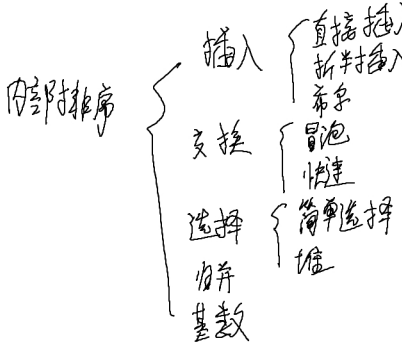
内部排序

稳定排序 => 相同元素 排序后相对位置不变



插入排序

直接插入排序

```
void sort(int arr[], int n){
    for(int i=1; i<n; i++)
        if(arr[i] < arr[i-1]){
            int tmp=arr[i]; int j=i-1;
            for( ; j>=0 && arr[j]>tmp ; j--)
                arr[j+1]=arr[j];
            arr[j+1]=tmp;
        }
}
```

指点	(49)	38	65	97	76	13	27	49
(38)	(38	49)	65	97	76	13	27	49
(65)	(38	49	65)	97	76	13	27	49
(97)	(38	49	65	97)	76	13	27	49
(76)	(38	49	65	76	97)	13	27	49
(13)	(13	38	49	65	76	97)	27	49
(27)	(13	27	38	49	65	76	97)	49
(49)	(13	27	38	49	49	65	76	97)

稳定

空间 $O(1)$

时间
- 最好 $O(n)$
- 最坏 $O(n^2)$
- 平均 $O(n^2)$

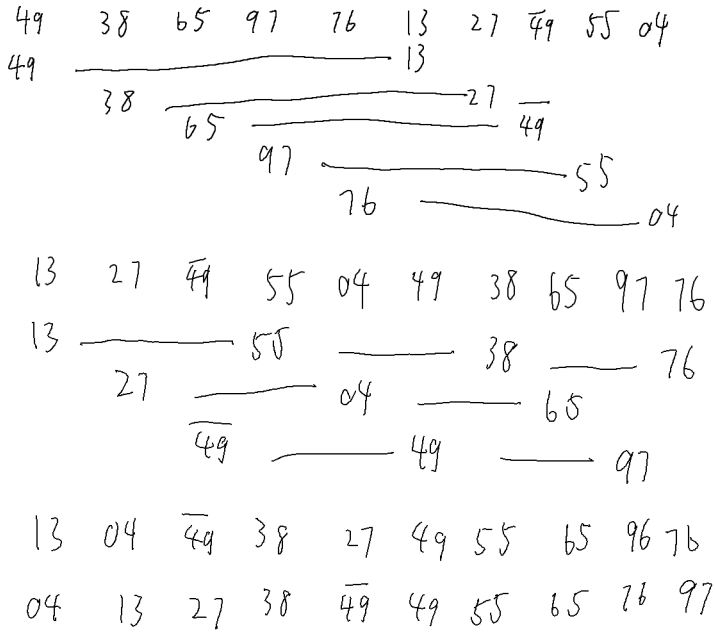
折半插入

```
void sort(int arr[], int n){
    for(int i=1; i<n; i++){
        int j=i-1, l=0, r=i-1, tmp=arr[i];
        while(l<=r){
            int mid=(l+r)>>1;
            if(arr[mid]>tmp) r=mid-1;
            else l=mid+1;
        }
        for(; j>=r+1; j--)
            arr[j+1]=arr[j];
        arr[r+1]=tmp;
    }
}
```

$O(n^2)$ 稳定

希尔排序

```
void shell(int arr[], int n){
    for(step=n/2; step>=1; step/=2){
        for(int i=step; i<n; i++=step)
            if(arr[i]<arr[i-step]){
                int j=i-step, tmp=arr[i];
                for(; j>=0 && arr[j]>tmp; j-=step)
                    arr[j+step]=arr[j];
                arr[j+step]=tmp;
            }
    }
}
```


49 38 65 97 76 13 27 49 55 04
49 13
38 27
65 49
97 55
76 04
13 27 49 55 04 49 38 65 97 76
13 55 38 76
27 04 65
49 49 97
13 04 49 38 27 49 55 65 96 76
04 13 27 38 49 49 55 65 76 97
不穩定 O(n²)

交换排序

冒泡排序

```
void sort (int arr[], int n){
    for(int i=0; i<n-1; i++)
        for(int j=0; j<n-i-1; j++)
            if(arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}
```

稳定　最好 $O(n)$　最坏 $O(n^2)$　平均 $O(n^2)$

	49	38	65	97	76	13	27	49
→	38	49	65	76	13	27	49	97
→	38	49	65	13	27	49	76	97
→	38	49	13	27	49	65	76	97
→	38	13	27	49	49	65	76	97
→	13	27	38	49	49	65	76	97

快速排序

```
void quick(int arr[], int l, int r){
    // l=0, r=n-1
    int i=l, j=r
    int mid=arr[(l+r)>>1];
    do{
      while(arr[l] < mid)      i++;
      while(arr[r] > mid)      j--;
      if(i<=j){
          swap(arr[l], arr[r]);
      }  i++; j--;
    }while(i<=j);
    if( l<j )  quick(arr, l, j);
    if(i<r)    quick(arr, i, r);

}
```

不稳定

最坏 $O(n^2)$

平均 $O(n\log n)$

(49) 38 65 97 76 13 27 49

27 38 65 97 76 13 49

27 38 97 76 13 65 49

27 38 13 97 76 65 49

27 38 13 (49) 76 97 65 49

→ {27} 38 13} (49) {(76) 97 65 49}

→ 13 27 38 49 49 65 76 97

选择排序

简单选择排序

```
void sort(int arr[], int n){
    for(int i=0; i<n-1; i++){
        int min=i;
        for(int j=i+1; j<n; j++)
            if(arr[j]<arr[min])
                min=j;
        swap(arr[i], arr[min]);
    }
}
```

不稳定　最好 最坏都是 $O(n^2)$

堆排序　建堆 $O(n)$ 最好、最坏 $O(n\log n)$ 不稳定

适合类似建字段的情况

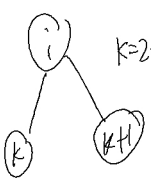
例：在1亿个数数中找出前100个最大值

先建立前100个元素的小根堆，然后读入后的数，拿下的数与堆顶（比这100个数都小）舍去，大则取代堆顶，重新调整

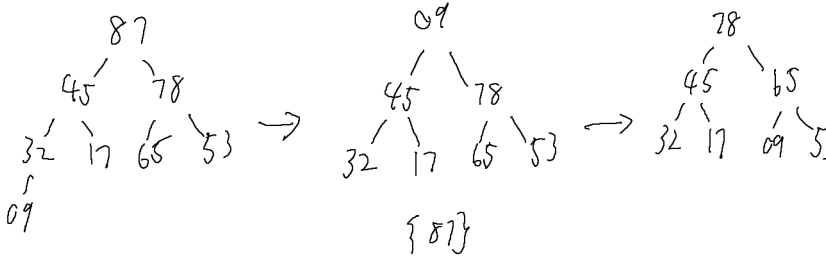
这100个元素的小根堆 就是前100个最大值，堆顶就是这100个中的最小值

排序 结果 升序（从小到大）用大根堆

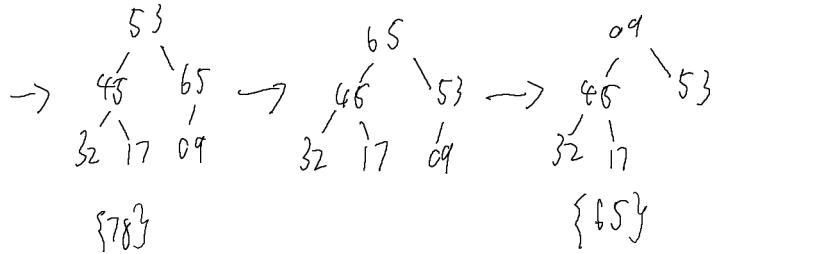
```
void HeapSort(int arr[], int n){
    //建堆
    for(int i=n/2-1; i>=0; i--)
        HeapAdjust(arr, i, n);
    //輸出最大值,重調堆
    for(int j=n-1; j>0; j--){
        swap(arr[j], arr[0]);
        HeapAdjust(arr, 0, j);
    }
}
void HeapAdjust(int arr[], int i, int n){
    int tmp=arr[i];
    for(int k=2*i+1; k<n; k=2*(k+1)){
        if(k+1<n && arr[k]<arr[k+1])
            k++;
        if(arr[k]>tmp){
            arr[i]=arr[k];
            i=k;
        }else{
            break;
        }
    }
    arr[i]=tmp;
}
```



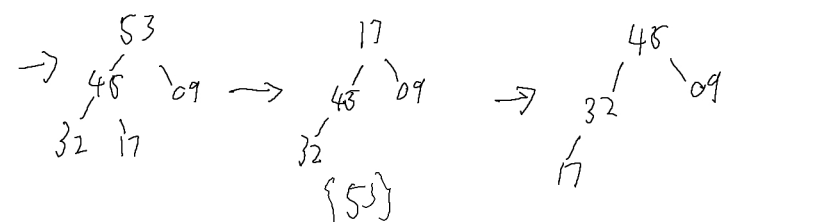
{87, 45, 78, 32, 17, 65, 53, 09}



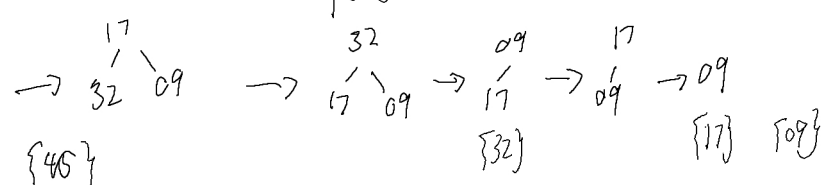
{87}



{78} {65}



{53}



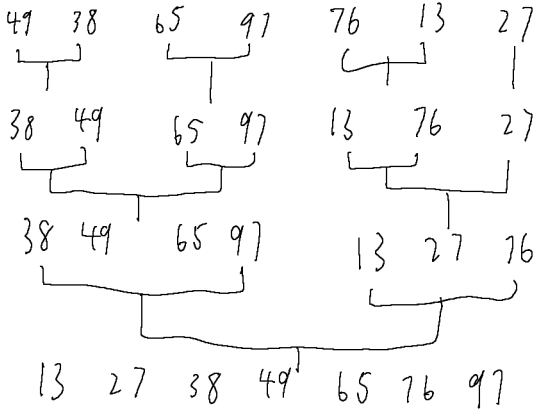
{45} {32} {17} {09}

{09, 17, 32, 45, 53, 65, 78, 87}

归并排序 (2路归并)

```
void MergeSort(int arr[], int l, int r){
    if(l<r){    // [l, mid] [mid+1, r]  l=0, r=n-1
        int mid=(l+r)>>1;
        MergeSort(arr, l, mid);
        MergeSort(arr, mid+1, r);
        Merge(arr, l, mid, r);
    }
}
void Merge(int arr[], int l, int mid, int r){
    int i,j,k; int T[r-l+1];
    for(k=l; k<=r; k++)
        T[k]=arr[k];
    for(i=l, j=mid+1, k=l; i<=mid && j<=r; k++){
        if(T[i]<=T[j])
            arr[k]=T[i++];
        else arr[k]=T[j++];
    }
    while(i<=mid)  arr[k++]=T[i++];
    while(j<=r)    arr[k++]=T[j++];
}
```

空间O(n) 时间O(nlogn) 稳定



基数排序 通常采用链式

{278 109 063 930 589 184 505 269 008 083}

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
930			063 083	184	505			278 008	109 589 269

{930 063 083 184 505 278 008 109 589 269}

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
505 008 109			930			063 269	278	083 184 589	

{505 008 109 930 063 269 278 083 184 589}

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
008 063 083	109 184	269 278			505 589				930

{008 063 083 109 184 269 278 505 589 930}

空间O(r) 时间O(n+rd) 稳定

r是基数
d是位数

内部排序算法的较

算法	时间			空间	稳定
	最好	最坏	平均		
直接插入	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	T
冒泡	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	T
简单选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	F
希尔	数学上难以分析	n在某个范围内$O(n^{1.3})$ 最坏 $O(n^2)$		$O(1)$	F
快速	$O(n\log n)$	$O(n^2)$	$O(n\log n)$	$O(\log n)$	F
堆	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	F
2路归并	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	T
基数	$O(n+rd)$	$O(n+rd)$	$O(n+rd)$	$O(r)$	T

应用

① n较小　直接插入或简单选择
　若记录本身信息大较大　简单选择

② 初始基本有序　冒泡或直接插入

③ n较大　快速、堆、归并
　要求稳定　归并
　关键字随机分布　快速
　辅助空间小　堆

④ n很大但r较小　基数

⑤ 记录信息量较大，为避免移动　采用链表存储