**Unity3D教程：手游开发常用排序算法**

Posted on 2013年06月24日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 71 次

**一、插入排序(Insertion Sort)**

1. 基本思想：

每次将一个待排序的数据元素，插入到前面已经排好序的数列中的适当位置，使数列依然有序；直到待排序数据元素全部插入完为止。

2. 排序过程：

【示例】：

[初始关键字] [49] 38 65 97 76 13 27 49

J=2(38) [38 49] 65 97 76 13 27 49

J=3(65) [38 49 65] 97 76 13 27 49

J=4(97) [38 49 65 97] 76 13 27 49

J=5(76) [38 49 65 76 97] 13 27 49

J=6(13) [13 38 49 65 76 97] 27 49

J=7(27) [13 27 38 49 65 76 97] 49

J=8(49) [13 27 38 49 49 65 76 97]

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | Procedure InsertSort(**Var** R : FileType); |
| 02 |  |
| 03 | *//对R[1..N]按递增序进行插入排序, R[0]是监视哨//* |
| 04 |  |
| 05 | Begin |
| 06 |  |
| 07 | **for** I := 2 To N **Do** *//依次插入R[2],...,R[n]//* |
| 08 |  |
| 09 | begin |
| 10 |  |
| 11 | R[0] := R[I]; J := I - 1; |
| 12 |  |
| 13 | **While** R[0] < R[J] **Do** *//查找R[I]的插入位置//* |
| 14 |  |
| 15 | begin |
| 16 |  |
| 17 | R[J+1] := R[J]; *//将大于R[I]的元素后移//* |
| 18 |  |
| 19 | J := J - 1 |
| 20 |  |
| 21 | end |
| 22 |  |
| 23 | R[J + 1] := R[0] ; *//插入R[I] //* |
| 24 |  |
| 25 | end |
| 26 |  |
| 27 | End; *//InsertSort //* |
| 28 |  |

**二、选择排序**

1. 基本思想：

每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排好序的数列的最后，直到全部待排序的数据元素排完。

2. 排序过程：

【示例】：

初始关键字 [49 38 65 97 76 13 27 49]

第一趟排序后 13 ［38 65 97 76 49 27 49]

第二趟排序后 13 27 ［65 97 76 49 38 49]

第三趟排序后 13 27 38 [97 76 49 65 49]

第四趟排序后 13 27 38 49 [49 97 65 76]

第五趟排序后 13 27 38 49 49 [97 97 76]

第六趟排序后 13 27 38 49 49 76 [76 97]

第七趟排序后 13 27 38 49 49 76 76 [ 97]

最后排序结果 13 27 38 49 49 76 76 97

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | Procedure SelectSort(**Var** R : FileType); *//对R[1..N]进行直接选择排序 //* |
| 02 |  |
| 03 | Begin |
| 04 |  |
| 05 | **for** I := 1 To N - 1 **Do** *//做N - 1趟选择排序//* |
| 06 |  |
| 07 | begin |
| 08 |  |
| 09 | K := I; |
| 10 |  |
| 11 | **For** J := I + 1 To N **Do** *//在当前无序区R[I..N]中选最小的元素R[K]//* |
| 12 |  |
| 13 | begin |
| 14 |  |
| 15 | **If** R[J] < R[K] Then K := J |
| 16 |  |
| 17 | end; |
| 18 |  |
| 19 | **If** K <> I Then *//交换R[I]和R[K] //* |
| 20 |  |
| 21 | begin Temp := R[I]; R[I] := R[K]; R[K] := Temp; end; |
| 22 |  |
| 23 | end |
| 24 |  |
| 25 | End; *//SelectSort //* |

**三、冒泡排序(BubbleSort)**

1. 基本思想：

两两比较待排序数据元素的大小，发现两个数据元素的次序相反时即进行交换，直到没有反序的数据元素为止。

2. 排序过程：

设想被排序的数组R［1..N］垂直竖立，将每个数据元素看作有重量的气泡，根据轻气泡不能在重气泡之下的原则，从下往上扫描数组R，凡扫描到违反本原则的轻气泡，就使其向上“漂浮”，如此反复进行，直至最后任何两个气泡都是轻者在上，重者在下为止。

【示例】：

49 13 13 13 13 13 13 13

38 49 27 27 27 27 27 27

65 38 49 38 38 38 38 38

97 65 38 49 49 49 49 49

76 97 65 49 49 49 49 49

13 76 97 65 65 65 65 65

27 27 76 97 76 76 76 76

49 49 49 76 97 97 97 97

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | Procedure BubbleSort(**Var** R : FileType) *//从下往上扫描的起泡排序//* |
| 02 |  |
| 03 | Begin |
| 04 |  |
| 05 | **For** I := 1 To N-1 **Do** *//做N-1趟排序// //Unity3D教程手册* |
| 06 |  |
| 07 | begin |
| 08 |  |
| 09 | NoSwap := **True**; *//置未排序的标志//* |
| 10 |  |
| 11 | **For** J := N - 1 DownTo 1 **Do** *//从底部往上扫描//* |
| 12 |  |
| 13 | begin |
| 14 |  |
| 15 | **If** R[J+1]< R[J] Then *//交换元素//* |
| 16 |  |
| 17 | begin |
| 18 |  |
| 19 | Temp := R[J+1]; R[J+1 := R[J]; R[J] := Temp; |
| 20 |  |
| 21 | NoSwap := **False** |
| 22 |  |
| 23 | end; |
| 24 |  |
| 25 | end; |
| 26 |  |
| 27 | **If** NoSwap Then **Return***//本趟排序中未发生交换，则终止算法//* |
| 28 |  |
| 29 | end |
| 30 |  |
| 31 | End; *//BubbleSort//* |

**四、快速排序（Quick Sort）**

1. 基本思想：

在当前无序区R[1..H]中任取一个数据元素作为比较的“基准”(不妨记为X)，用此基准将当前无序区划分为左右两个较小的无序区：R[1..I-1]和R[I+1..H]，且左边的无序子区中数据元素均小于等于基准元素，右边的无序子区中数据元素均大于等于基准元素，而基准X则位于最终排序的位置上，即R[1..I-1]≤X.Key≤R[I+1..H](1≤I≤H)，当R[1..I-1]和R[I+1..H]均非空时，分别对它们进行上述的划分过程，直至所有无序子区中的数据元素均已排序为止。

2. 排序过程：

【示例】：

初始关键字 [49 38 65 97 76 13 27 49］

第一次交换后 ［27 38 65 97 76 13 49 49］

第二次交换后 ［27 38 49 97 76 13 65 49］

J向左扫描，位置不变，第三次交换后 ［27 38 13 97 76 49 65 49］

I向右扫描，位置不变，第四次交换后 ［27 38 13 49 76 97 65 49］

J向左扫描 ［27 38 13 49 76 97 65 49］

（一次划分过程）

初始关键字 ［49 38 65 97 76 13 27 49］

一趟排序之后 ［27 38 13］ 49 ［76 97 65 49］

二趟排序之后 ［13］ 27 ［38］ 49 ［49 65］76 ［97］

三趟排序之后 13 27 38 49 49 ［65］76 97

最后的排序结果 13 27 38 49 49 65 76 97

各趟排序之后的状态

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | Procedure Parttion(**Var** R : FileType; L, H : Integer; **Var** I : Integer); |
| 02 |  |
| 03 | *//对无序区R[1,H]做划分，I给以出本次划分后已被定位的基准元素的位置 //* |
| 04 |  |
| 05 | Begin |
| 06 |  |
| 07 | I := 1; J := H; X := R[I] ;*//初始化，X为基准//* |
| 08 |  |
| 09 | Repeat |
| 10 |  |
| 11 | **While** (R[J] >= X) And (I < J) **Do** |
| 12 |  |
| 13 | begin |
| 14 |  |
| 15 | J := J - 1 *//从右向左扫描，查找第1个小于 X的元素//* |
| 16 |  |
| 17 | **If** I < J Then *//已找到R[J] 〈X//* |
| 18 |  |
| 19 | begin *//Unity3D教程手册：www.unitymanual.com* |
| 20 |  |
| 21 | R[I] := R[J]; *//相当于交换R[I]和R[J]//* |
| 22 |  |
| 23 | I := I + 1 |
| 24 |  |
| 25 | end; |
| 26 |  |
| 27 | **While** (R[I] <= X) And (I < J) **Do** |
| 28 |  |
| 29 | I := I + 1 *//从左向右扫描，查找第1个大于 X的元素///* |
| 30 |  |
| 31 | end; |
| 32 |  |
| 33 | **If** I < J Then *//已找到R[I] > X //* |
| 34 |  |
| 35 | begin R[J] := R[I]; *//相当于交换R[I]和R[J]//* |
| 36 |  |
| 37 | J := J - 1 |
| 38 |  |
| 39 | end |
| 40 |  |
| 41 | Until I = J; |
| 42 |  |
| 43 | R[I] := X *//基准X已被最终定位//* |
| 44 |  |
| 45 | End; *//Parttion //* |
| 46 |  |
| 47 | Procedure QuickSort(**Var** R :FileType; S,T: Integer); *//对R[S..T]快速排序//* |
| 48 |  |
| 49 | Begin |
| 50 |  |
| 51 | **If** S < T Then *//当R[S..T]为空或只有一个元素是无需排序//* |
| 52 |  |
| 53 | begin |
| 54 |  |
| 55 | Partion(R, S, T, I); *//对R[S..T]做划分//* |
| 56 |  |
| 57 | QuickSort(R, S, I-1);*//递归处理左区间R[S,I-1]//* |
| 58 |  |
| 59 | QuickSort(R, I+1,T);*//递归处理右区间R[I+1..T] //* |
| 60 |  |
| 61 | end; |
| 62 |  |
| 63 | End; *//QuickSort//* |

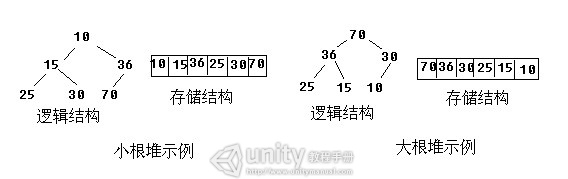
**五、堆排序(Heap Sort)**

1. 基本思想：

堆排序是一树形选择排序，在排序过程中，将R[1..N]看成是一颗完全二叉树的顺序存储结构，利用完全二叉树中双亲结点和孩子结点之间的内在关系来选择最小的元素。

2. 堆的定义: N个元素的序列K1,K2,K3,...,Kn.称为堆，当且仅当该序列满足特性：

Ki≤K2i Ki ≤K2i+1(1≤ I≤ [N/2])

[](http://www.unitymanual.com/wp-content/uploads/2013/06/128.jpg)

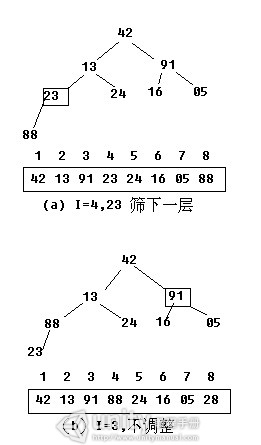
Unity3D教程：手游开发常用排序算法

堆实质上是满足如下性质的完全二叉树：树中任一非叶子结点的关键字均大于等于其孩子结点的关键字。例如序列10,15,56,25,30,70就是一个堆，它对应的完全二叉树如上图所示。这种堆中根结点（称为堆顶）的关键字最小，我们把它称为小根堆。反之，若完全二叉树中任一非叶子结点的关键字均大于等于其孩子的关键字，则称之为大根堆。

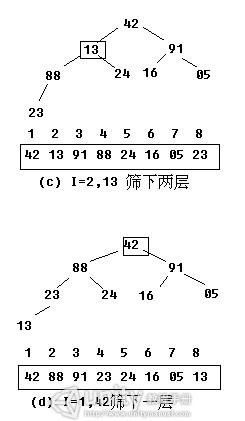
3. 排序过程：

堆排序正是利用小根堆（或大根堆）来选取当前无序区中关键字小（或最大）的记录实现排序的。我们不妨利用大根堆来排序。每一趟排序的基本操作是：将当前无序区调整为一个大根堆，选取关键字最大的堆顶记录，将它和无序区中的最后一个记录交换。这样，正好和直接选择排序相反，有序区是在原记录区的尾部形成并逐步向前扩大到整个记录区。

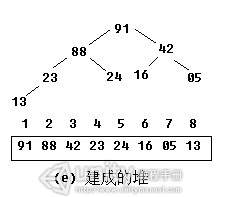
【示例】：对关键字序列42，13，91，23，24，16，05，88建堆

[](http://www.unitymanual.com/wp-content/uploads/2013/06/217.jpg)

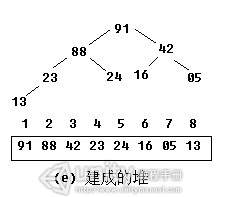
Unity3D教程：手游开发常用排序算法

[](http://www.unitymanual.com/wp-content/uploads/2013/06/312.jpg)

Unity3D教程：手游开发常用排序算法

[](http://www.unitymanual.com/wp-content/uploads/2013/06/48.jpg)

Unity3D教程：手游开发常用排序算法

[](http://www.unitymanual.com/wp-content/uploads/2013/06/48.jpg)

Unity3D教程：手游开发常用排序算法

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | Procedure Sift(**Var** R :FileType; I, M : Integer); |
| 02 | *//在数组R[I..M]中调用R[I]，使得以它为完全二叉树构成堆。事先已知其左、右子树(2I+1 <=M时)均是堆//* |
| 03 | Begin |
| 04 | X := R[I]; J := 2\*I; *//若J <=M, R[J]是R[I]的左孩子//* |
| 05 | **While** J <= M **Do** *//若当前被调整结点R[I]有左孩子R[J]//* |
| 06 | begin |
| 07 | **If** (J < M) And R[J].Key < R[J+1].Key Then |
| 08 | J := J + 1 *//令J指向关键字较大的右孩子//* |
| 09 | *//J指向R[I]的左、右孩子中关键字较大者//* |
| 10 | **If** X.Key < R[J].Key Then *//孩子结点关键字较大//* |
| 11 | begin |
| 12 | R[I] := R[J]; *//将R[J]换到双亲位置上//* |
| 13 | I := J ; J := 2\*I *//继续以R[J]为当前被调整结点往下层调整//* |
| 14 | end; |
| 15 | **Else** |
| 16 | Exit*//调整完毕，退出循环//* |
| 17 | end |
| 18 | R[I] := X;*//将最初被调整的结点放入正确位置//* |
| 19 | End;*//Sift//* |
| 20 |  |
| 21 | Procedure HeapSort(**Var** R : FileType); *//对R[1..N]进行堆排序//* |
| 22 | Begin |
| 23 | **For** I := N Div Downto 1 **Do** *//建立初始堆//* |
| 24 | Sift(R, I , N) |
| 25 | **For** I := N Downto 2 **do** *//进行N-1趟排序//* |
| 26 | begin |
| 27 | T := R[1]; R[1] := R[I]; R[I] := T;*//将当前堆顶记录和堆中最后一个记录交换//* |
| 28 | Sift(R, 1, I-1) *//将R[1..I-1]重成堆//* |
| 29 | end |
| 30 | End; *//HeapSort//* |

六、几种排序算法的比较和选择

1. 选取排序方法需要考虑的因素：

(1) 待排序的元素数目n；

(2) 元素本身信息量的大小；

(3) 关键字的结构及其分布情况；

(4) 语言工具的条件，辅助空间的大小等。

2. 小结：

(1) 若n较小(n <= 50)，则可以采用直接插入排序或直接选择排序。由于直接插入排序所需的记录移动操作较直接选择排序多，因而当记录本身信息量较大时，用直接选择排序较好。

(2) 若文件的初始状态已按关键字基本有序，则选用直接插入或冒泡排序为宜。

(3) 若n较大，则应采用时间复杂度为O(nlog2n)的排序方法：快速排序、堆排序或归并排序。 快速排序是目前基于比较的内部排序法中被认为是最好的方法。

(4) 在基于比较排序方法中，每次比较两个关键字的大小之后，仅仅出现两种可能的转移，因此可以用一棵二叉树来描述比较判定过程，由此可以证明：当文件的n个关键字随机分布时，任何借助于“比较”的排序算法，至少需要O(nlog2n)的时间。

(5) 当记录本身信息量较大时，为避免耗费大量时间移动记录，可以用链表作为存储结构。