基于比较排序算法，下界nlogn
归并排序、堆排序渐近最优，归排稳定

用快排不用堆排?

- 堆排跳着访问，对CPU缓存不友好
- 堆排无用功多，交换次数大于逆序度
- 时间复杂度只是粗略的估计，一般情况下快排更快

in-place：所需的辅助空间并不依赖于问题的规模n，O（1）extra space
stability：相等的顺序不变，不打乱原有的另一种顺序

基础排序总结

```
插入排序：O（nn）  O（1）   就地，   稳定
选择排序：O（nn）  O（1）   就地，   不稳定
冒泡排序：O（nn）  O（1）   就地，   稳定
归并排序：O（nlgn）O（n）   非就地，  稳定
堆排序：  O（nlgn）O（1）   就地，   不稳定
```

1、选择排序

> A是堆就是堆排序，堆排序也是一种选择排序

递归

> 每次选最大也行

SelectionSortRec(A):

```
if (|A|==1)
  return A
else
  min = GetMinElement(A)
  A’ = RemoveElement(A,min)
  return Concatenate(min,SelectSortionRec(A’))
```

迭代

SelectionSort(A):

```
for (i=1 to A.length-1)
  minIdx = i
  for (j=i+1 to A.length)
    if (A[j]<A[minIdx])
      minIdx=j
  Swap(i,minIdx)
```

- O（nn）O（1）原地 不稳定

2、冒泡排序bubble

BubbleSort(A):

```
for (i=A.length downto 2)
  for (j=1 to i-1)
    if (A[j]>A[j+1])
      Swap(A[j],A[j+1])
```

- Θ(nn) O(1)稳定

改进：检验基本上有序

BubbleSortImproved(A):

```
n=A.length
repeat
  swapped=false
  for (j=1 to n-1)
    if (A[j]>A[j+1])
      Swap(A[j],A[j+1])
      swapped=true
  n=n-1
until (swapped==false)
```

再改进：跳过有序片段

BubbleSortImprovedAgain(A):

```
n=A.length
repeat
  lastSwapIdx=-1
  for (j=1 to n-1)
    if (A[j]>A[j+1])
      Swap(A[j],A[j+1])
      lastSwapIdx=j+1
  n=lastSwapIdx-1
until (n<=1)
```

3、希尔排序Shell（略）

4、快速排序

> 关键和难点在partition

Inplacepartition

> 最后一步把最后一项移到他应该的位置
> 只是换位置，完成就地划分切割

```
Inplacepartition(A,p,r):
i=p,x=A[r]
for(j=p to r)
    if(A[j]<x)
        swap(A[i],A[j])
        i++
swap(A[i],A[r])
return i
```

随机快排

> 取完random交换

- 基础快排：最坏情况很差Θ（nn）,最好Θ（nlgn），需要均匀分割

> 只要划分是常数倍n，就可以达到logn树高

```
RndQuicksort(A,p,r):
if(p<r)
    i=Random(p,r)
    swap(A[i],A[r])
    q=Inplacepartition(A,p,r)
    RndQuicksort(A,p,q-1)
    RndQuicksort(A,q+1,r)
```

改进1：处理重复元素

> 分割时加一段=

改进2：输入小规模不再递归

> 比如r-p>10才递归

改进3：选更多主元，划分更多区域

> 2主元有优势

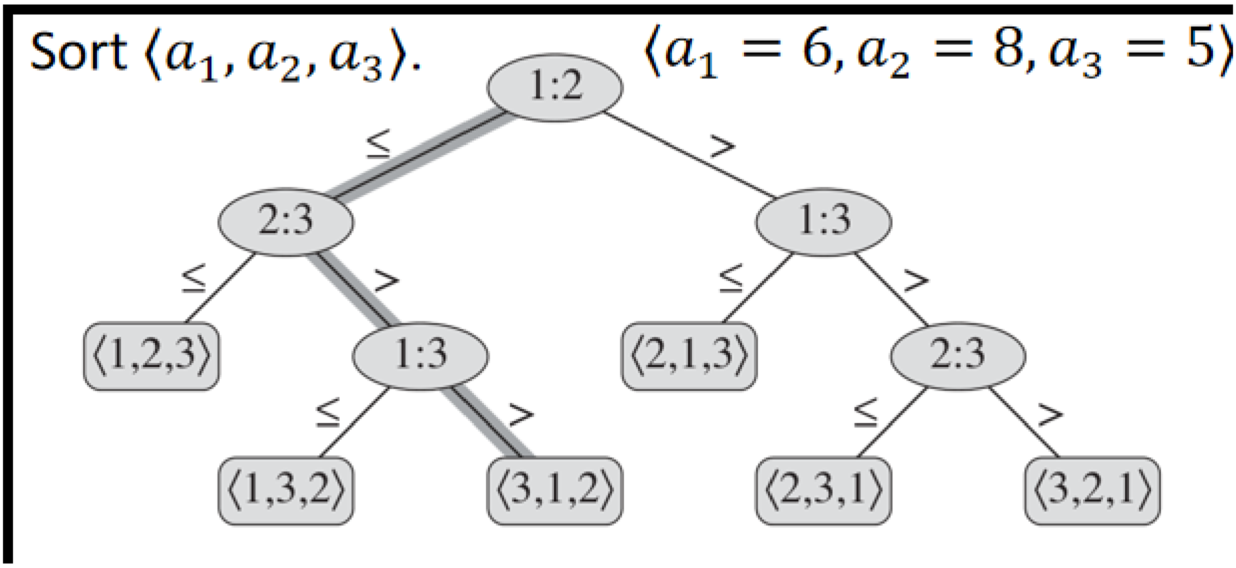
算法评估

对手--lower bound

决策树

> 内部结点代表查询

> 深度代表次数下届
>
> 可以假设都是<=



证明

> 叶>=n!
>
> 深度>=lg（n！），说明Ω(nlogn)