1、． 以关键码序列(503,087,512,061,908,170,897,275,653,426)为例，手工执行以下排序算法，写出每一趟排序结束时的关键码状态：

(1)直接插入排序；(2)希尔排序(增量 d[1]=5)；(3)快速排序；(4)堆排序

解：

（1） 直接插入排序

STEP1:

[503], {087, 512, 061, 908, 170, 897, 275, 653, 426}

STEP2:

[087, 503], {512, 061, 908, 170, 897, 275, 653, 426}

STEP3:

[087, 503, 512], {061, 908, 170, 897, 275, 653, 426}

STEP4:

[061, 087, 503, 512], {908, 170, 897, 275, 653, 426}

STEP5:

[061, 087, 503, 512, 908], {170, 897, 275, 653, 426}

STEP6:

[061, 087, 170, 503, 512, 908], {897, 275, 653, 426}

STEP7:

[061, 087, 170, 503, 512, 897, 908], {275, 653, 426}

STEP8:

[061, 087, 170, 275, 503, 512, 897, 908], {653, 426}

STEP9:

[061, 087, 170, 275, 503, 512, 653, 897, 908], {426}

STEP10:

[061, 087, 170, 275, 426, 503, 512, 653, 897, 908]

（2） 希尔排序

STEP1(DK = 5):

FROM: 503 087 512 061 908 170 897 275 653 426

TO : 170 087 275 061 426 503 897 512 653 908

STEP2(DK = 3)

FROM: 170 087 275 061 426 503 897 512 653 908

TO : 061 087 275 170 426 503 897 512 653 908

STEP3(DK = 2)

FROM: 061 087 275 170 426 503 897 512 653 908

TO : 061 087 275 170 426 503 653 512 897 908

STEP4(DK = 1)

FROM: 061 087 275 170 426 503 653 512 897 908

TO : 061 087 170 275 426 503 653 512 897 908

（3） 快速排序

取第一个关键码 503 为关键字：

第一次排序

STEP1:

426	087	512	061	908	170	897	275	653	426
i									j

STEP2:

426	087	512	061	908	170	897	275	653	426
	i								j

STEP3:

426	087	512	061	908	170	897	275	653	512
	i							j	

STEP4:

426	087	512	061	908	170	897	275	653	512
	i						j		

STEP5:

426	087	275	061	908	170	897	275	653	512
		i					j		

STEP6:

426	087	275	061	908	170	897	275	653	512
			i				j		

STEP7:

426	087	275	061	908	170	897	908	653	512
			i			j			

STEP8:

426	087	275	061	908	170	897	908	653	512
			i	j					

STEP9:

426	087	275	061	170	503	897	908	653	512
				ij					

对其左子序进行排序 426 087 275 061 170（参照上述步骤可得）

087 275 061 170 426

进行递归处理得到

061 087 170 275 426

同理可得右子序 897 908 653 512 进行递归排序后得

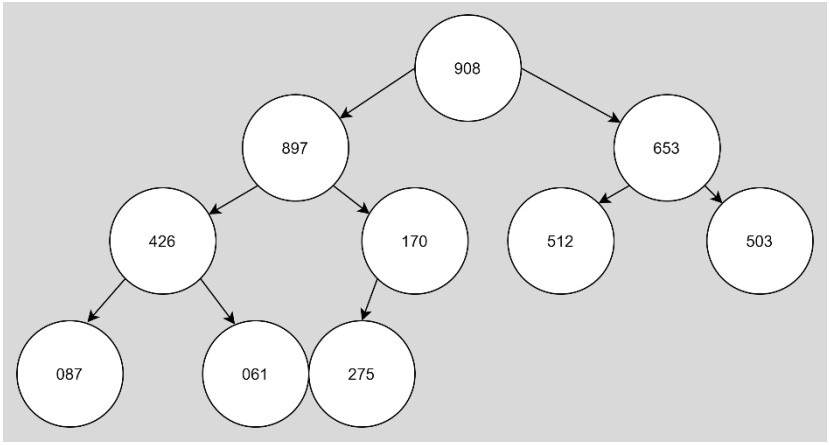
512 653 897 908

由此整个排序已完成

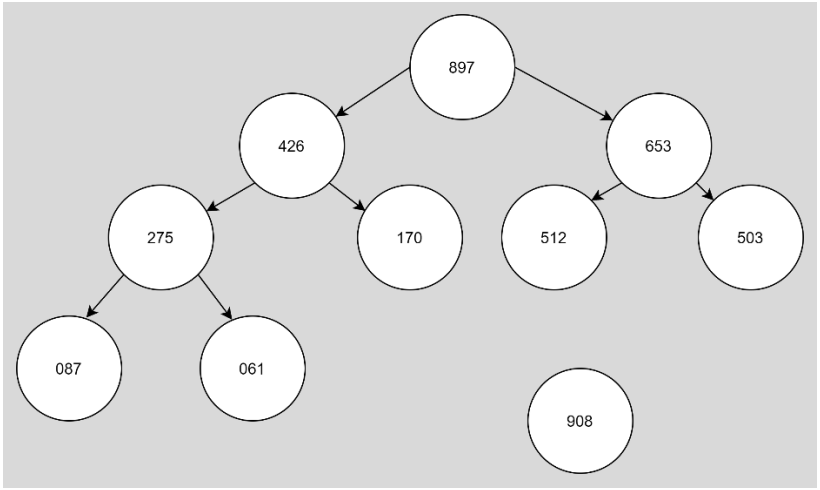
061 087 170 275 426 512 061 087 170 275 426

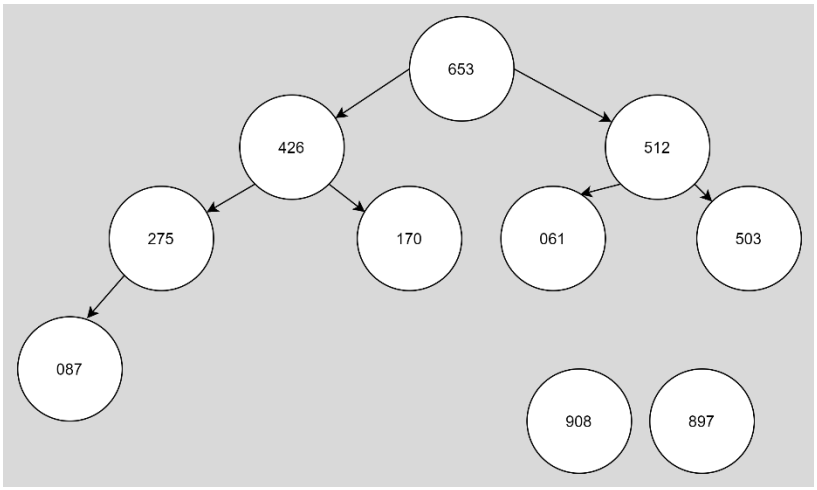
（4） 堆排序

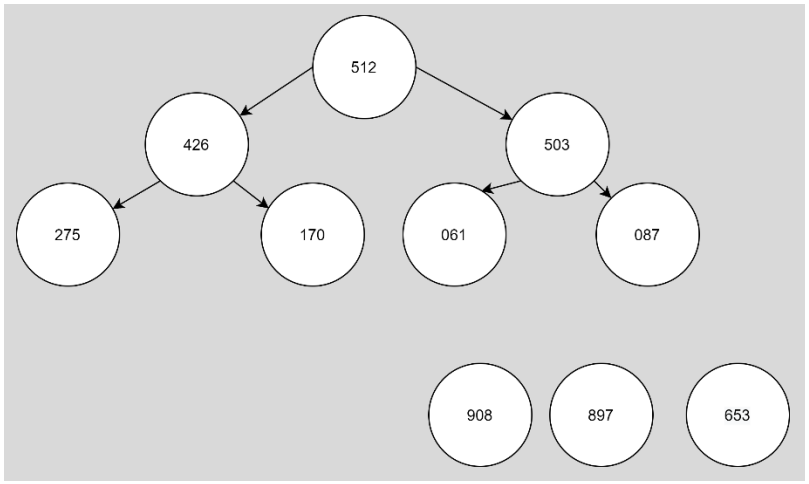
初始大顶堆

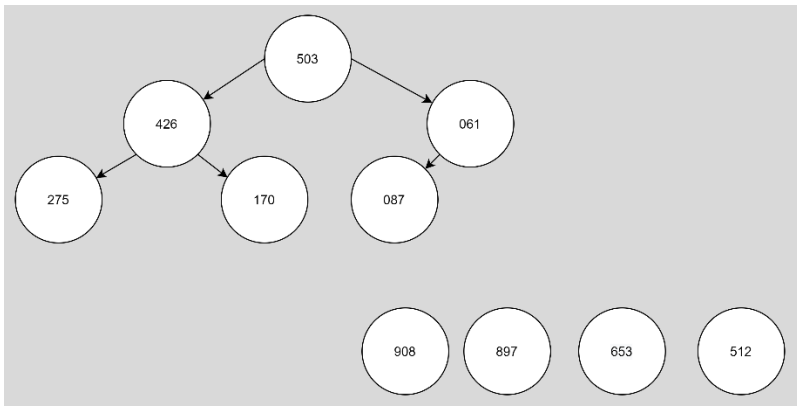


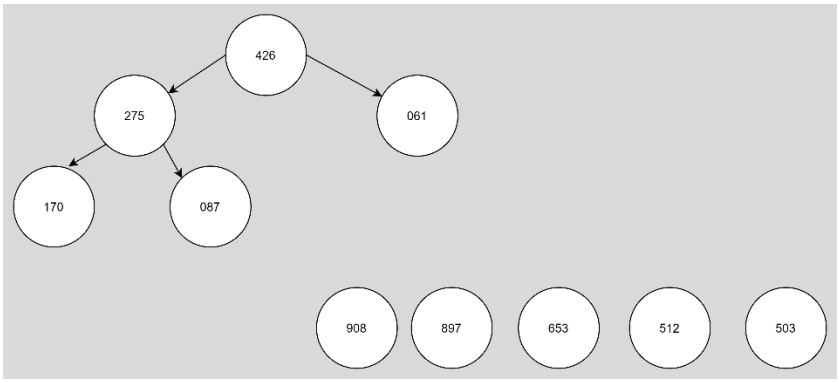
开始进行堆排序

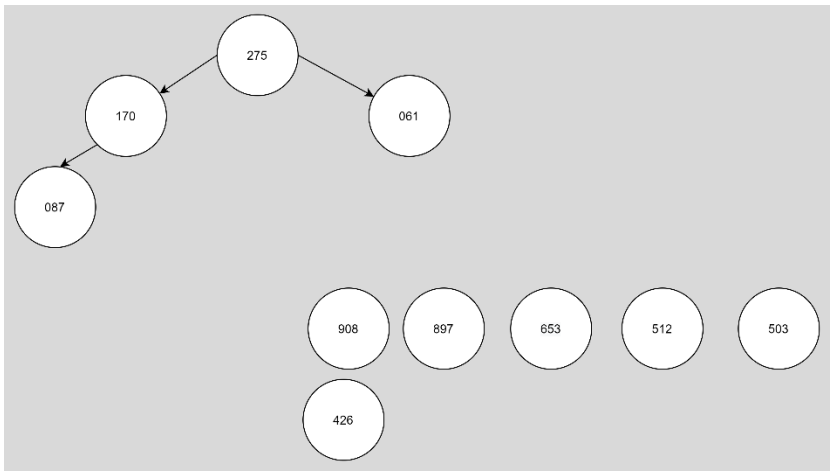


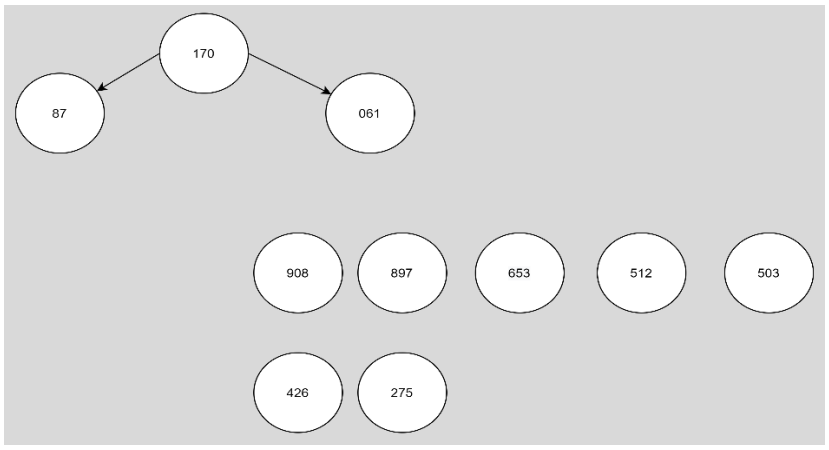

653
426
512
275
170
061
503
087
908
897

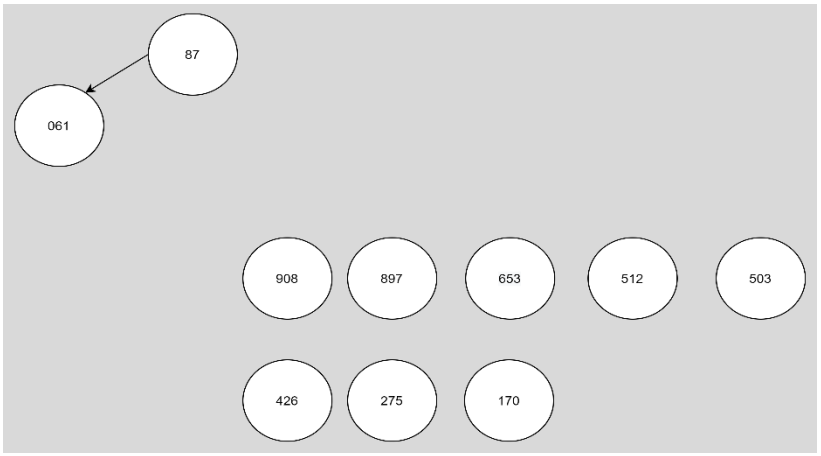

512
426
503
275
170
061
087
908
897
653

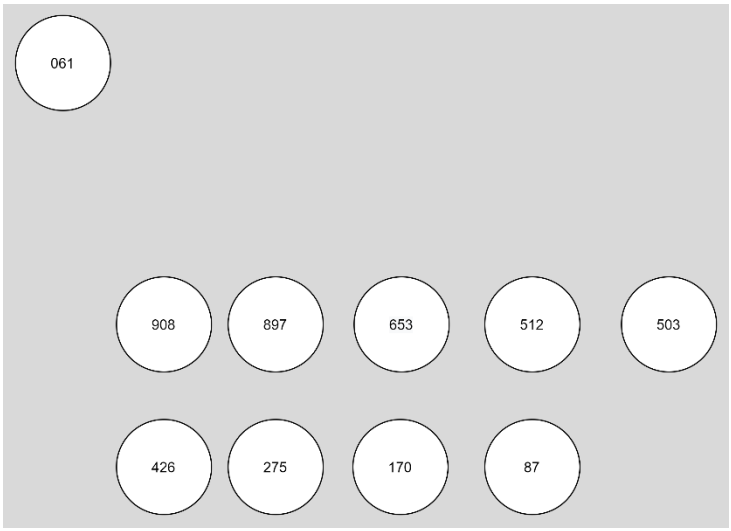

503
426
061
275
170
087
908
897
653
512


426
275
061
170
087
908
897
653
512
503


275
170
061
087
908
897
653
512
503
426


170
87
061
908
897
653
512
503
426
275


87
061
908
897
653
512
503
426
275
170



完成堆排序

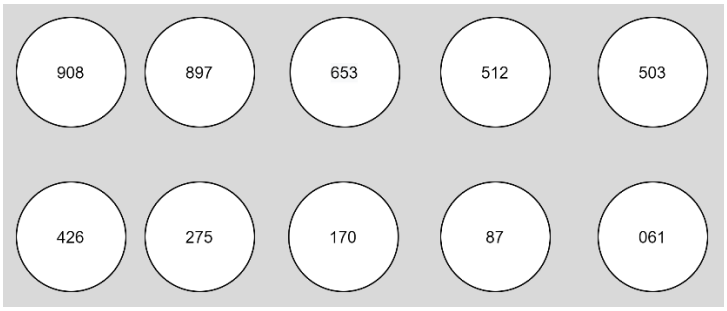


Figure1 堆排序流程图

2、 算法设计：

有 n 个记录存储在带头结点的双向链表中，现用双向冒泡排序法对其按上升序进行排序，请写出这种排序的算法。（注：双向冒泡排序即相邻两趟排序向相反方向冒泡）。

解：

该题的完整代码（可运行程序）在 `Code` 文件夹下（HW15.2.cpp）

双向冒泡的核心思想：
利用双链表的特性，将原本一次循环进行一次上升或者下降的冒泡排序进行拓展，变成为一次循环下降一次（大的沉底）上升一次（小的上浮），因此一次循环最好可以向有序迈前进“两步“并非”一步“

```c
// 双向冒泡排序
void bilateralBubbleSort(struct Node *head)
{
    int swapped; // 标记是否发生交换
    struct Node *start = head;
    struct Node *end = NULL;

    // 检查链表是否为空
    if (head == NULL)
        return;

    do
    {
        swapped = 0;
        struct Node *current = start; // 从头开始冒泡

        // 从左往右冒泡，将较大的节点冒泡到右侧或者最后一个节点
        while (current->next != end)
        {
            if (current->data > current->next->data)// 如果当前节点的数据大于下一个节点的数据则交换
            {
                swapData(current, current->next);
                swapped = 1;
            }
            current = current->next;// 指针后移
        }
        end = current; // 将最后一个节点赋值给end

        if (!swapped)// 如果没有发生交换则说明已经排好序了，则退出循环
            break;

        swapped = 0; // 前面有交换的话swapped为1，这里需要重置swapped
        current = end; // 从最后一个节点开始冒泡

        // 从右往左冒泡，将较小的节点冒泡到左侧
        while (current->prev != start)
        {
            if (current->data < current->prev->data)
            {
                swapData(current, current->prev);
                swapped = 1;
            }
            current = current->prev;
        }
        start = current;// 将第一个节点赋值给start
        // 重复以上步骤，直到没有发生交换
    } while (swapped);
}
```

Figure2 函数`bilateralBubbleSort`代码快照

```c
// 双向冒泡排序
void bilateralBubbleSort(struct Node *head)
{
    int swapped; // 标记是否发生交换
    struct Node *start = head;
    struct Node *end = NULL;

    // 检查链表是否为空
    if (head == NULL)
        return;

    do
    {
        swapped = 0;
        struct Node *current = start; // 从头开始冒泡

        // 从左往右冒泡，将较大的节点冒泡到右侧或者最后一个节点
        while (current->next != end)
        {
            if (current->data > current->next->data)// 如果当前节点的数据大于下一个节点的数据则交换
            {
                swapData(current, current->next);
                swapped = 1;
            }
            current = current->next;// 指针后移
        }
        end = current; // 将最后一个节点赋值给 end

        if (!swapped)// 如果没有发生交换则说明已经排好序了，则退出循环
            break;

        swapped = 0; // 前面有交换的话 swapped 为 1，这里需要重置 swapped
        current = end; // 从最后一个节点开始冒泡
```

```
        // 从右往左冒泡，将较小的节点冒泡到左侧
        while (current->prev != start)
        {
            if (current->data < current->prev->data)
            {
                swapData(current, current->prev);
                swapped = 1;
            }
            current = current->prev;
        }
        start = current;// 将第一个节点赋值给 start
        // 重复以上步骤，直到没有发生交换
    } while (swapped);
}
```

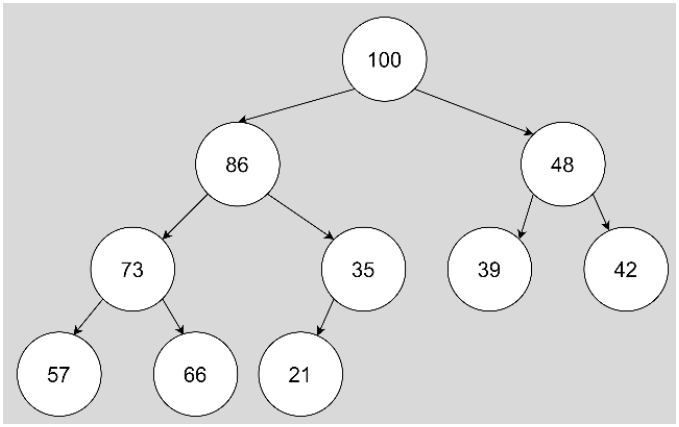
3. 判断以下序列是否是最小堆？如果不是，将它调整为最小堆。

（1）{100, 86, 48, 73, 35, 39, 42, 57, 66, 21}

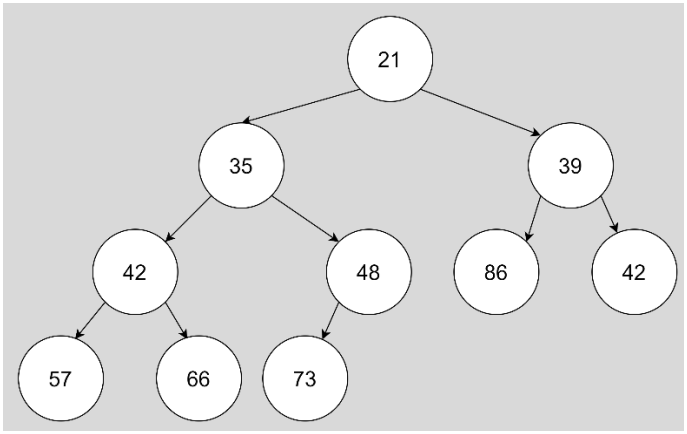
（2）{12, 70, 33, 65, 24, 56, 48, 92, 86, 33}

解：

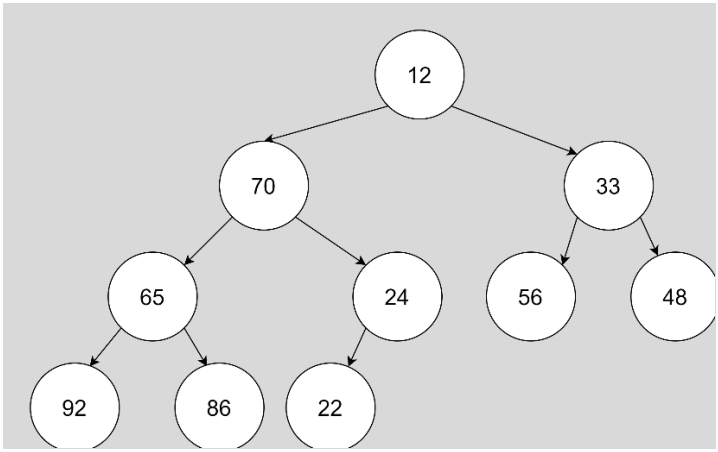
（1）原序列堆图像



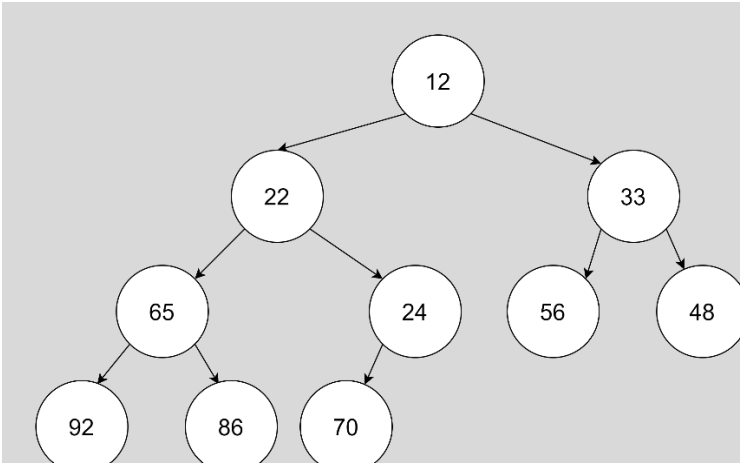
调整为最小堆后



（2） 原序列堆图像



调整为最小堆后



----------END----------