```
//////////------------------------//////////
Relação de Recorrência MergeSort
T(n) = 2 * T(n/2) + T(n)
a = 2  / b = 2  / c = n

n = log 2 2
n = n¹

Sendo Caso 2 pelo Teorema mestre, logo -> n log n


T(n) = c24 + c25 + c26 + (n log n) + c27 + c28 + c29 + c30 + c31 + c32(n) + c33(n-1) + c34(n-1) + c35(n-1) + c36(n-1) + c37(n-1) + c38(n-1) + c39
T(n) = (c33 + c34 + c35 + c36 + c37 + c38)n  + (n log n) + (c24 + c25 + c26 + c27 + c28 + c29 + c30 + c31 + c39 - c33 - c34 - c35 -c36 - c37 - c38)

O(n log n) (Pior caso) --- base 2



//////////------------------------//////////

int resolver(pAuditorio pAuditorio) {                                   c24 * 1
  if (!pAuditorio) {                                                    c25 * 1
    return FALSE;                                                       c26 * 1
  }

  ordenar(pAuditorio->atividades, pAuditorio->maxTamanho);              1 * O (Theta)n log n
  int count = 0;                                                        c27 * 1

  Atividade atual = pAuditorio->atividades[0];                          c28 * 1
  atual.reservado = TRUE;                                               c29 * 1
  pAuditorio->atividades[0] = atual;                                    c30 * 1
  count++;                                                              c31 * 1

  for (int i = 1; i < pAuditorio->maxTamanho; i++) {                    c32 * (n)
    if (pAuditorio->atividades[i].inicio < atual.termino) {             c33 * (n-1)
      continue;                                                         c34 * (n-1)
    }

    atual = pAuditorio->atividades[i];                                  c35 * (n-1)
    atual.reservado = TRUE;                                             c36 * (n-1)
    pAuditorio->atividades[count] = atual;                              c37 * (n-1)
    count++;                                                            c38 * (n-1)
  }

  return TRUE;                                                          c39 * 1
}

void ordenar(Atividade* atividades, int tamanho) {
  merge_sort(atividades, 0, tamanho - 1);
```

```
50  }
51
52  void merge(Atividade* v, int l, int m, int r) {
53    int i, j, k;
54    int n1 = m - l + 1;
55    int n2 = r - m;
56    Atividade vl[n1];
57    Atividade vr[n2];
58
59    for (i = 0; i < n1; i++) {
60      vl[i] = v[l + i];
61    }
62    for (j = 0; j < n2; j++) {
63      vr[j] = v[m + j + 1];
64    }
65
66    i = 0;
67    j = 0;
68    k = l;
69    //contador while
70    while (i < n1 && j < n2) {
71      if (vl[i].termino <= vr[j].termino) {
72        v[k] = vl[i];
73        i++;
74      } else {
75        v[k] = vr[j];
76        j++;
77      }
78      k++;
79    }
80
81    while (i < n1) {
82      v[k] = vl[i];
83      i++;
84      k++;
85    }
86
87    while (j < n2) {
88      v[k] = vr[j];
89      j++;
90      k++;
91    }
92  }
93
94  void merge_sort(Atividade* v, int i, int f) {    c46 * 1
95    if (i < f) {                                   c47 * 1
96      int m = i + (f - i) / 2;                     c48 * 1
97      count
98      merge_sort(v, i, m);                         T(n/2)
99      count
100     merge_sort(v, m + 1, f);                     T(n/2)
```

```
101       count
102       merge(v, i, m, f);                          T(n)
103     }
104   }
105   /// T(n) = 2 * T(n/2) + T(n) Relação de Recorrência
106
107
108
109
```