# 内部排序与外部排序

内部排序：待排序记录存放在计算机随机存储器中（说简单点，就是**内存**）进行的排序过程。

外部排序：待排序记录的数量很大，以致于内存不能一次容纳全部记录，所以在排序过程中**需要对外存进行访问**的排序过程。

# https://img-blog.csdn.net/20170506212357095?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvTnVtYmVyT25lT3JOb3RoaW5n/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

# 交换排序

# 冒泡排序

**算法思想：**

从数组头部开始，不断比较相邻的两个元素的大小，让较大的元素逐渐往后移动（交换两个元素的值），直到数组的末尾。经过第一轮的比较，就可以找到最大的元素，并将它移动到最后一个位置。第一轮结束后，继续第二轮。仍然从数组头部开始比较，让较大的元素逐渐往后移动，直到数组的倒数第二个元素为止。经过第二轮的比较，就可以找到次大的元素，并将它放到倒数第二个位置。以此类推，进行 n-1（n 为数组长度）轮“冒泡”后，就可以将所有的元素都排列好。

比较是相邻元素，一趟都会将当前状态下最大或者最小的元素移动到队头/尾

**Example: 2种冒泡的实现方式**

**稳定的**

**void Bubble\_Sort(int k[],int n)**

**{**

**int temp;**

**for (int i=0;i<n-1;i++)**

**{**

**for (int j = i + 1; j < n; j++)**

**{**

**if (k[i] > k[j])**

**{**

**temp = k[j];**

**k[j] = k[i];**

**k[i] = temp;**

**}**

**}**

**}**

**}**

**void Bubble\_Sort(int k[],int n)**

**{**

**int temp;**

**bool flag = false;**

**for (int i = 0; i < n-1; i++)**

**{flag = false;**

**for (int j = 0; j < n - 1 - i; j++)**

**{**

**if (k[j] > k[j + 1])**

**{**

**temp = k[j];**

**k[j] = k[j + 1];**

**k[j + 1] = temp;**

**flag = true;**

**}**

**}**

**if (false == flag)**

**break;**

**}**

**}**

**冒泡排序为什么稳定？**

**由于只有在 不等的时候才交换k[j] > k[j + 1] 所以是稳定的，相同的元素不会换位置**

**复杂度O（n^2）**

# 快速排序

分治法

分：一个问题分解成若干个子问题 （递归）

治：解决这些小问题

越有序，效率越低 越无须 效率越高

最小O(nlog2n) 最大 O(n^2)

空间复杂度 最好 O（log 2n） 最好 O(n)斜树

不稳定

5 2 2

#include "stdafx.h"

int Partation(int k[],int low,int high)

{

//用于分割数组 返回分割线

int partation = k[low];

//先将第一个元素设置为分割点

while (low < high)

{

//从high开始向左找

while (low<high&&k[high] >= partation)high--; //low<high条件为了防止越界

k[low] = k[high];

while (low<high&&k[low]<=partation)low++; //low<high为了防止越界

k[high] = k[low];

}

k[low] = partation;

return low;

}

void QuickSort(int k[],int low,int high)

{

//用于快速排序

if (low < high) {

int partation = Partation(k, low, high);

QuickSort(k, low, partation- 1);

QuickSort(k, partation + 1, high);

//递归完成！

//直到low>=high为止

}

}

int main()

{

int k[] = { 1,4,6,3,5,8,0,4,2,24,7,9,31,4,56,90,100 };

QuickSort(k, 0, sizeof(k) / sizeof(int) - 1);

return 0;

}

快速排序为什么不稳定？

看一组数 比如说 5 2 2

那么将5设置为portation i=5 j=2（最后一个） 2<5 所以 2 和 5 换

有 2（第三个） 2 5 那么同样是2 2（第三个）2 和 2 2（第三个）都是一样，所以不稳定

快速排序为什么需要O(log2n)的辅助空间

因为快速排序需要用到递归

QuickSort(k, low, partation- 1);

QuickSort(k, partation + 1, high);

需要用栈的空间

**选择排序**

# **选择排序 不稳定 （2 3 5 2 9 8 8）**

**void SelectSort(int k[], int n)**

**{**

**int temp, min;**

**int count = 0;**

**int exchange = 0;**

**for (int i = 0; i < n - 1; i++)**

**{**

**min = i;**

**for (int j = i + 1; j < n; j++)**

**{**

**count++;**

**if (k[min] > k[j])**

**{**

**min = j;**

**}**

**}**

**if (min != i)**

**{**

**exchange++;**

**temp = k[min];**

**k[min] = k[i];**

**k[i] = temp;**

**}**

**}**

**printf("compare %d exchange %d\n", count, exchange);**

**}**

**复杂度O（n^2）**

**选择排序为什么不稳定？**

# **看一组数组 k[ ] = （2 3 5 2 9 8 8）**

第一轮，设2（第零个）为min，那么找到第三个2的时候 与第一个交换了位置，所以不稳定

# 堆排序 堆是一个完全二叉树

任何一个非叶节点都大于左右孩子

大顶堆/小顶堆

初始化的复杂度 O（n）

排序的复杂度O（nlog2n）

大顶堆：节点比孩子大

小顶堆：结点比孩子小

#include "stdafx.h"

void print(int k[], int n)

{

for (int i = 1; i <= n; i++)

printf("%d ", k[i]);

printf("\n");

}

void Swap(int k[],int i,int j)

{

int temp = 0;

temp = k[i];

k[i] = k[j];

k[j] = temp;

print(k, 20);

}

void AdjustHeap(int k[],int index,int n)

{

int temp = k[index];//记录根节点

for (int i = index\*2; i <=n; i\*=2)

{

if (i<n&&k[i] < k[i + 1]) //可能这棵树的n/2取下限没有右孩子

{

i++;

}

if (temp > k[i])

break;

k[index] = k[i]；

k[i] = temp;

index = i;

}

}

void HeapSort(int k[],int n)//这里n就是个数 k[0]不用

{

//调整大顶堆

//从下自上，从右到左调整

//时间复杂度O(N)

for (int i = n / 2; i >= 1; i--)

{

AdjustHeap(k, i, n);

}

print(k, 20);

//调整之后，开始交换再调堆

//时间复杂度O（log2N）

//总时间复杂度 O（N）+O（log2N）

for(int j=0;j<n;j++)

{

Swap(k, 1, n - j);

AdjustHeap(k, 1, n - j - 1);

}

}

int main()

{

int k[] = { -1,9,8,7,8,0,8,7,9,5,4,8,89,75,100,87,67,46,2,1,0 };

HeapSort(k, sizeof(k) / sizeof(int) - 1);

return 0;

}

**堆排序为什么不稳定**

1. **看数组k[]=**3 27 36 27

如果堆顶3先输出，则，第三层的27（最后一个27）跑到堆顶，然后堆稳定，继续输出堆顶，是刚才那个27，这样说明后面的27先于第二个位置的27输出，不稳定。

1. **看代码**

**if (i<n&&k[i] < k[i + 1])**

**{**

**i++;**

**}**

**if (temp > k[i])**

**break;**

**k[index] = k[i]；**

**k[i] = temp;**

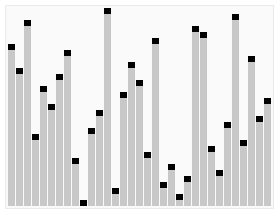
**index = i;**

**只要是非temp>k[i]都执行交换，也就是存在等于**

**比如 3 25 27（1） 27（2）**

**27（2）25 27（1）3**

**这里面就交换了位置**



# 插入排序O(n)-O（n^2）

**void InsertSort(int \*k,int n)**

**{**

**int temp,j;**

**for (int i = 0; i < n-1; i++)**

**{**

**if (k[i] > k[i + 1])**

**{**

**temp = k[i + 1];**

**for (j = i; k[j] > temp; j--)**

**{**

**k[j + 1] = k[j];**

**}**

**k[j+1] = temp;**

**}**

**}**

**}**

# **希尔排序O(n^1.3-2) 不稳定**

基本有序 （插入排序效率高）

记录比较少

两个条件过多苛刻

记录分组，每个子序列进行基本的插入排序



**void ShellSort(int \*k, int n)**

**{**

**//设置分组 组局为gap**

**int gap = n;//最开始的组距为数组k的长度**

**int j=0;**

**int temp = 0; //用作交换元素的中间变量**

**do {**

**gap = gap / 3 + 1;**

**//使用公式： n/3+1来逐渐减小gap，直到gap==1为止**

**for (int i = gap; i < n; i++)**

**{**

**if (k[i] < k[i - gap])**

**{**

**temp = k[i];**

**for ( j= i-gap; k[j] > temp; j -= gap)**

**{**

**//向前探测索引，直到比temp小的一个元素出现为止**

**k[j+gap] = k[j];**

**//向后移动元素**

**}**

**//不成功元素的索引值在插入位置前一个单位，所以要先向后移动一个单位(gap)**

**k[j + gap] = temp;**

**}**

**}**

**} while (gap > 1)**

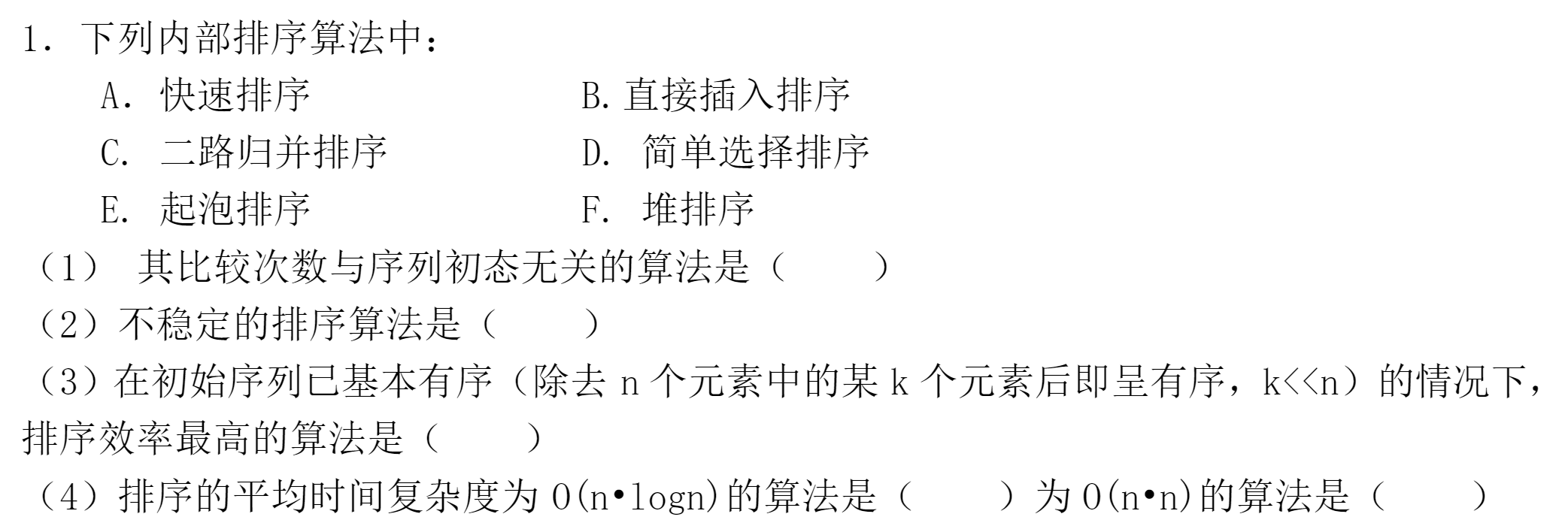
**}**

希尔排序为什么不稳定？

由于多次[插入排序](https://www.baidu.com/s?wd=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "_blank)，我们知道一次[插入排序](https://www.baidu.com/s?wd=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)是稳定的，不会改变相同元素的相对顺序，但在不同的[插入排序](https://www.baidu.com/s?wd=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)过程中，相同的元素可能在各自的插入排序[中移动](https://www.baidu.com/s?wd=%E4%B8%AD%E7%A7%BB%E5%8A%A8&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，最后其稳定性就会被打乱，所以[shell排序](https://www.baidu.com/s?wd=shell%E6%8E%92%E5%BA%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)是不稳定的。



# 有关习题

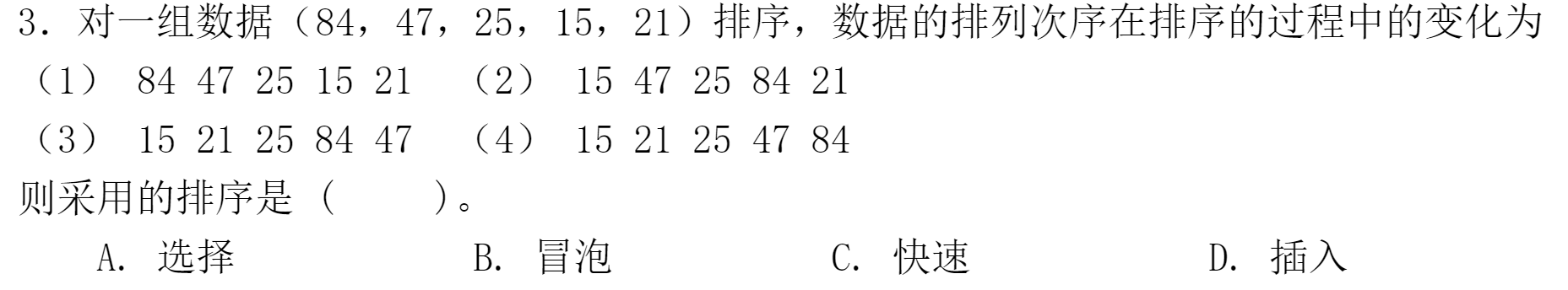


* 1. 快速排序的复杂度范围是O（nlog2n）--- O（n^2）数据越无序，效率越高
  2. 插入排序里面，如果数据本来就有序，那么就没有往前比较的这个过程，也就是说，数据整体上越有序，比较次数就越少
  3. 二路归并排序，不论数组有序与否，比较次数一样
  4. 简单选择排序，选择排序减少的只不过是相对于冒泡排序的交换的次数，但是比较的次数还是那么多
  5. 冒泡排序如果整体上有序，那么交换的次数会少
  6. 堆排序如果一开始数据就是有序的，那么调整堆的时候就会交换次数变多

所以答案CD

1. A,D,F
2. B 这里面冒泡排序比较次数不变，插入排序次数更少
   1. A快速排序平均复杂度O(nlog2n)
   2. B插入基本复杂度O(n^2)
   3. 二路归并排序复杂度平均O（nlogn）
   4. 简单选择排序复杂度平均O(n^2)
   5. 起泡排序平均复杂度O(n^2)
   6. 堆排序平均复杂度O(nlog2n)

所以第一问 ACF 第二问 BDE



发现，这里面从第一位开始，每一次排序都有一位排列整齐，而且都是选择剩下序列中最小的那一个，所以是选择排序。

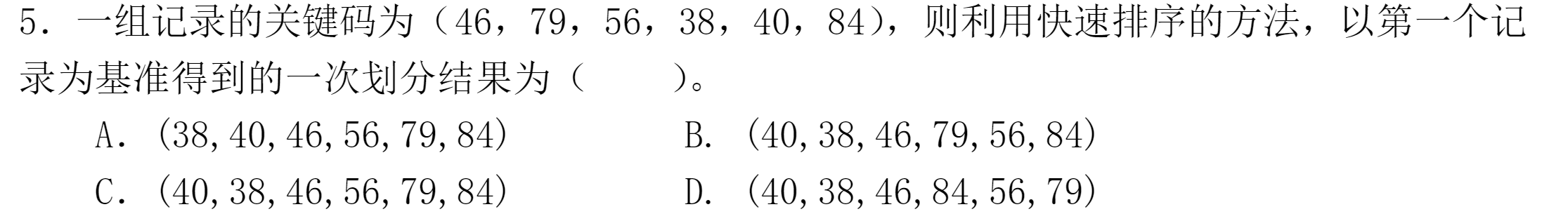


选择一趟结束之后最小的元素放第一个

冒泡一趟结束后最大的元素放最后一个

堆排序一趟结束之后，如果是小顶堆，最小的放最后，如果是大顶堆一趟结束后，最大的放最后

归并排序不是，比较出来下一次递归还要重新排列,所以选C



第一个为基准partation 那么 i=46 j=84

J往前找，找到40比46小，于是将40给46

40 79 56 38 40 84

I往后找，发现79大于46，所以将79赋给j

40 79 56 38 79 84

J往前找 发现38小于46 所以与i换

40 38 56 38 79 84

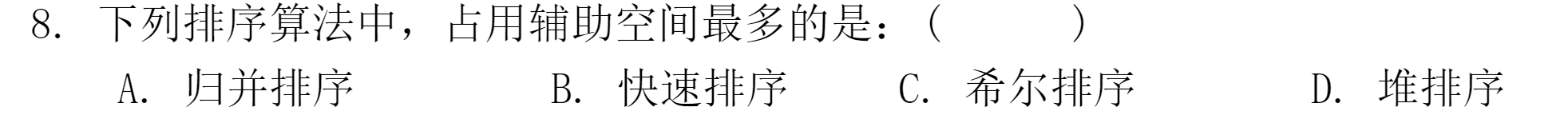
I往后找，找到56比46大，换给j

40 38 56 56 79 84

J往前找，与i重合，将partation换过来 得到第一趟结果：

40 38 46 56 79 84

所以答案选C

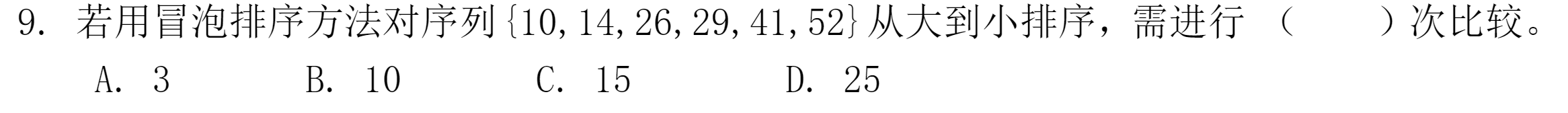


堆排序<快速排序<归并排序

归并排序需要一个temp[MAX\_LENGTH]的数组来辅助排序，并且需要栈的辅助

快速排序由于递归，所以需要栈的空间

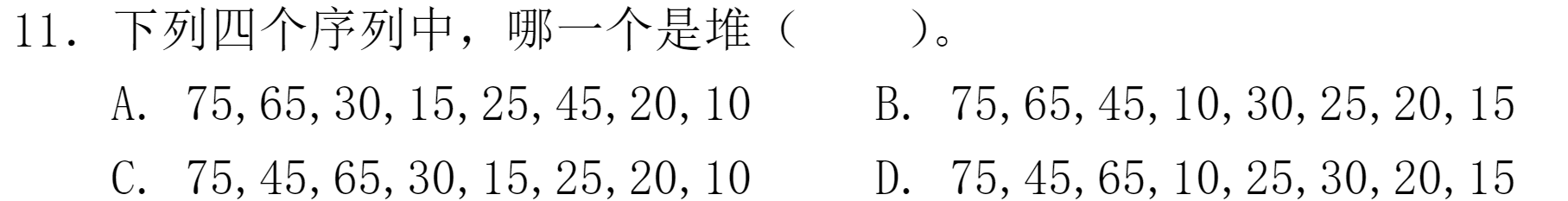
堆排序空间复杂度为O（1）不需要空间



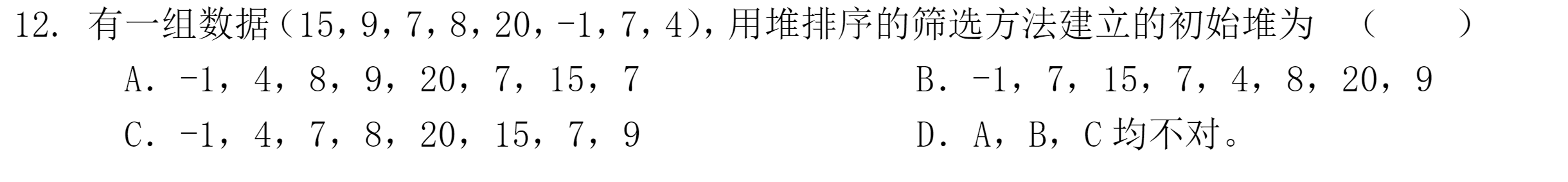
10 14 26 29 41 52

由于数列是从小到大顺序排列的，所以改成从大到小需要比较5+4+3+2+1=15次

（1+5）\*5/2=15次 选C



堆有大顶堆和小顶堆 所以明显C是大顶堆



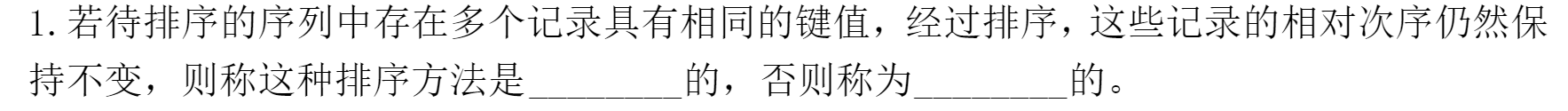
A的最后一个7不符合小顶堆

B的4不对，不符合小顶堆

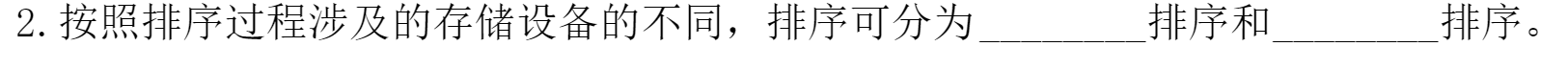
C符合小顶堆

故答案选C

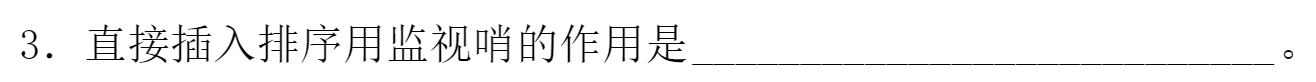
稳定的概念：



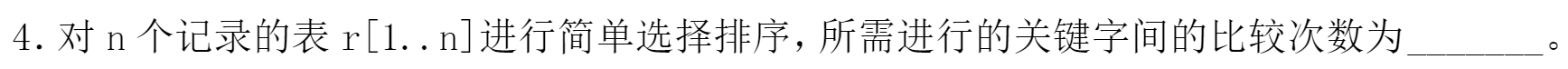
排序之后，相同的值顺序不变称为稳定，否则不稳定



内部排序&外部排序



减少了循环中语句的个数，提高了执行效率



第一轮 n-1次

第二轮 n-2次

第三轮 n-3次

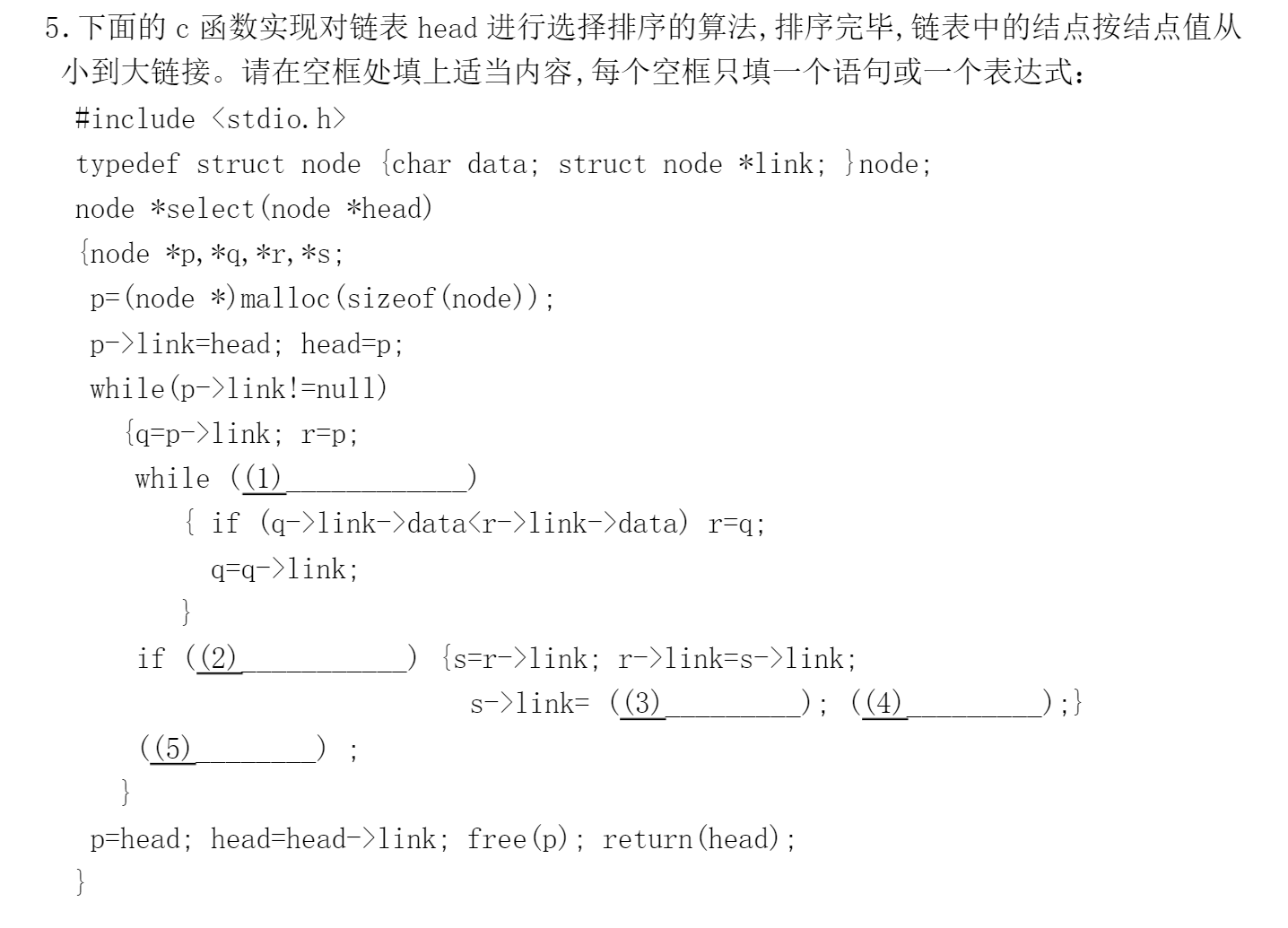
………

倒数第一轮 2次

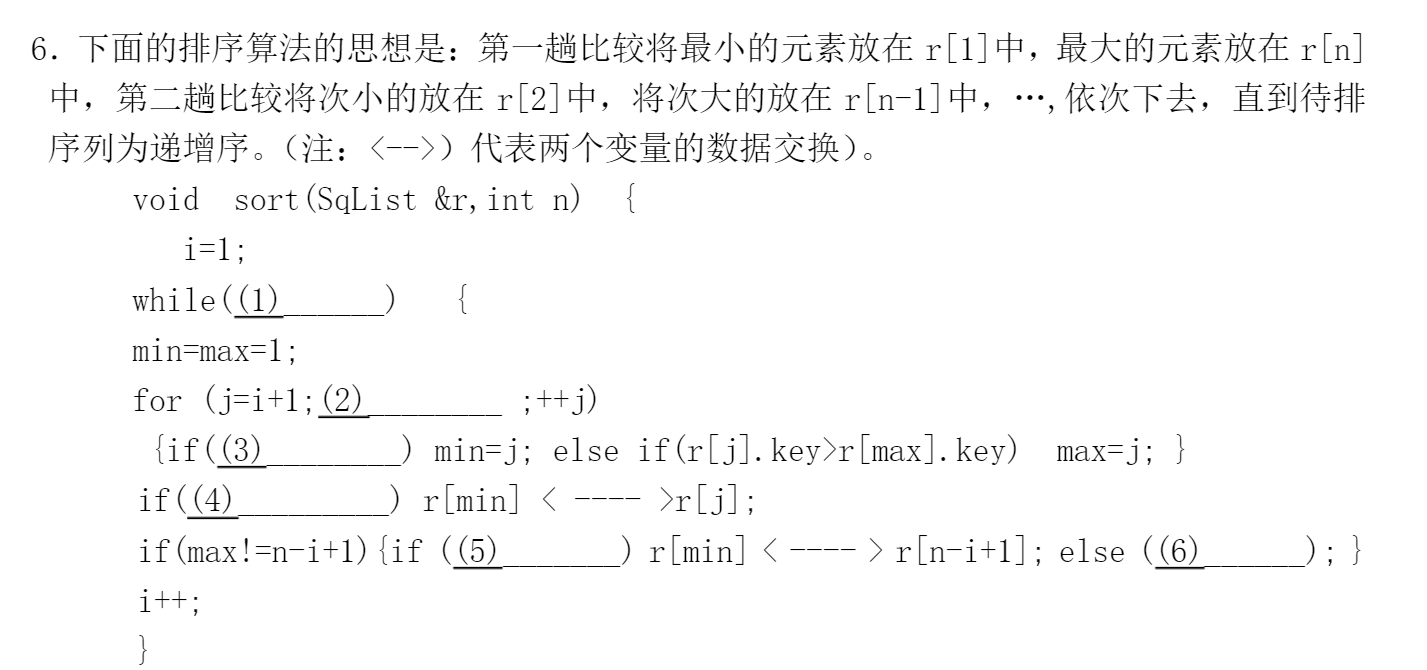
最后一轮 1次

所以是 1+2+3+4+……+n-1

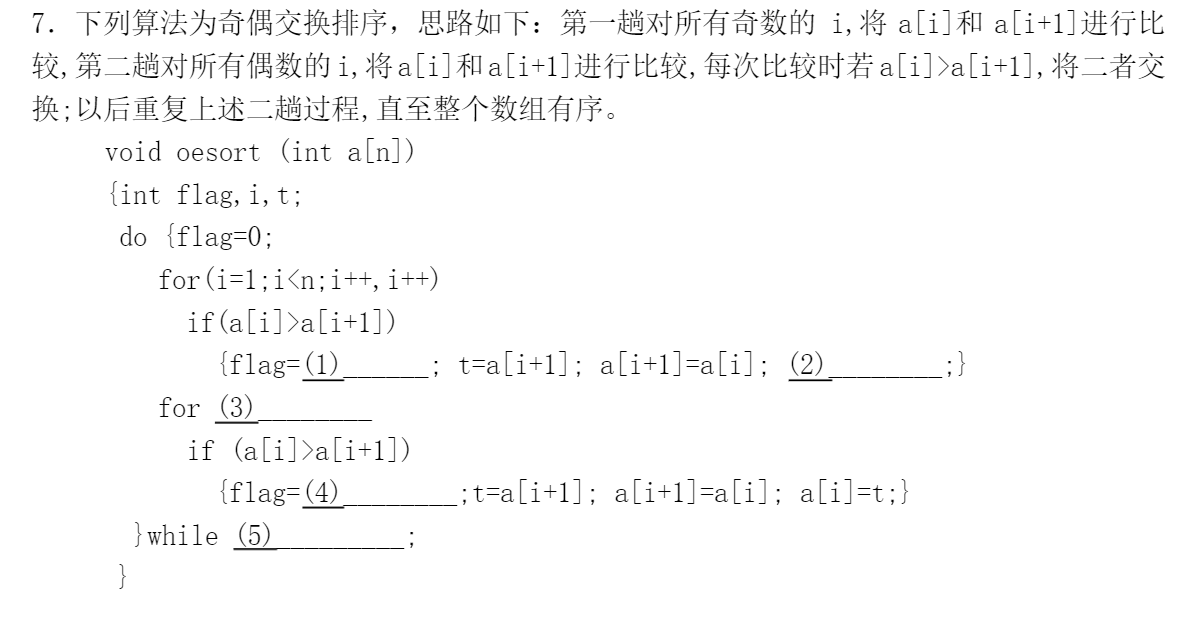
等差数列求和 次数为：n(n-1)/2

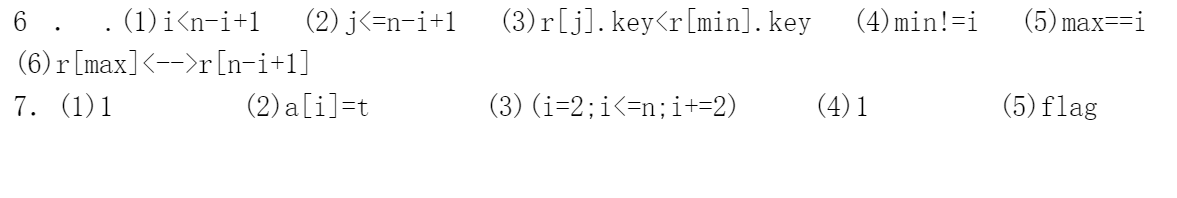


1. P->link!=NULL
2. r!=p; r==p说明r没有变化 也就是r就是最小的那个
3. P->link;
4. P->link=s;
5. p=p->link;



}//sort





**1. 大多数排序算法都有两个基本的操作： 查找数据 和 移动数据 。**

**2. 在对一组记录（54，38，96，23，15，72，60，45，83）进行直接插入排序时，当把第7个记录60插入到有序表时，为寻找插入位置至少需比较 3 次。**

**54**

**38 54**

**38 54 96**

**23 38 54 96**

**15 23 38 54 96**

**15 23 38 54 72 96**

**15 23 38 54 60 72 96**

**和 96 72 54 比较3次**

**3. 在插入和选择排序中，若初始数据基本正序，则选用 插入 ；若初始数据基本反序，则选用 选择 。**

**4. 在堆排序和快速排序中，若初始记录接近正序或反序，则选用 堆排序 ；若初始记录基本无序，则最好选用 快速排序 。**

**5. 对于n个记录的集合进行冒泡排序，在最坏的情况下所需要的时间是 O（n^2） 。若对其进行快速排序，在最坏的情况下所需要的时间是 O(n^2) 。**

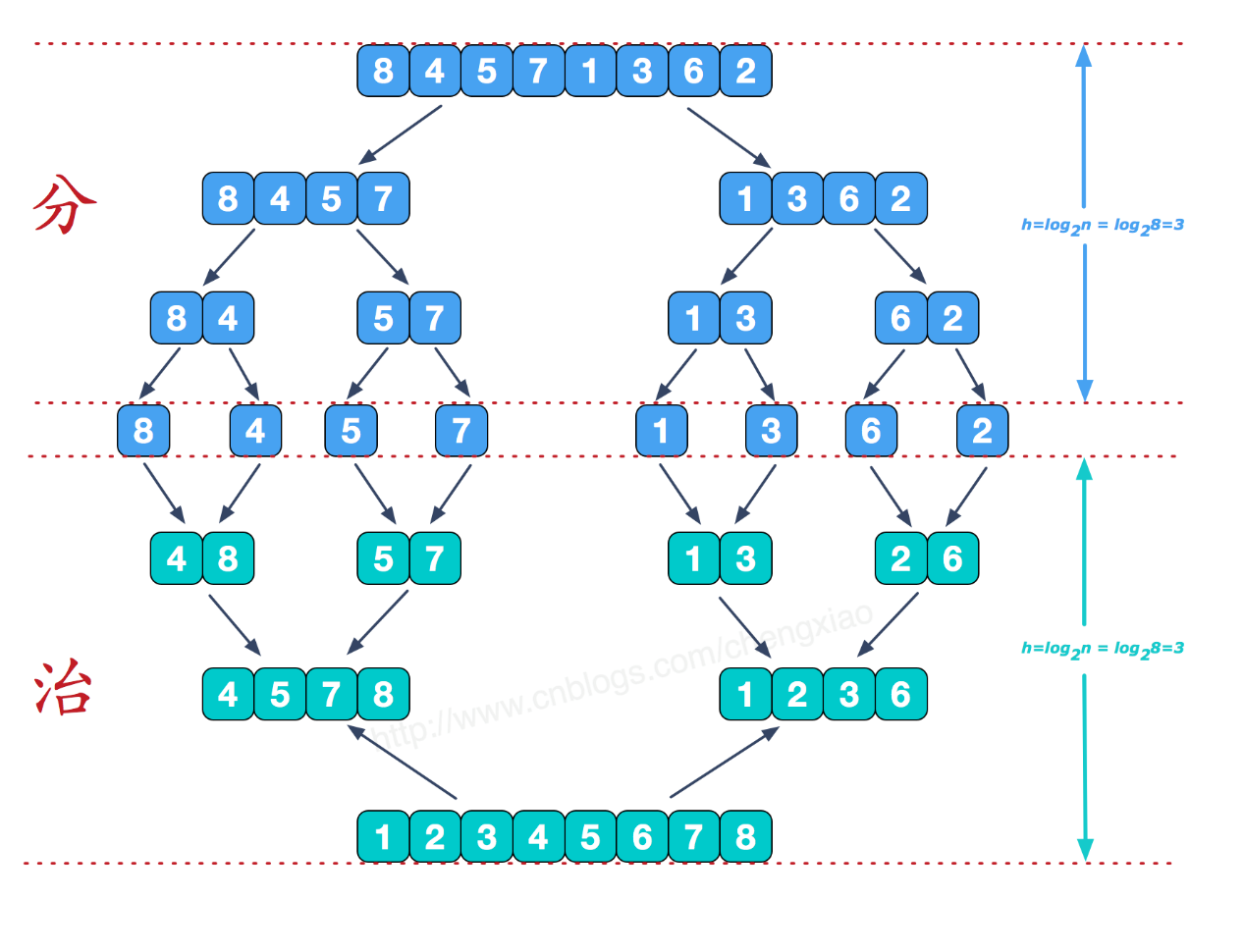
**6. 对于n个记录的集合进行归并排序，所需要的平均时间是 O(nlog2n) ，所需要的附加空间是 O(n) 。**

**7． 对于n个记录的表进行2路归并排序，整个归并排序需进行 log2n向上取整 趟（遍）。**

**可以看出归并排序是一棵完全二叉树，所以高度为log2n向上取整**

**这里面是n个记录，如果按照结点算的话，完全二叉树的高度是log2n向下取整+1**

**那么这里面不看最后一层（最后一层没有比较趟数）有 log27向下取整+1=2+1=3**



**8. 设要将序列（Q, H, C, Y, P, A, M, S, R, D, F, X）中的关键码按字母序的升序重新排列，则：**

**冒泡排序一趟扫描的结果是 （H,C,Q,P,A,M,S,R,D,F,X,Y） ；**

**初始步长为4的希尔（shell）排序一趟的结果是 (P,A,C,S,Q,D,F,X,R,H,M,Y) ；**

**二路归并排序一趟扫描的结果是 ；**

**快速排序一趟扫描的结果是 (F,H,C,DP,A,M,Q,R,S,Y,X) ；**

**堆排序初始建堆的结果是 (A,D,C,R,F,Q,M,S,Y,P,H,X) 。**

**9. 在堆排序、快速排序和归并排序中，**

**若只从存储空间考虑，则应首先选取 堆排序O(1) 方法，其次选取 快速排序O（log2n)方法，最后选取**

**归并排序O(n) 方法；**

**若只从排序结果的稳定性考虑，则应 选取 归并排序 方法；**

**若只从平均情况下最快考虑，则应选取 快速排序 方法；**

**若只从最坏情况下最快并且要节省内存考虑，则应选取 堆排序 方法。 堆排序 最好最坏平均都是O(log2n) 归并排序同 但是归并空间复杂度O(n) 堆排序O（1）**

**二、单项选择题（每小题1分，共18分）**

**（ C ）1．将5个不同的数据进行排序，至多需要比较 次。**

**Ａ. 8 Ｂ. 9 Ｃ. 10 Ｄ. 25**

**考虑以下几种排序**

1. **冒泡排序 最坏情况是 n(n-1)/2 也就是逆序的时候 带入5得10**
2. **选择排序 与数组无关 一直都是n(n-1)/2=10**
3. **插入排序 如果全是逆序就比较 n(n-1)/2=10**
4. **归并排序 都是 nlog2n >10**
5. **快速排序 退化为n(n-1)/2**
6. **堆排序 nlog2n >10**
7. **堆排序为什么是nlog2n的比较次数？**
   1. **因为对于堆排序，无论大顶堆还是小顶堆 高度都是log2n，如果换上去的每个元素都从上到下走一遍，那么就是最麻烦的次数 一共n个元素，所以一共nlog2n次**

**（ C ）2． 排序方法中，从未排序序列中依次取出元素与已排序序列（初始时为空）中的元素进行比较（选择不比较），将其放入已排序序列的正确位置上的方法，称为**

**Ａ. 希尔排序 Ｂ. 冒泡排序 Ｃ. 插入排序 Ｄ. 选择排序**

**（ D ）3．从未排序序列中挑选元素，并将其依次插入已排序序列（初始时为空）的一端的方法，称为**

**Ａ. 希尔排序 Ｂ. 归并排序 Ｃ. 插入排序 Ｄ. 选择排序**

**（ B ）4．对ｎ个不同的排序码进行冒泡排序（自小到大），在下列哪种情况下比较的次数最多。**

**Ａ. 从小到大排列好的 Ｂ. 从大到小排列好的 Ｃ. 元素无序 Ｄ. 元素基本有序**

**（ D ）5．对ｎ个不同的排序码进行冒泡排序，在元素无序的情况下比较的次数为**

**Ａ. n+1 Ｂ. n Ｃ. n-1 Ｄ. n(n-1)/2**

**（ C ）6．快速排序在下列哪种情况下最易发挥其长处。**

**Ａ. 被排序的数据中含有多个相同排序码 Ｂ. 被排序的数据已基本有序**

**Ｃ. 被排序的数据完全无序 Ｄ. 被排序的数据中的最大值和最小值相差悬殊**

**（B ）7． 对有n个记录的表作快速排序，在最坏情况下，算法的时间复杂度是**

**A．O(n) B．O(n2) C．O(nlog2n) D．O(n3)**

**（ C ）8．若一组记录的排序码为（46, 79, 56, 38, 40, 84），则利用快速排序的方法，以第一个记录为基准得到的一次划分结果为**

**Ａ. 38, 40, 46, 56, 79, 84 Ｂ. 40, 38, 46 , 79, 56, 84**

**Ｃ. 40, 38，46, 56, 79, 84 Ｄ. 40, 38, 46, 84, 56, 79**

**（ D ）9．下列关键字序列中， 是堆。**

**Ａ. 16, 72, 31, 23, 94, 53 Ｂ. 94, 23, 31, 72, 16, 53**

**Ｃ. 16, 53, 23, 94，31, 72 Ｄ. 16, 23, 53, 31, 94, 72**

**（ B ）10．堆是一种 排序。**

**Ａ. 插入 Ｂ.选择 Ｃ. 交换 Ｄ. 归并**

**（ C ）11．堆的形状是一棵**

**Ａ. 二叉排序树 Ｂ.满二叉树 Ｃ. 完全二叉树 Ｄ. 平衡二叉树**

**（ B ）12．若一组记录的排序码为（46, 79, 56, 38, 40, 84），则利用堆排序的方法建立的初始堆为**

**Ａ. 79, 46, 56, 38, 40, 84 Ｂ. 84, 79, 56, 38, 40, 46**

**Ｃ. 84, 79, 56, 46, 40, 38 Ｄ. 84, 56, 79, 40, 46, 38**

**（ C ）13． 下述几种排序方法中，要求内存最大的是**

**Ａ. 插入排序 Ｂ.快速排序 Ｃ. 归并排序 Ｄ. 选择排序**