学号：ZF2221315

姓名：陆坤

《程序设计与算法》大作业队伍：静姐的背包

按照小组任务分配，此次大作业我承担的主要工作是实现普通的排序算法。

1、程序的设计思路

1、定义一个随意数组。

```
### 通过rand()函数生成随机数进而定义随机数组。c语言生成的随机数为无符号数，即都是正的，要想生成正负随机的，可以通过模除限定返回，再减去对应范围的中间值即可。
```

```
int size,i;
int * arr;
arr = (int *) malloc(30000 * sizeof(int ));//定义动态数组
printf("请输入随机数组的长度:\n");
scanf("%d",&size);
printf("生成的随机数组:\n");
for ( i = 0; i < size; i++) {
arr[i]=rand()%30001 – 15000；
printf("%d ",arr[i]);
}
```

2）输出时每20个元素为一组。

```
if(i%20==19)
printf("\n");
```

2、定义排序函数

选择排序

1）工作原理

第一次从待排序的中数据元素选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素的个数为零。

2）代码实现

```
void selection_sort(int* p, int len)
{
int i = 0;
int j = 0;
for (i = 0; i < len – 1; i++)
{
int index = i;
for (j = i + 1; j < len; j++)
{
if (p[index] > p[j])
index = j;
}
if (index != i)
{
int tmp = p[index];
p[index] = p[i];
p[i] = tmp;
}
}
}
```

归并排序

1）工作原理

基本算法

该算法是采用分治法的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列。即先使每个子序列有序，再使子序列段间有序

分离

将已有数列不断分离成两段长度基本相同（当已有数列长度是奇数时，则一半长一半短），直到分离成长度为 1 的 n 个数列（其实就是 n 个数。

合并

将数列两两合并，每次合并时进行比较和排序，直到完成排序。

2）代码实现

```
void merge_sort(int a[],int left,int right){
if(left<right){
int mid = (left + right) / 2;
merge_sort(a,left, mid);
merge_sort(a, mid + 1, right);
merge(a, left, right, mid);
}
}
void merge(int a[],int left,int right,int mid) {
int s[100];
int i = left, j = mid + 1;
int sor = left;
while (i <= mid && j <= right) {
if (a[i] < a[j]) {
s[sor++] = a[i++];
}
else {
s[sor++] = a[j++];
}
}
while (i <= mid) s[sor++] = a[i++];
while (j <= right) s[sor++] = a[j++];
sor = left;
while (sor <= right) {
a[sor] = s[sor];
sor++;
}
}
```

快速排序

1）基本原理

快速排序是对冒泡排序的一种改进。它的基本思想是：通过一趟排序将待排序记录分割成独立的两部分，其中一部分记录的关键字均比另一部分记录的关键字小，则可以分别对着两部分记录继续进行排序，以达到整个序列有序。

2）代码实现

```
void quick_sort(int num[], int low, int high )
{
int i,j,temp;
int tmp;

i = low;
j = high;
tmp = num[low]; //任命为中间分界线，左边比他小，右边比他大,通常第一个元素是基准数

if(i > j) //如果下标i大于下标j，函数结束运行
{
return;
}
```

```
while(i != j)
{
    while(num[j] >= tmp && j > i)
    {
        j--;
```

```
}
while(num[i] <= tmp && j > i)
{
i++;
}
if(j > i)
{
temp = num[j];
num[j] = num[i];
num[i] = temp;
}
}
num[low] = num[i];
num[i] = tmp;
quick_sort(num,low,i–1);
quick_sort(num,i+1,high);
}
```

希尔排序

1）基本原理

希尔排序的是插入排序的提升(建议先去了解一下插入排序)。它是通过将数据根据每一次的步长不断的将数据进行分组，并且进行处理，使得数值序列整体不会变得太过杂乱。使得在利用插入排序的过程中减少交换的次数，从而使整体得到优化。

2）代码实现

```
void ShellSort(int* arr, int sz)
{
int count=0;//这是我为了看看希尔排序和直接插入排序的性能比较而设置的计数
int gap = sz;//设置排序的间隔
while ( gap > 1 )
{
//这里一定要保证gap最后进来循环后为1，所以对此加1
gap = gap / 3 + 1;//gap>1为与排序，gap==1，为直接插入排序

for (int i = 0; i < sz – gap; i++)//这里并不是一次性把一组排完，而是挨个往后，一组一个轮流排
{
int end = i;
int tmp = arr[end + gap];
while (end >= 0)
{
if (tmp < arr[end])
{
arr[end + gap] = arr[end];
end –= gap;
}
else
{
break;
}
}
arr[end + gap] = tmp;
}
}
}
```

基数排序

1）基本原理

先将待排序序列按个位数排好，然后按顺序复制回原数组；再按十位排序好，再按顺序复制回原数组；依次类推，按百位、千位，排序的趟数就是最大数的位数，比如千位数就要排四趟，百位数就要排3趟。

2）代码实现

```
void RadixSort(int* arr, int n)
{
int max = arr[0];
int min = arr[0];
int base = 1;
for (int i = 0; i < n; i++)
{
if (arr[i] > max)
{
max = arr[i];
}
if (arr[i] < min)
{
min = arr[i];
}
}

for (int i = 0; i < n; i++)
{
arr[i] += abs(min);
}
int* tmp = (int*)malloc(sizeof(int)*n);
while (max / base > 0)
{

int bucket[10] = { 0 };
for (int i = 0; i < n; i++)
{
bucket[arr[i] / base % 10]++;
}
for (int i = 1; i < 10; i++)
{
bucket[i] += bucket[i – 1];
}

for (int i = n – 1; i >= 0; i––)
{
tmp[bucket[arr[i] / base % 10] – 1] = arr[i];
bucket[arr[i] / base % 10]––;
}

for (int i = 0; i < n; i++)
{
arr[i] = tmp[i];
}
base *= 10;
}
free(tmp);
for (int i = 0; i < n; i++)
{
arr[i] –= abs(min);
}
}
```

3、计算函数运行时间

```
clock_t start, finish;
double Total_time;
start = clock();
输入排序函数名称;
finish = clock();
Total_time = (double)(finish – start) / CLOCKS_PER_SEC; //单位换算成秒
```

4、输出排序后的函数