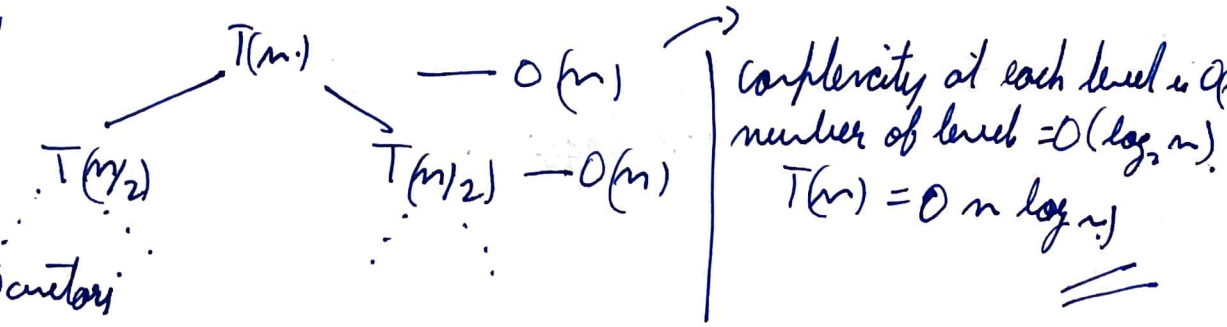
⇒ Since we know that the above recurrence is correct.

Applying Master theorem

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n).$$

we get complexity $T(n) = O(n \log n)$ ⟹

Recursion tree



Corrections → ↓

3(f). If we compute the length of prefix & suffix with equal values at corner of given sub-array then we can do work of part C in algorithm in O(1).

Assume 1.

Assume 2.

Hence our recurrence relation $\boxed{T(n) = 2T\left(\frac{n}{2}\right) + O(1)}$

Solution by recursion tree method $T(n) = (1)O$

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

We have $O(\log n)$ levels and level 'i' has 2^i complexity.

$$\therefore T(n) = \sum_{0}^{\lceil \log_2 n \rceil} 2^i$$

$$= 2^{\log_2 n} - 1.$$

$$= O(n). \quad \text{——} (1)$$

We could have also applied master theorem directly.

Changes in Part C work as instead of looping i and j we request to required position we can directly jump there to the position by simple arithmetic operation.

L> Logic.

⇒ Code.

Procedure ModifiedPart C (A[l .... m], int x, int y)

n = m - l + 1.

if n < 3.

return (∞, ∞).

mid = (l-1) + n/2.

i = mid - int x

j = mid + 1 + int y.

(rest of the code remains same)

end Procedure

n = m - l + 1.

if n < 3 then

calculate prefix & suffix length values let these be prefix & suffix respectively.

return ((∞, ∞), pref, suff)

end if.

mid = l - 1 + n/2.

X = PartF (A[l ... mid])

Y = PartF (A[mid+1 ... m])

Z = ModifiedPart C (A[l ... m], X.suff, Y.pref).

pref = X.pref;

if X.pref == mid - l + 1 & A[mid] == A[mid+1] then

pref = X.pref + Y.pref.

end if.

return (Valid span with minimum coordinate &ran.{x,y,z}),
fref, suff.);
end procedure.