数据结构

Processon：https://www.processon.com/view/link/5caf6129e4b0773d8c0be48b

数据结构 1
冒泡（基于交换） 1
快排（基于交换） 2
简单选择 4
归并 5
桶排序 8

冒泡（基于交换）

```
1.  #include <iostream>
2.  using namespace std;
3.  template<typename T>
4.  //整数或浮点数皆可使用
5.  void bubble_sort(T arr[], int len)
6.  {
7.      int i, j;  T temp;
8.      for (i = 0; i < len - 1; i++)
9.          for (j = 0; j < len - 1 - i; j++)
10.         if (arr[j] > arr[j + 1])
11.         {
12.             temp = arr[j];
13.             arr[j] = arr[j + 1];
14.             arr[j + 1] = temp;
15.         }
16. }
17. int main()
18. {
19.     int arr[] = { 61, 17, 29, 22, 34, 60, 72, 21, 50, 1, 62 };
20.     int len = (int) sizeof(arr) / sizeof(*arr);
21.     bubble_sort(arr, len);
22.     for (int i = 0; i < len; i++)
23.         cout << arr[i] << ' ';
24.
25.     cout << endl;
26.
27.     float arrf[] = { 17.5, 19.1, 0.6, 1.9, 10.5, 12.4, 3.8, 19.7, 1.5, 25.4, 28.6, 4.4, 23.8, 5.4 };
28.     len = (int) sizeof(arrf) / sizeof(*arrf);
29.     bubble_sort(arrf, len);
```

```
    for (int i = 0; i < len; i++)
        cout << arrf[i] << ' ';

    return 0;
}
```

快排（基于交换）

```
#include <iostream>
 //快排
using namespace std;

void Qsort(int arr[], int low, int high){
    if (high <= low) return;
    int i = low;
    int j = high + 1;
    int key = arr[low];
    while (true)
    {
        /*从左向右找比 key 大的值*/
        while (arr[++i] < key)
        {
            if (i == high){
                break;
            }
        }
        /*从右向左找比 key 小的值*/
        while (arr[--j] > key)
        {
            if (j == low){
                break;
            }
        }
        if (i >= j) break;
        /*交换 i,j 对应的值*/
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    /*中枢值与 j 对应值交换*/
    int temp = arr[low];
    arr[low] = arr[j];
    arr[j] = temp;
    Qsort(arr, low, j - 1);
```

```
    Qsort(arr, j + 1, high);
}

int main()
{
    int a[] = {57, 68, 59, 52, 72, 28, 96, 33, 24};

    Qsort(a, 0, sizeof(a) / sizeof(a[0]) - 1);/*这里原文第三个参数要减 1 否则内存越界*/

    for(int i = 0; i < sizeof(a) / sizeof(a[0]); i++)
    {
        cout << a[i] << "";
    }

    return 0;
}/*参考数据结构 p274(清华大学出版社，严蔚敏)*/
```

简单选择

```
#include<iostream>
#include<time.h>
#include<iomanip>
using namespace std;
const int N=10;
int main()
{
   int a[N],i,j,temp,b;
   srand(time(NULL));
   for(i=0;i<N;i++)
      a[i]=rand()%100;
   for(i=0;i<N;i++)
      cout<<setw(3)<<a[i];
   cout<<endl;
   for(i=0;i<N-1;i++)
   {
      temp=i;
      for(j=i+1;j<N;j++)
      {
         if(a[temp]>a[j])
            temp=j;
      }
      if(i!=temp)
      {
         b=a[temp];
         a[temp]=a[i];
         a[i]=b;
      }
   }
   for(i=0;i<N;i++)
      cout<<setw(3)<<a[i];
   cout<<endl;
}
```

归并

//非递归

```
#include<iostream>
#include<ctime>
#include<cstring>
#include<cstdlib>
using namespace std;
/**将 a 开头的长为 length 的数组和 b 开头长为 right 的数组合并 n 为数组长度，用于最后一组
*/
void Merge(int* data,int a,int b,int length,int n){
  int right;
  if(b+length-1 >= n-1)  right = n-b;
  else  right = length;
  int* temp = new  int[length+right];
  int i=0, j=0;
  while(i<=length-1 && j<=right-1){
    if(data[a+i] <= data[b+j]){
      temp[i+j] = data[a+i];i++;
   }
    else{
    temp[i+j] = data[b+j];
    j++;
   }
  }
  if(j == right){//a 中还有元素，且全都比 b 中的大,a[i]还未使用
   memcpy(temp + i + j, data + a + i, (length - i) * sizeof(int));
  }
 else if(i == length){
   memcpy(temp + i + j, data + b + j, (right - j)*sizeof(int));
 }
  memcpy(data+a, temp, (right + length) * sizeof(int));
  delete [] temp;
}
void MergeSort(int* data, int n){
  int step = 1;
  while(step < n){
    for(int i=0; i<=n-step-1; i+=2*step)
      Merge(data, i, i+step, step, n);
  //将 i 和 i+step 这两个有序序列进行合并
  //序列长度为 step
  //当 i 以后的长度小于或者等于 step 时，退出
    step*=2;//在按某一步长归并序列之后，步长加倍
```

```
    }
}
int    main(){
    int    n;
    cin>>n;
    int*    data = new    int[n];
    if(!data)    exit(1);
    int    k = n;
    while(k--){
        cin>>data[n-k-1];
    }
    clock_t    s = clock();
    MergeSort(data, n);
    clock_t    e = clock();
    k=n;
    while(k--){
        cout<<data[n-k-1]<<' ';
    }
    cout<<endl;
    cout<<"the algorithm used"<<e-s<<"miliseconds."<<endl;
    delete    data;
    return    0;
}
```

//递归

```
#include<iostream>
using namespace std;
void merge(int *data, int start, int mid, int end, int *result)
{
    int i, j, k;
    i = start;
    j = mid + 1;                //避免重复比较 data[mid]
    k = 0;
    while (i <= mid && j <= end)      //数组 data[start,mid]与数组(mid,end]均没有全部归入数组 result 中去
    {
        if (data[i] <= data[j])       //如果 data[i]小于等于 data[j]
            result[k++] = data[i++];   //则将 data[i]的值赋给 result[k]，之后 i,k 各加一，表示后移一位
        else
            result[k++] = data[j++];   //否则，将 data[j]的值赋给 result[k]，j,k 各加一
    }
    while (i <= mid)              //表示数组 data(mid,end]已经全部归入 result 数组中去了，而数组 data[start,mid]还有剩余
        result[k++] = data[i++];      //将数组 data[start,mid]剩下的值，逐一归入数组 result
```

```
    while (j <= end)                //表示数组 data[start,mid]已经全部归入到 result 数组中去了，而数组(mid,high]还有剩余
        result[k++] = data[j++];      //将数组 a[mid,high]剩下的值，逐一归入数组 result

    for (i = 0; i < k; i++)         //将归并后的数组的值逐一赋给数组 data[start,end]
        data[start + i] = result[i];   //注意，应从 data[start+i]开始赋值
}
void merge_sort(int *data, int start, int end, int *result)
{
    if (start < end)
    {
        int mid = start + (end-start) / 2;//避免溢出 int
        merge_sort(data, start, mid, result);                //对左边进行排序
        merge_sort(data, mid + 1, end, result);              //对右边进行排序
        merge(data, start, mid, end, result);                //把排序好的数据合并
    }
}
void amalgamation(int *data1, int *data2, int *result)
{
    for (int i = 0; i < 10; i++)
        result[i] = data1[i];
    for (int i = 0; i < 10; i++)
        result[i + 10] = data2[i];
}
int main()
{
    int data1[10] = { 1,7,6,4,9,14,19,100,55,10 };
    int data2[10] = { 2,6,8,99,45,63,102,556,10,41 };
    int *result = new int[20];
    int *result1 = new int[20];
    amalgamation(data1, data2, result);
    for (int i = 0; i < 20; ++i)
        cout << result[i] << "  ";
    cout << endl;
    merge_sort(result, 0, 19, result1);
    for (int i = 0; i < 20; ++i)
        cout << result[i] << "  ";
    delete[]result;
    delete[]result1;
    return 0;
}
```

桶排序

桶排序 (Bucket sort)或所谓的**箱排序**，是一个排序算法，工作的原理是将数组分到有限数量的桶子里。每个桶子再个别排序（有可能再使用别的排序算法或是以递归方式继续使用桶排序进行排序）。桶排序是鸽巢排序的一种归纳结果。当要被排序的数组内的数值是均匀分配的时候，桶排序使用线性时间（Θ（*n*））。但桶排序并不是 比较排序，他不受到 O(n log n) 下限的影响。

中文名	桶排序	数据结构设计	链表可以采用很多种方式实现
要　求	数据的长度必须完全一样	性　质	平均情况下桶排序以线性时间运行
公　式	Data=rand()/10000+10000	原　理	桶排序利用函数的映射关系
		领　域	计算机算法

海量数据

一年的全国高考考生人数为500 万，分数使用标准分，最低100 ，最高900 ，没有小数，要求对这500 万元素的数组进行排序。

分析：对500W数据排序，如果基于比较的先进排序，平均比较次数为O(5000000*log5000000)≈1.112亿。但是我们发现，这些数据都有特殊的条件： 100=<score<=900。那么我们就可以考虑桶排序这样一个“投机取巧”的办法、让其在毫秒级别就完成500万排序。

方法：创建801(900-100)个桶。将每个考生的分数丢进f(score)=score-100的桶中。这个过程从头到尾遍历一遍数据只需要500W次。然后根据桶号大小依次将桶中数值输出，即可以得到一个有序的序列。而且可以很容易的得到100分有***人，501分有***人。

实际上，桶排序对数据的条件有特殊要求，如果上面的分数不是从100-900，而是从0-2亿，那么分配2亿个桶显然是不可能的。所以桶排序有其局限性，适合元素值集合并不大的情况。

典型

在一个文件中有10G个整数，乱序排列，要求找出中位数。内存限制为2G。只写出思路即可（内存限制为2G意思是可以使用2G空间来运行程序，而不考虑本机上其他软件内存占用情况。） 关于中位数：数据排序后，位置在最中间的数值。即将数据分成两部分，一部分大于该数值，一部分小于该数值。中位数的位置：当样本数为奇数时，中位数=(N+1)/2；当样本数为偶数时，中位数为N/2与1+N/2的均值（那么10G个数的中位数，就第5G大的数与第5G+1大的数的均值了）。

分析：既然要找中位数，很简单就是排序的想法。那么基于字节的桶排序是一个可行的方法。