UNIFEI - UNIVERSIDADE FEDERAL DE ITAJUBÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

SIN110 - ALGORITMOS E GRAFOS

RESOLUÇÃO DOS EXERCÍCIOS E10 DO DIA 23/10/2015

ITAJUBÁ

2015

Exercícios E10 – 23/10/15

Aluna: Karen Dantas **Número de matrícula:** 31243

1)

```
void uniao_inters (int *vet1,int *vet2, int *vet_u,int *vet_i, int m, int n){
    int verif, verif2, verif3;
    if (i==n)
        return;
    if (i<m){
        verif = busca_sequencial (vet2,n,vet1[i]);
        if (verif == -1){
            vet_i[pos_i]= vet1[i];
            pos_i++;
        }
        vet_u[pos_u]= vet1[i];
        pos_u++;
    }
    verif2 = busca_sequencial (vet1, m, vet2[i]);
    if (verif2 == -1){
        vet_i[pos_i]= vet2[i];
        pos_i++;
    }
    verif3 = busca_sequencial (vet1, m, vet2[i]);
    if (verif3 == -1){
        vet_u[pos_u]= vet2[i];
        pos_u++;
    }
    i++;
    uniao_inters (vet1,vet2, vet_u, vet_i, m, n);
}
```

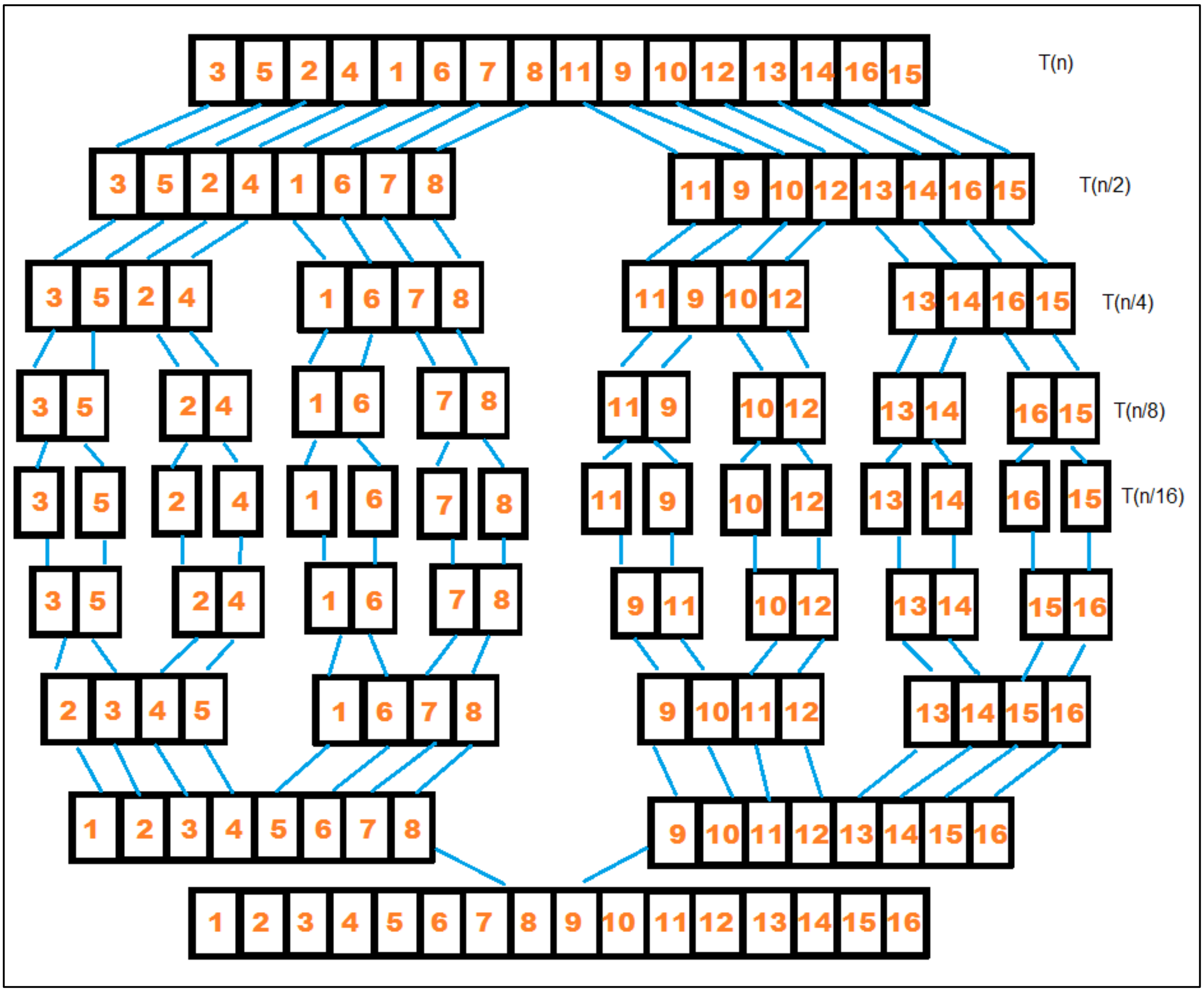
O algoritmo em linguagem C acima faz a união e interseção de dois conjuntos representados por dois vetores de tamanho ‘m’ e ‘n’ desconsiderando-se os valores repetidos e tratando o problema de os números não estarem ordenados utilizando-se a busca sequencial.

Complexidade: $T(n) = O(1)$, se $m=1$ e $n=2$

$T(n) = T(n) + O(n)$, se $m>1$ e $n>2$

Utilizando o teorema Mestre, identifiquei nessa equação o segundo caso analisado na equação geral de recorrências: $f(n) = af(n/b) + g(n)$, com $a = b^k$, com as constantes $a = 1$, $b = 1$ e $k = 1$, tendo como resultado: $T(n) = O(nlgn)$. Como o tamanho dos vetores é n e m obtêm-se $O(nlgm)$.

2) Árvore de recursão para o algoritmo MergeSort aplicado a um vetor de 16 elementos:



A técnica de programação dinâmica não é capaz de acelerar o algoritmo porque, apesar de ser um algoritmo recursivo e que utiliza o método dividir para conquistar, ele utiliza a função de intercalação que possui tempo linear a qual é chamada repetidas vezes para ir ordenando e fazendo o ‘merge’ dos vetores. Além do fato de que alguns números que anteriormente já foram comparados entre si podem ser comparados entre si novamente.

---

3) Os algoritmos recebem dois números e calculam o binômio de Newton.

a) Ao executar a primeira função nota-se que quanto maiores forem ‘m’ e ‘n’ a função será executadas muitas e muitas vezes pois a função divide-se em muitos subproblemas os quais gerarão somas sucessivas. Por exemplo, o resultado de (15 9) resulta em 5005, ou seja, serão realizadas 5005 somas sucessivas mais as verificações realizadas pela função na linha 1, logo, sua complexidade é exponencial: $O(2^n)$.

A complexidade da segunda função é $O(n^2)$, pois, na linha 5 tem um ciclo ‘para’ que está dentro do ciclo ‘para’ da linha 4.

b) A segunda função é a mais eficiente pois, apesar de possuir complexidade quadrática, para números muito grandes ela apresenta melhor desempenho se comparada à segunda função que possui complexidade exponencial.

---

4) <u>Algoritmo:</u>

```
Maximal (MAT, l, c, n, m)
max←0, i←1, j←1
se (linha=l e coluna=c)
        devolve MX_FINAL
senão
        se (cont = c){
                cont←0;
                coluna←1;
                poslinha← poslinha+1;
                linha←poslinha;
        enquanto (j<m)
                max← MAT[linha][coluna]+max
                MX[i][j]←MAT[linha][coluna]
                i←i+1
                linha←linha+1
                se (i = n-1){
                        i←1
                        linha←poslinha
                        coluna←coluna+1
                        j←j+1
        cont←cont+1
        coluna←cont
        linha←poslinha
        se (max>total)
                total ←max
                MX_FINAL← MX
        maximal(MAT, l, c, n, m)
```

Observação: As variáveis ‘linha’, ‘coluna’, e ‘poslinha’ são variáveis globais que inicialmente possuíam valor igual a um. E a variável global ‘total’ inicialmente possuía valor igual a zero.