CSE 548 Assignment 3

Divyansh Upreti (SBU ID: 112026646)

1 Prob 1- 3

Number of bank cards = n. (Each account can have many equivalent bank accounts associated with it)

Algorithm 1 majorityCardTest(N,0,n)

```
1: In this algorithm, we use divide and conquer to recursively divide the set of
   cards and check if a majority card is found at a level.
2: begin
3: if count(N)==1:
           return the card in N
4: if count(N)==2:
           if cards are equivalent:
                   return any one of the cards in N
5: N1 = majorityCardTest(N,0,n/2)
6: Check if returned card N1 forms majority relation with other cards at this
   level. If yes, return this card.
7: N2 = majorityCardTest(N,n/2,n)
8: Check if returned card N2 forms majority relation with other cards at this
   level. If yes, return this card
```

Time complexity = $\mathcal{O}(n * log(n))$, n = number of cards (there are n levels and the problem set is divided into 2 half sized array resulting in recurrence relation of T(n) = 2T(n/2)+O(n) which calculates to time complexity $\mathcal{O}(n * log(n)$

This algorithm is correct because the majority element will form majority with cards in either of the two levels of the input array. At each level we keep on forming a majority group and keep returning a card for this group, ultimately giving us a majority card if one is found.

2 Prob 1- 5

In this problem , we will start by sorting the edges in order of increasing slope. The first and last lines in this order when viewed from the top will always be visible. The second line will be visible if it intersects with the first line at a point to the left of where third and first lines meet. When we combine any two sub-solutions, we will find every intersection point and consider which solution has the uppermost line at this level.

Algorithm 2 visibleLines(n)

```
1: begin
2: if n $\leq$ 3:
       Return the visible line(s) (Trivial case).
3: Divide step: We divide the set of n lines into two groups and sort each
   line group as per the order of increasing slope and get two sets $line_1$ and
   $line_2$. Also we compute sequence of points $p_1$ and $p_2$ respectively where
   each element in both arrays denotes the intersection of consecutive lines in
   each line set considered independently. Both $p_1$ and $p_2$ will be sorted in
   increasing order of x coordinates.
4: Combine step: We combine $line_1$ and $line_2$ in such a way that a sorted set
   $line$ is obtained. Also, we merge the sorted lists $p_1$ and $p_2$ in to a single $p$
   list (in order of increasing x coordinate).
```

Proof of correctness: For each point b, we consider the lines uppermost in $line_1$ and $line_2$. If a is the smallest index for which uppermost line in 1st set is above than uppermost line in 2nd set and (x,y) denote the point in the plane where any pair of two lines from the each set intersect, then x will lie between x coordinate of a and a-1. And so, the sequence of sorted lines in set $line_1$ combined with sorted lines set $line_2$ together with point set p (sorted in increasing order) is valid solution at each level and thus the algorithm stands correct.
Time complexity = $\mathcal{O}(n * log(n))$, n = number of lines (As Time taken for sorting = $\mathcal{O}(n * log(n))$ and recursive calls and merging procedure as well will take $\mathcal{O}(n * log(n))$ time)

3 Prob 1- 6

The local minimum can be found using the following algorithm:

Algorithm 3 findLocalMinuimum(root)

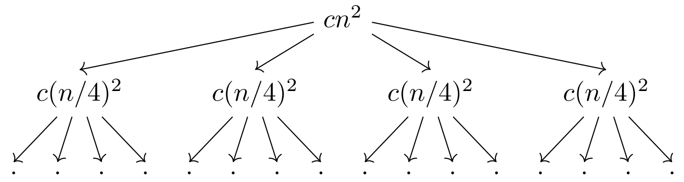
```
1: begin
2: Start from the root of the tree.
3: If root is smaller than its two children, root is a local minimum.
4: If root is smaller than a child, go to that child and iterate.
5: Repeat the above step with child as root till you reach a node n smaller
   than both children. Return this node n. If you reach a leaf l, then return
   this leaf l.
```

Time complexity = $\mathcal{O}(\log n)$, n = number of nodes in binary tree

Proof of correctness: If root is minimum, it is a local minimum and returned. If a node is smaller than its parent and its children, then it is correctly returned as the local minimum. If we reach leaf node, the leaf node will be smaller than its parent (that is why we ended up on leaf node) and will be returned correctly. Thus, the algorithm is correct.

4 Prob 2

1. $A(n) = 4A(\lfloor \frac{n}{2} \rfloor + 5) + n^2$
   Dropping the floor and constant terms, we get:
   $A(n) = 4A(\frac{n}{2}) + cn^2$
   Building the decision tree for the above recurrence:

   The height of the above tree= $\log_4 n$. From the above tree, we can see, that A(n) can be written as:

   A(n) = $\sum_i^n (4/16)^i cn^2$+$\theta(n^{\log_4 4})$

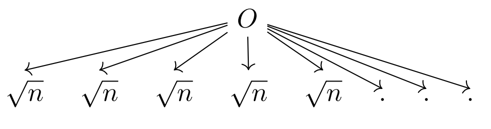
   Since $\sum_i^n (4/16)^i$ a G.P. entity that converges to 1, the time complexity will be given as follows:

   A(n) = $\theta(n^2)$

2. $B(n) = B(n-4) + 1/n + 5/(n^2+6) + 7n^2/(3n^3+8)$
   Since we are interested in finding the time complexity for large values of n, and for large values the algebraic expression will converge to a constant c.

   So, we will find for:
   $B(n) = B(n-4) + c$
   which evaluates to: $B(n) = \theta(n)$

3. C(n) = n+2$\sqrt{n}$C($\sqrt{n}$)

   For the above decision tree, as solved before:
   The recurrence relation will come out to be:

   C(n) = n.i+$2^i.C(n^{1/2^i})$,
   where i = $2^{\lg\lg n}$ (i being the total number of levels in the tree (depth))
   so, C(n) = n*$\theta(\lg\lg n)$

5 Prob 3

1. Let A =

$$\begin{pmatrix} p & q \\ r & s \end{pmatrix}$$

   A.A =

$$\begin{pmatrix} p & q \\ r & s \end{pmatrix}\begin{pmatrix} p & q \\ r & s \end{pmatrix}$$

   A.A =

$$\begin{pmatrix} (p*p)+(q*r) & (p*q)+(q*s) \\ (r*p)+(s*r) & (r*q)+(s*s) \end{pmatrix}$$

   =

$$\begin{pmatrix} (p*p)+(q*r) & q*(p+s) \\ r*(p+s) & (r*q)+(s*s) \end{pmatrix}$$

   Let A_1 = (p*p), $A2$ = (q*r), A_3 = q*(p+s), A_4 = r*(p+s), A_5 = (s*s). Therefore,

   A.A =

$$\begin{pmatrix} \mathrm{A}_1+A_2 & \mathrm{A}_3 \\ \mathrm{A}_4 & \mathrm{A}_2+A_5 \end{pmatrix}$$

   Hence 5 multiplications are sufficient to compute the square of a 2 X 2 matrix.

2. In Strassen's algorithm, let A =

$$\begin{pmatrix} P & Q \\ R & S \end{pmatrix}$$

   At some point. where each of these matrices is of size n/2 X n/2. Computing the square of A:

   A.A =

$$\begin{pmatrix} (P*P)+(Q*R) & (P*Q)+(Q*S) \\ (R*P)+(S*R) & (R*Q)+(S*S) \end{pmatrix}$$

   In this matrix, the only subproblem of size of 5 multiplications is P*P and S*S. In addition to this multiplication, the algorithm is computing for Q*R, P*Q, Q*S, R*P, S*R, R*Q. So, the given algorithm is not correct to have 5 multiplications and also suggested running time is incorrect.

3. (a) Given two matrices A and B: We can write AB + BA as $(A+B)^2$ - A^2 - B^2. In this, we see three matrix multiplications and if we consider the running time to be S(n). Then, adding the n^2 operations for performing additions and subtractions for n X n matrices A and B, the total running time will come out to be $3S(n)+\mathcal{O}(n^2)$

(b) For the given matrices A and B:
AB + BA =

$$\begin{pmatrix} 0 & XY \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & XY \\ 0 & 0 \end{pmatrix}$$

Thus, AB+BA has been expressed in terms of X and Y.

(c) We will use the result of (b) part as the value here. When we calculate the value of AB + BA, the non zero elelment that we get is XY and by (a) part of the problem, we can say that for 2n X 2n matrix size, the running time of the algorithm will be $3S(2n) + \mathcal{O}(n^2)$ which is equivalent to $\mathcal{O}(n^c)$ as S(n) = $\mathcal{O}(n^c)$.