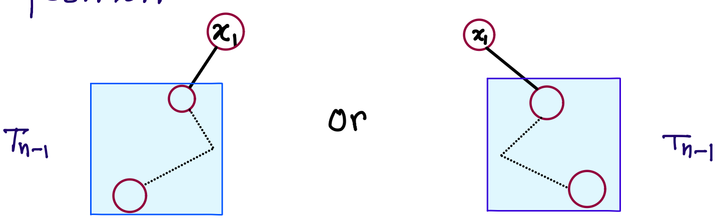
Homework-2

Design and Analysis of algorithm

Trishan Mondal

§ Problem 5

(i) Given numbers $\{x_1, \ldots, x_2\}$ we are making a BST for doing comparison on the basis of orders. At i^{th} step we can have atmost $i-1$ comparison, so totally atmost $\frac{n(n-1)}{2}$ comparison possible. In order to have an array where $\frac{n(n-1)}{2}$ comparison needed, we need exactly $i-1$ comparisn at i^{th} step for $i=1,2,\ldots,n$. If we look at BST diagram, it's not possible for any x_i have two branch as in the next step the element x_{i+2} will need atmost i comparison. Thus the BST will have one branch at each position.



We don't want two sided branching at x_1 at any step so, x_1 will be either $\max x_i$ or $\min x_i$. Let, T_n be the number of sequence have $n(n-1)/2$ comparison. After we have choosen x_1 we have to do same for $(x_2, \ldots, x_n)$, we will have the recurrence, $T_n = 2T_{n-1} \Rightarrow T_n = 2^{n-1}T_1$, For one element there is only option possible $T_n = 2^{n-1}$ as $T_1 = 1$.

(ii) $x_1, \dots, x_i$ are already Sorted in BST. If x_j get Compared to x_i, there is only two possible option for x_j, ① it Suffices all Condition to be Compared with x_{i-1} and $x_{i-1} < x_j < x_i$ (or, $x_i < x_j < x_{i-1}$ according to the x_i, x_{i+1} are Sorted ascending or descending order)

* ⑪ $x_{i-1} < x_i < x_j$ (or $x_j < x_i < x_{i-1}$ according to x_i, x_{i-1} are Sorted in ascending or descending order). So, x_j can fit at two gap.

(iii) Let, X be the random variable, Counts the number of Comparison for a Sequence $(x_1, \dots, x_n)$. Let, Ω be the Sample Space, $\Omega := \{(x_1, \dots, x_n)$ Sequences, all are distinct$\}$, $X : \Omega \to \mathbb{N}$

Let, X_i be the random variable defined as following,

$$X_i = \sum_{j>i} \mathbb{I}_{ij}$$

Where, $\mathbb{I}_{ij}$ is the indicator, $\mathbb{I}_{ij} = \begin{cases} 1 & \text{if } i \text{ is Compared to } j \\ 0 & \text{otherwise} \end{cases}$

* Now, $\underset{j>i}{\mathbb{E}}[\mathbb{I}_{ij}] = \mathbb{P}(\mathbb{I}_{ij} = 1) = \mathbb{P}(j \text{ is compared to } i)$

$$= \frac{2}{i+1} \qquad \text{(By part (ii))}$$

So, $\mathbb{E}[X_i] = \frac{(n-i)}{i+1}$ and hence, $\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{2(n-i)}{i+1}$.

$$\mathbb{E}[X] = \sum_{i=1}^{n} \frac{2(n-i)}{i+1}$$

$$= 2n \sum_{i=1}^{n} \frac{1}{i+1} - 2\sum_{i=1}^{n} 1 + 2\sum_{i=1}^{n} \frac{1}{i+1} = \sum_{i=1}^{n} \frac{2(n+1)}{i+1}$$

$$\leq 2(n+1)\log(n+1) - (2n)$$

$$= 2(n-1)\log(n+1) - (\underbrace{2n - 4\log(n+1)}_{>0 \ (\text{for } n \geq 2)})$$

$$\leq 2(n-1)\log(n+1)$$

■

§ Problem 6

(i)

```
1  def Search(A, B, l)
2          n = len(A)   m = len(B)
3          k = len(A)//2   r = len(B)//2
4  if  l = 0 :  return  min(A[0], B[0])
5  if   n == 0 :   return  B[l]
6  if   m == 0 :   return  A[l]
7     if  k+r < l:
8            if  A[k] < B[r]:
9         return  Search (A[k+1,...,n], B, l-k-1)
10           if  A[k] ≥ B[r]:
11        return  Search (A, B[r+1,...,m], l-r-1)
12    else  (# a+b ≥ l):
13           if  A[k] < B[r]:
14         return   Search (A, B[0,...,r], l)
15           else
16         return   search (A, B[0..k], B, l)
17
```

Correctness. Finding k^{th} largest element in array of $m+n$ number is equivalent to finding $(l+1)^{th}$ smallest number ($l = n-k$). Here we will proceed by induction in order to show correctness. Line 5-6 can be used as base case of the induction. In line 7, $k+r<l$ means the element we are looking for, comes after the halfway point of union $(A \cup B)$. If $A[k] < B[r]$ all the elements in $A[0,\dots,k]$ is smaller than $B[r]$, we get kth element occuring left of $\frac{m+n}{2}$. Thus we are calling Search for $(A[k+1,\dots,n], B, l-k-1)$. Hence, the k is reducing. From 12- we can see we are keeping k fix but reducing the array size, the algorithm will terminate. Since we are readjusting k accordingly by induction we can say our algorithm works correctly.

Time Complexity. total problem size is $m+n$. at line 9 it's reducing to size $\frac{n}{2}+m$ and at line 11 it's size is reducing to $n+\frac{m}{2}$. Similar thing is happening at line 14, line 16. Thus, combining them

$$\therefore \; T(2(m+n)) \leq T\left(\tfrac{3}{2}(m+n)\right)$$

$$\Rightarrow T(m+n) \leq T\left(\tfrac{3}{4}(m+n)\right)$$

Here, $\log_{\frac{4}{3}} 1 = 0$ and hence $T(m+n) \sim O(\log(m+n))$ this is $O(\log m + \log n)$ as it will be dominated by $\log m, \log n$ according to $m>n$ or $n>m$ which is the same case as $O(\log(m+n))$. ■

(ii) Let $A = (a_{ij})_{1 \leq i,j \leq n}$ be the matrix whose rows are sorted in ascending order. i.e $a_{ij} \leq a_{ik}$ for fixed i and $j<k$. Let, $B = (b_{ij})_{1 \leq i,j \leq n}$ be the matrix which we obtained after sorting the columns of A. Here, $b_{ik} \leq b_{jk}$ for $i \leq j$. Let, i be a number b/w 1 and m. We will show k^{th} row is sorted. b_{ij} is the i^{th} smallest number in j^{th} - column of B. We will get $a_{k_1 j}, \dots, a_{k_{n-i} j}$ in A such that $b_{ij} \leq a_{k_r j} \leq a_{k_r (j+1)}$, for $r = 1, \dots, n-i$, this follows from the fact rows of A are sorted. If one of elements $\{a_{k_1 j}, \dots, a_{k_{n-i} j}\}$ occurs in the set $\{b_{k(j+1)}\}_{k=1}^{i-1}$, then we have $b_{i(j+1)} \geq a_{k_\ell (j+1)} \geq b_{ij}$ (as colums are sorted). If no element of $\{a_{k_1 j}, \dots, a_{k_{n-i} j}\}$ occurs in $\{b_{k(j+1)}\}_{k=1}^{i-1}$, they must fall in rest $(n-i)$ part of $(j+1)^{th}$ row, and hence $b_{i(j+1)} = a_{k_\ell}(j+1) \geq b_{ij}$. Thus i^{th} row of B is sorted. ■

§ Problem 7

(i)

```
1  def   Majority (A )
2        n =  len (A[]), k=n//2
3        if  n==1 ,  return  A[1],
4        A_L = A[1,....,k] ,  A_R = A[k+1,...,n]
5        M_L= Majority (A_L), M_R = Majority (A_R)
6        if  M_L=M_R:
7              return  M_L
8        else
9              Count_M_L = 0 ,  Count_M_R = 0
10             for  i=1,..,n
11                  if A[i] == M_R
12                       Count_M_R ++
13             for  i= 1,...,n
14                  if  A[i] == M_L
15                       Count_M_L ++
16          if  Count_M_R > n/2
17               return  M_R
18          if  Count_M_L > n/2
19               return  M_L
20          else
                return  NULL
```

Correctness. In the above algorithm if the array length is 1 then we are returning the element $A[1]$, which is correct. We are splitting any array A of length n into two parth A_L, A_R of size $n/2$, then we are calling the function Majority, recursively it will again split the array into two part of size $n/4$. By induction, it will terminate and will return values M_L, M_R. From line 6-20, we have to check some statement, so the algorithm will terminate at finite step.

For an array A of $\text{len}(A) > 1$, we are

Sub dividing it at two part A_L and A_R. By induction let Majority $(A_L) = M_L$ and Majority $(A_R) = M_R$ are correct result.

M_L occurs $> \frac{n}{4}$ times in A_L and M_R occurs $> \frac{n}{4}$ times in A_R, hence if, $M_L = M_R$ it occurs $> n/2$ times in A. If $M_L \neq M_R$, the we are checking for how may i, $A[i] = M_L$, if that number $Count\_M_L > n/2$ then M_L is majority. Similar thing holds for M_R. Majority of A either will be M_L or M_R or A don't hav majority element. If M_A was the majority element then M_A must occur $> n/2$ times and M_L, M_R occurs $> n/4$ times, but then M_A, M_L, M_R occurs $> n$ times in total. Hence our algorithm is correct.

Time Complexity. Here we are dividing the problem of size n to two subproblem of size $n/2$. Equality checking loop 9-13 will take $\sim O(n)$ time thus.

$$T(n) = 2T(n/2) + O(n), \text{ with } T(1) = 1$$

By Master's theorem we have, $T(n) \sim O(n \log n)$.

(ii)

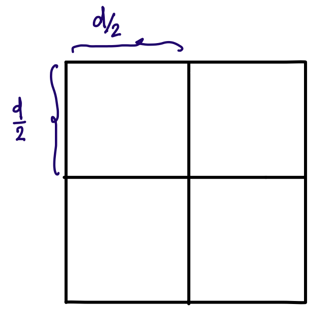
```
1  Input: an array A of size n≥1.
2  inarr = A[0],   proc = 0
3  for  0≤i≤n-1:
4          if  proc = 0:
5                  inarr = A[i]
6          else:
7              if  inarr = A[i]:
8                  proc ++
9              else
10                 proc --
11 Stab = 0
12       for  i=0,...,n-1:
13           if inarr = A[i]:
14               proc ++
15       if  2* proc > n:
16           return  inarr
17       else
18           return  NULL
```

Description and Correctness. The variable inarr and proc is initialized with A[0] and 0 respectively. If proc = 0 we set the element of A[0] as new value of inarr. Otherwise increament proc and decreament otherwise. After this we iterate over A, counting the actual number of occurences of inarr using the variable Stab. If inarr is not the majority element, it fails if 2* proc ≤ n. Then we return NULL.

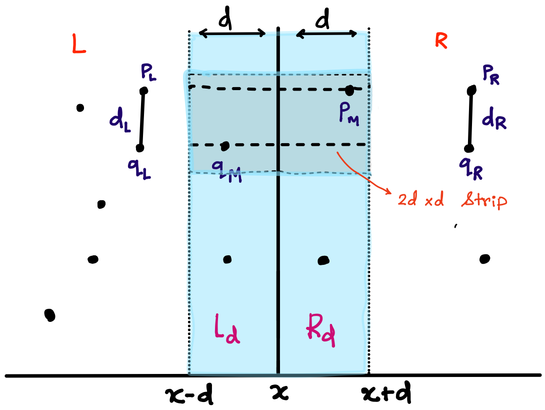
If there is a majority element x of A, it mus occur > $n/2$ times. If inarr != x at the end of the loop, proc updates to 0 for some value iff some other values occurs at least as many time. Which is not possible. So Cand = x, if majority element exist. This proves our algorithm is correct.

Time Complexity. For loop at line 3, 12 iterates n time after iteration finished we check, 2*Stab > n it takes ~ $(\log n)^2$ (for multiplication). The overall complexity is thus O(n). ■

§ Problem 8



(a) Let S be a $d \times d$ square, divide it into 4 equal sub-square. If the square contains 5 or more points of L, then by PHP at least two of these points will fell into one of the square of size $d/2 \times d/2$. The maximum distance b/w those points could be $\sqrt{2}\, d/2 = d/\sqrt{2} < d$. But $d = \min\{d_L, d_R\}$. This leads us to a contradiction. So any $d \times d$ square can contain atmost 4 points of L (same holds for R).



(b) **Correctness.** The algorithm is surely correct if d_L or d_R (according to picture) is minimum possible distance b/w any two pair of $L \cup R$. We need to check if the algorithm is correct when closest pair is P_M, q_M such that $P_M \in L$ and $q_M \in R$.

So, $dist(P_M, q_M) < d$. Thus P_M must lie in $L \cap \{(p,q) : x-d \le p \le x\} = L_d$ and q_M must lie in $R \cap \{(p,q) : x \le p \le x+d\} = R_d$. Let, $P_M = (x_p, y_p)$ and $q_M = (x_q, y_q)$. WLOG, assume $y_p < y_q$.

The way algorithm is given, if the algorithm do not return (P_M, q_M) as the "closest pair", there will exist 7 pairs $(x_1, y_1), \dots, (x_7, y_7)$ such that, $y_p < y_1 < \dots < y_7 < y_q$. Since, $d(P_M, q_M) < d$, Then The Points $P_M, (x_1, y_1), \dots, (x_7, y_7), q_M$.

will lie within a 2d x d strip. If we split 2d x d into two d x d square, at least 5 of the 9 point will lie in one of d x d square. 2d x d strip in $L_d \cup R_d$ has one d x d in L_d and another d x d in R_d. But it is not possible by Part (a)

(c)

This is the psudoCode for algorithm above.

```
def Closest_pair ( P= (X,Y) ):
    n = len(P)
    # primary cases
    if  n==2 :  dist(P)= dist(P[1], P[2])
                    return  P
    if  n==3 :
            d= min { d (P[1], P[2]), d(P[2], P[3]),
                     d (P[1]  P[3] )  }
        # d is the function measure distance  b/w
          two point  (x1,y1), (x2,y2).
            if  d (P[1], P[2]) == d : return (P[1], P[2])
            if  d (P[2], P[3]) == d: return ( P[2], P[3])
            if  d(P[3], P[1]) == d: return (P[3], P[1])
    # dividing the problem into Sub problem
    else :
        mid = n/2 ,     X_1 = Sort(X)   # here we
        are Sorting the array of X, by any sorting
        algorithm, the array X now has points in ascending
        order.
          PL = closest_pair (X_1[1,..., mid], Y)
          PR = Closest_pair (X_1[mid+1, ..., n], Y)
        dL = dist (PL), dR = dist (PR) , d = min (dL, dR)
    # Combining two Sub problem.
    S= the points of P whose X Coordinate lie
        in the range [x-d, x+d].
    Y_1 = Y co-ordinates of points in S.
    Y_2 = Sort (Y_1)
    s = len (S)
            for i 1 to s :
                for j   1 to 7
                    d1[j] = dist ((X,Y_2(i)), (X,Y_2(i+j)))
              d2[i] = min d1[j]
        d3 = min d2[i]
    if d3 > d :
            if dL = d  : return PL
            if dR = d  : return PR
    if d3 < d :
            PM = which Pair of points in S has dist (_,_) = d3
            return PM .
```

Time Complexity. Sorting in line 16, 27 needs $\sim n\log n$ time the codes from 29-32 need $\sqrt{n}$ multiplication which is atmost of order $\sim \sqrt{n}$ thus total time need for these calculation is $\sim n\log n$. We are dividing the problem of size n to two problem of size $n/2$.

$$T(n) \leq 2T(n/2) + O(n\log n)$$

$$\leq 2T(n/2) + c_1 n\log n \leq 2^2 T\left(\frac{n}{2^2}\right) + c_1 n\log n + c_2 \frac{n}{2}\log\frac{n}{2}$$

$$\vdots$$

$$\leq 2^{\log n} T(1) + c_1 n\log n + c_2 \frac{n}{2}\log\frac{n}{2} + \cdots + C_{\log n} \frac{n}{2^{\log n - 1}} \log\frac{n}{2^{\log n - 1}}$$

$$\sim n + (c_1 + c_2 + \cdots + c_{\log n}) n\log n$$

$$\sim c\, n(\log n)^2 + n \sim O(n\log n)$$

$\therefore$ Time Complexity $T(n) \sim O(n(\log n)^2)$. $\blacksquare$

(d)

See next Page !!

Problem 8 Part (d)

```
{ X_1 = sort (X)
{ Y_1 = sort (Y)
                                  # in this case we are
                                  sorting X and Y outside the
                                  function

def closest_pair ( P= (X,Y) ):
    n = len(P)
    # primary cases
    if  n==2 :  dist(P)= dist(P[1], P[2])
                    return  P
    if  n==3 :
            d= min { d(P[1], P[2]), d(P[2], P[3]),
                     d(P[1] P[3])  }
    # d is the function measure distance b/w
      two point (x1,y1), (x2,y2).
          if  d(P[1], P[2]) == d : return (P[1], P[2])
          if  d(P[2], P[3]) == d : return ( P[2], P[3])
          if  d(P[3], P[1]) == d : return (P[3], P[1])
   # dividing the problem into sub problem
   else :
      mid = n/2 ,                    # here we
      are sorting the array of X, by any sorting
      algorithm, the array X now has points in ascending
      order.
       P_L = closest_pair (X_1 [1,..., mid], Y)
       P_R = closest_pair (X_1[mid+1, ..., n], Y)
       d_L = dist(P_L) , d_R = dist(P_R) , d = min (d_L, d_R)
   # Combining two sub problem.
    S= the points of P whose X coordinate lie
       in the range  [x-d, x+d].
    Y_2 = Y_1 co-ordinates of points in S.

    s = len (S)
          for i 1 to s :
              for j  1 to 7
                 d_1[j] = dist ( (X, Y_2(i)), (X, Y_2(i+j)) )
          d_2[i] = min d_1[j]
       d_3 = min d_2[i]
   if d_3 > d :
          if  d_L = d  : return P_L
          if  d_R = d  : return P_R
   if d_3 < d :
          P_M =  which pair of points in S has dist (_,_) = d_3
          return  P_M .
```

Correctness of this algorithm is same as before, Since, X_1 is already sorted outside closest_pair, the recurrence would be $T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) \sim O(n\log n)$, sorting outside would take $\sim n\log n$ time. So total time complexity is $\sim O(n \log n)$ ∎

§Problem 9

(i) For this part let, $A(x) = \sum_{i=0}^{n-1} a_{n-1-i} x^i$ and, $B(x) = \sum_{\ell=0}^{n-1} b_i x^i$

We will check the coefficient of x^{n+j-1} in $C_j(x) = A(x)B(x)$.

This coefficient is, $\sum_{k=0}^{n-1} a_k b_{j+k}$. By the observation given in the question we can say a is contained in b at jth position if the coefficient $\sum_{k=0}^{n-1} a_k b_{j+k}$ is n.

(ii) If the k^{th} position of a has $*$ then we remove x^{n-1-k} part from $A(x)$ (it is the same polynomial constructed in previous part).

Take $B(x) = \sum_{\ell=0}^{n-1} b_i x^i$. If the coefficent of x^{n-1} is $n - m_*$ then a is contained in b at j^{th} position. (m_* is the total number of $*$ in a).

(iii) By taking $A(x)$, $B_j(x)$ and to multiply $A(x)$ and $B_j(x)$ by fast fourier transform (FFT) we need complexity, $\sim O(\max\{n,m\} \log(\max\{m,n\}))$ Since, $m>n$, this is $\sim \underline{O(m \log m)}$.

For each j we need to check coefficient of x^{n-1-j} in $C(x)$, we need to do n multiplication to determine C_{n-1-j} in $C(x)$. We have to do it for $m-n$ time so time complexity $O((m-n)n) \sim O(mn)$, it's better than FFT if n is small than $\log m$. ■

---

* Remark : Solution of problem 8 part (d) is mainly due to Deepta Basak, 7th problem part ii has been discussed with Aaratrick basu and I have shared/discussed some of solutions with Soumya Dasgupta, Priyatosh Jana.