José Douglas Gondim Soares, 485347

CANA - AV2 - Assíncrona

1- a)

// Estruturas de Dados

```
linha: {                    ponta: {
  int id;                     double x, y;
  double a;                 }
  double b;
}
```

```
1.  Procedimento acharAreaVisivel(L[1..n]) {
2.      mergesort(L[1..n]); // ordena linhas em ordem crescente de gradiente.
3.      lista <- [1..n]; int fim, i, j;
4.      ponta inicio <- [1..n], final <- [1..n], intersec;
5.      lista[0] <- 0; inicio[0] <- {-∞, -∞}; final[0] <- {+∞, +∞}
6.      fim <- 0;
7.      Para (i de 1 até N) {
8.          Enquanto (Verdade) {
9.              j <- lista[fim];
10.             intersec <- acharIntersecao(L[1..n], i, j);
11.             Se (intersec.x > inicio[fim].x) break;
12.             fim--;
13.         }
14.         final[fim] <- intersec; fim++;
15.         lista[fim] <- i; inicio[fim] <- intersec;
16.         final[fim] <- {+∞, +∞}
17.     }
18.     retorna lista;
19. }
```


CONTINUA

```
1. Algoritmo achar Intersecção (L[1..n], i, j){
2.     ponto I;
3.     Se(L[i].m == L[j].m){
4.         imprime('Erro. Linhas paralelas.');
5.         finaliza Programa();
6.     }Senão{
7.         I.x ← (L[j].b - L[i].b)/(L[i].a - L[j].a);
8.         I.y ← L[i].a * I.x + L[i].c;
9.         retorna I;
10.    }
11. }
```

b) Análise de tempo.

Algoritmo achar Intersecção

O procedimento interior não possui loops, apenas atribuições, logo, podemos considerar que executa em tempo constante $O(1)$.

Procedimento achar Area Visível

Linha 2 - temos o algoritmo mergesort que executa em tempo $O(n \log n)$. Isso já foi amplamente demonstrado.

Linhas [3-6] temos apenas atribuições, logo consideramos execução em tempo constante $O(1)$.

CONTINUA..

Linhas [7-8] - É fácil ver que o loop mais externo "linha 7" executa no máximo n vezes, já o Enquanto da linha 8 é fácil ver que mesmo no pior caso, a lista das linhas que compõem a área visível, já vai terá tamanho n. Isso nos informa de que os loops encadeados linhas 7 e 8 executa em tempo menor que $O(n^2)$.

Conclusão: É evidente que a linha mais custosa do procedimento é a linha do mergesort. Logo, temos um procedimento que executa em tempo máximo $O(n \cdot \log n)$.

C) Análise de corretude.

Obs: Vamos partir de que os algoritmos mergesort e achaIntersecção já estão corretos.

Variáveis i, j e fim

regras: $i \leq n$, j = última linha analisada., fim $\leq n$.

III -

No início $i=1$, $j=0$ e $fim=0$ ✓

IV - O for da linha 7 sempre garante que $i \leq n$. A linha 9 garante que j sempre aponta para a última linha analisada.
Fim sempre é decrementado em no máximo n vezes linha 12 e incrementa no máximo n vezes linha 14. ✓

Eu, José Douglas Grandim Soares, declaro que não pesquisei na internet, nem perguntei a colegas nada a respeito da prova acima.