算法 HSort

```
//快速排序的非递归算法，变量 M 已给定，5≤M≤15．该算法对文件（R1，R2，…，Rn）
//进行排序，R 包含 n+2 个记录，有效数据为 1,2,3,…,n, R[0]的关键词为 MinKey ,
//R[n+1]的关键词为 MaxKey
typedef struct{int x;int y;}stacktype;
void HSort(int n,Element* R,int M)
{
    Stack<stacktype> stackptr;
    stacktype temp;
    int f,t,j;
    temp.x=temp.y=0;
    stackptr.Push(temp);
    f=1;t=n;
    while (f<t)      //对长度大于或等于 M 的记录序列分划排序
    {
        j=Part(R,f,t);
        if( (j-f < M) && (t-j < M) )
        {
          temp=stackptr.Pop();
          f=temp.x;
          t=temp.y;
            continue;
        }
        if( (j-f < M) && (t-j >=M) )
        {
            f=j+1;
          continue;
        }
        if( (j-f >=M) && ( t-j < M) )
        {
            t=j-1;
            continue;
        }
        if( (j-f >=M) && (t-j >=M) )
        {
            if (j-f > t-j)
            {
                temp. x=f;
                temp. y=j-1;
                stackptr.Push(temp);
                f=j+1;
          }
            else
            {
                temp.x=j+1;
                temp.y=t;
                stackptr.Push(temp);
                t=j-1;
            }
        }
    }
    InsertSortA(R,n);              //插入排序
}
```

7.3.4 节 算法 Restore

```
//重建堆：重建树根为 tree[root]的二叉树，使之满足堆的特性。tree[root]的左、右子树是堆，
//且以 tree[root]为根的树中的任意结点，其编号均不大于 n。
void Restore ( Element *tree,const int root,const int n)
{
    int m;
    int j= root;
    while( j <=(int)(n/2))
     {
        if((2*j<n)&&(tree[2*j].GetKey()<tree[2*j+1].GetKey())) m=2*j+1;
        else m=2*j;
        if(tree[j].GetKey()<tree[m].GetKey())
```

```
            {
                Interchange(tree,j,m);     // 交换 tree 中下标为 j 和 m 的两个记录
                j=m;
            }
            else j= n;
        }
}
```