# 数据结构项目十文档

**同济大学 软件学院 15级2班 1552651 王依睿**

# 使用说明

## 操作手册

### 开始

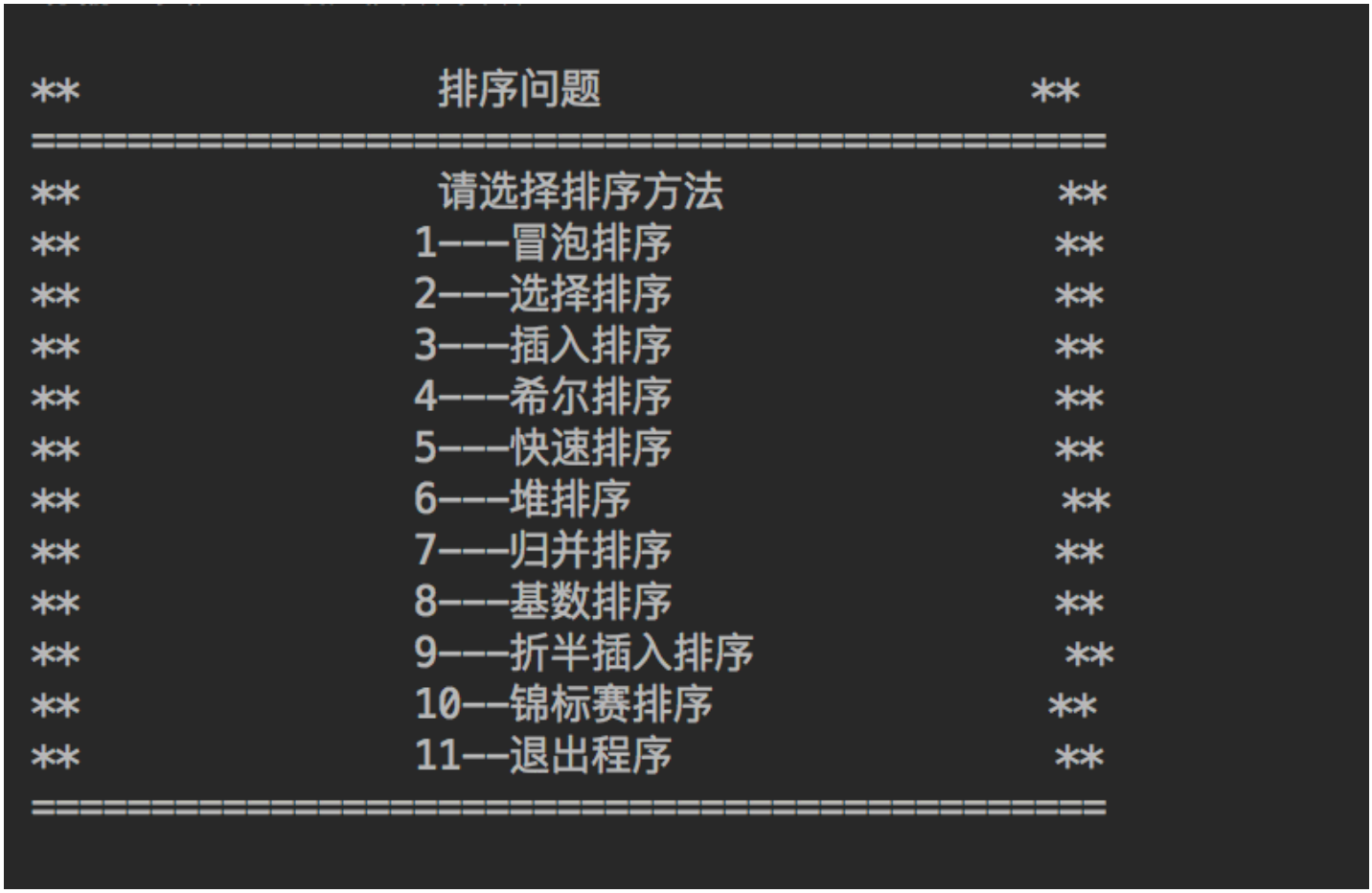
运行程序后，请求用户输入要产生随机数的个数。

```
请输入要产生的随机数个数：
```

用户输入要产生随机数的个数。

```
请输入要产生的随机数个数：10000
```

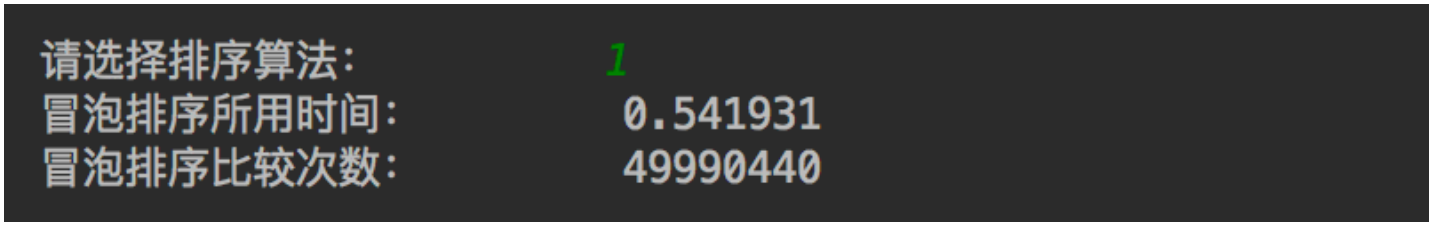
输出的操作说明。




请求用户输入想执行的排序算法。


**冒泡排序**

- 选择操作1。
- 输出冒泡排序所用的时间。
- 输出冒泡排序比较次数。




**选择排序**

- 选择操作2。
- 输出选择排序所用的时间。
- 输出选择排序比较次数。

```
请选择排序算法:          2
选择排序所用时间:        0.223692
选择排序比较次数:        49995000
```

**插入排序**

- 选择操作3。
- 输出插入排序所用的时间。
- 输出插入排序比较次数。

```
请选择排序算法:          3
插入排序所用时间:        0.170939
插入排序比较次数:        24873993
```

**希尔排序**

- 选择操作4。
- 输出希尔排序所用的时间。
- 输出希尔排序比较次数。

```
请选择排序算法:          4
希尔排序所用时间:        0.004320
希尔排序比较次数:        251085
```

**快速排序**

- 选择操作5。
- 输出快速排序所用的时间。
- 输出快速排序比较次数。

```
请选择排序算法:          5
快速排序所用时间:         0.007307
快速排序比较次数:         281556
```

**堆排序**

- 选择操作6。
- 输出堆排序所用的时间。
- 输出堆排序比较次数。

```
请选择排序算法:          6
堆排序所用时间:          0.006903
堆排序比较次数:          235479
```

**归并排序**

- 选择操作7。
- 输出归并排序所用的时间。
- 输出归并排序比较次数。

```
请选择排序算法:          7
归并排序所用时间:         0.005067
归并排序比较次数:         123676
```

**基数排序**

- 选择操作8。
- 输出基数排序所用的时间。
- 输出基数排序比较次数。

```
请选择排序算法:          8
基数排序所用时间:         0.005357
基数排序比较次数:         0
```

### 折半插入排序

- 选择操作9。
- 输出折半插入排序所用的时间。
- 输出折半插入排序比较次数。

```
请选择排序算法:          9
折半插入排序所用时间:      0.108180
折半插入排序比较次数:      118966
```

### 锦标赛排序

- 选择操作10。
- 输出锦标赛排序所用的时间。
- 输出锦标赛排序比较次数。

```
请选择排序算法:          10
锦标赛排序所用时间:       0.008730
锦标赛排序比较次数:       141765
```

### 退出

- 选择操作11。

```
请选择排序算法:          11

Process finished with exit code 0
```

## 注意事项

- 排序所用时间与运行此程序的计算机性能相关，故在不同的计算机上运行从程序时，相同的随机数和排序算法可能会出现时间不同的情况。
- 当随机数个数较大或计算机性能相对较弱时排序时间会较长，请耐心等待。

# 概述

## 项目功能要求

随机函数产生10000个随机数，用快速排序，直接插入排序，冒泡排序(又叫起泡排序)，直接选择排序的排序方法排序，并统计每种排序所花费的排序时间和交换次数。其中，随机数的个数由用户定义，系统产生随机数，并且显示他们的比较次数，排序算法包括冒泡排序，直接选择排序，直接插入排序，希尔排序，快速排序，堆排序，归并排序和基数排序(又叫基排序)。 (提示：还可加入折半插入排序和锦标赛排序，一共10种排序方式)

## 程序设计目的

用不同的排序算法实现从大到小对数组进行排序，比较不同排序算法的时间复杂度和空间复杂度。

## 算法思路

### 直接插入排序

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中从后向前扫描
3. 如果该元素（已排序）大于新元素，将该元素移到下一位置
4. 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置
5. 将新元素插入到该位置后
6. 重复步骤2~5

### 折半插入排序

在将一个新元素插入已排好序的数组的过程中，寻找插入点时，将待插入区域的首元素设置为a[low]，末元素设置为a[high]，则轮比较时将待插入元素与a[m]，其中m=(low+high)/2相比较,如果比参考元素大，则选择a[low]到a[m-1]为新的插入区域(即high=m-1)，否则选择a[m+1]到a[high]为新的插入区域（即low=m+1），如此直至low<=high不成立，即将此位置之后所有元素后移一位，并将新元素插入a[high+1]。

### 希尔排序

希尔排序通过将比较的全部元素分为几个区域来提升插入排序的性能。

### 直接选择排序

首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。

### 堆排序

是指利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

### 锦标赛排序

首先对n个记录进行两两比较，然后优胜者之间再进行两两比较，如此重复，直至选出最小关键字的记录为止。这个过程可以用一棵有n个叶子结点的完全二叉树表示。根节点中的关键字即为叶子结点中的最小关键字。在输出最小关键字之后，根据关系的可传递性，欲选出次小关键字，仅需将叶子结点中的最小关键字改为“最大值”，如∞，然后从该叶子结点开始，和其左（右）兄弟的关键字进行比较，修改从叶子结点到根的路径上各结点的关键字，则根结点的关键字即为次小关键字。

### 冒泡排序

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。
3. 针对所有的元素重复以上的步骤，除了最后一个。
4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

### 快速排序

1. 从数列中挑出一个元素，称为"基准"（pivot），
2. 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区结束之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。
3. 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

### 归并排序

1. 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列
2. 设定两个指针，最初位置分别为两个已经排序序列的起始位置
3. 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置
4. 重复步骤3直到某一指针到达序列尾
5. 将另一序列剩下的所有元素直接复制到合并序列尾

### 基数排序

将所有待比较数值（正整数）统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后，数列就变成一个有序序列。

## 数据结构、稳定性、时间、空间复杂度

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **排** | | **数** | **稳** | | | | | |

| 序算法 | 类别 | 据结构 | 定性 | 比较次数 | （最坏）时间复杂度 | 最优时间复杂度 | 平均时间复杂度 | 空间复杂度 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 直接插入排序 | 插入排序 | 数组 | 稳定 | $n^2/2$ | $O(n^2)$ | $O(n)$ | $O(n^2)$ | 总共$O(n)$，需要辅助空间$O(1)$ |
| 折半插入排序 | 插入排序 | 数组 | 稳定 | $n\log^2 n$ | $O(n^2)$ | $O(n)$ | $O(n^2)$ | 总共$O(n)$，需要辅助空间$O(1)$ |
| 希尔排序 | 插入排序 | 数组 | 不稳定 | 根据步长序列的不同而不同 | 根据步长序列的不同而不同。已知最好的：$O(n\log^2 n)$ | $O(n)$ | 根据步长序列的不同而不同。 | 总共$O(n)$，需要辅助空间$O(1)$ |
| 直接选择排序 | 选择排序 | 数组 | 不稳定 | $n(n-1)/2$ | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | 总共$O(n)$，需要辅助空间$O(1)$ |
| 堆排序 | 选择排序 | 数组 | 不稳定 | / | $O(n\log^2 n)$ | $O(n\log^2 n)$ | $O(n\log^2 n)$ | 总共$O(n)$，需要辅助空间$O(1)$ |
| 锦标赛排序 | 选择排序 | 数组 | 不稳定 | $O(n\log^2 n)$ | $O(n\log^2 n)$ | $O(n\log^2 n)$ | $2n-1$ | |
| 冒泡 | 交换 | 数 | 稳 | | | | 总共$O(n)$，需 | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 排序 | 排序 | 组 | 定 | $O(n^2)$ | $O(n)$ | $O(n^2)$ | 要辅助空间$O(1)$ | |
| 快速排序 | 交换排序 | 不定 | 不稳定 | $n^2/2$ | $O(n^2)$ | $O(n\log^2 n)$ | $O(n\log^2 n)$ | 根据实现的方式不同而不同 |
| 归并排序 | / | 数组 | 稳定 | *right*-*left*+1 | $O(n\log^2 n)$ | $O(n)$ | $O(n\log^2 n)$ | $O(n)$ |
| 基数排序 | / | 数组 | 稳定 | 0 | $O(kn)$ | $O(kn)$ | $O(kn)$ | $O(kn)$ |

### 文件目录

- 10_1552651_wangyirui.cpp（主文件）
- 10_1552651_wangyirui.exe（可执行文件）
- 10_1552651_wangyirui.pdf（项目文档）

---

# 实现

### main函数

```
int main() {

    int n;
    printf("请输入要产生的随机数个数: ");
    scanf("%d", &n);
    printf("\n**                  排序问题                    **\n");
    printf("==============================================\n");
    printf("**                 请选择排序方法               **\n");
    printf("**                1---冒泡排序                  **\n");
    printf("**                2---选择排序                  **\n");
    printf("**                3---插入排序                  **\n");
    printf("**                4---希尔排序                  **\n");
```

```
    printf("**              5---快速排序                **\n");
    printf("**              6---堆排序                  **\n");
    printf("**              7---归并排序                **\n");
    printf("**              8---基数排序                **\n");
    printf("**              9---折半插入排序            **\n");
    printf("**              10--锦标赛排序              **\n");
    printf("**              11--退出程序                **\n");
    printf("============================================\n\n");

    vector<int>num(n + 5);
    vector<int>sorted(n + 5);

    for (int j = 0; j < n; ++j)
        num[j] = rand();

    clock_t clockstart, clockend;
    double timeused;

    while (true) {

        int op;
        time_t t;
        int ts = 0;
        printf("\n请选择排序算法:          ");
        scanf("%d", &op);

        if (op == 1) {

            clockstart = clock();
            bubblesort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("冒泡排序所用时间:         %lf\n", timeused);
            printf("冒泡排序比较次数:         %d\n", ts);
        }

        if (op == 2) {

            clockstart = clock();
            selectionsort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("选择排序所用时间:         %lf\n", timeused);
            printf("选择排序比较次数:         %d\n", ts);
        }
```

```
        if (op == 3) {

            clockstart = clock();
            insertionsort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("插入排序所用时间:         %lf\n", timeused);
            printf("插入排序比较次数:         %d\n", ts);
        }

        if (op == 4) {

            clockstart = clock();
            shellsort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("希尔排序所用时间:         %lf\n", timeused);
            printf("希尔排序比较次数:         %d\n", ts);
        }

        if (op == 5) {

            clockstart = clock();
            quicksort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("快速排序所用时间:         %lf\n", timeused);
            printf("快速排序比较次数:         %d\n", ts);
        }

        if (op == 6) {

            clockstart = clock();
            heapsort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("堆排序所用时间:          %lf\n", timeused);
            printf("堆排序比较次数:          %d\n", ts);
        }

        if (op == 7) {

            clockstart = clock();
            mergesort(num, sorted, n, ts);
```

```
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("归并排序所用时间:         %lf\n", timeused);
            printf("归并排序比较次数:         %d\n", ts);
        }
        if (op == 8) {

            clockstart = clock();
            radixsort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("基数排序所用时间:         %lf\n", timeused);
            printf("基数排序比较次数:         %d\n", ts);
        }
        if (op == 9) {

            clockstart = clock();
            binaryinsertsort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("折半插入排序所用时间:     %lf\n", timeused);
            printf("折半插入排序比较次数:     %d\n", ts);
        }
        if (op == 10) {

            clockstart = clock();
            tournamentsort(num, sorted, n, ts);
            clockend = clock();
            timeused = (clockend - clockstart) / static_cast<double>(CLOCKS_PER_SE
C);
            printf("锦标赛排序所用时间:       %lf\n", timeused);
            printf("锦标赛排序比较次数:       %d\n", ts);
        }
        if(op == 11)
            break;
    }
    return 0;
}
```

## 排序函数

### 直接插入排序

```
void insertionsort(vector<int> A, vector<int> &B, int n, int &ts) {

    int temp;
    for (int i = 1; i < n; ++i) {
        temp = A[i];
        int j = i - 1;
        for (; (++ts, j >= 0 && temp < A[j]); --j)
            A[j + 1] = A[j];
        A[j + 1] = temp;
    }
    B = A;
    return;
}
```

### 折半插入排序

```
void binaryinsertsort(vector<int> A, vector<int> &B, int n, int &ts) {

    int temp, low, high, mid, i, j;
    for (i = 1; i < n; ++i) {

        low = 0;
        high = i - 1;
        temp = A[i];

        while (low <= high) {
            mid = (low + high) / 2;
            if ((++ts, A[mid] > temp))
                high = mid - 1;
            else
                low = mid + 1;
        }

        for (j = i - 1; j > high; --j)
            A[j + 1] = A[j];

        A[j + 1] = temp;
    }
    B = A;
    return;
}
```

### 希尔排序

```
void shellsort(vector<int> A, vector<int> &B, int n, int &ts) {

    int hibbard[1000];
    int temp, k = 0;
    sethibbard(n, hibbard, k);
    for (int i = k; i >= 0; --i) {
        int gap = hibbard[i];
        for (int j = gap; j < n; ++j) {
            temp = A[j];
            int l = j - gap;
            for (; (++ts, l >= 0 && A[l] > temp); l -= gap) {
                A[l + gap] = A[l];
            }
            A[l + gap] = temp;
        }
    }
    B = A;
    return;
}
```

### 直接选择排序

```
void selectionsort(vector<int> A, vector<int> &B, int n, int &ts) {

    int min;
    for (int i = 0; i < n - 1; ++i) {
        min = i;
        for (int j = i + 1; j < n; ++j) {
            if ((++ts, A[min] > A[j]))
                min = j;
        }
        swap(A[i], A[min]);
    }
    B = A;
    return;
}
```

### 堆排序

```
void heapsort(vector<int> A, vector<int> &B, int n, int &ts) {

    for (int i = n / 2 - 1; i >= 0; --i)
        maxheapify(A, i, n - 1, ts);
    for (int i = n - 1; i > 0; --i) {
        swap(A[0], A[i]);
        maxheapify(A, 0, i - 1, ts);
    }
    B = A;
    return;
}
```

### 锦标赛排序

```
oid tournamentsort(vector<int> A, vector<int> &B, int n, int &ts) {


    bool flag = false;

    if (n % 2) {
        A[n] = INFINITY;
        n += 1;
        flag = true;
    }
    vector<node*>tree(2 * n - 1);

    for (int i = 0; i < n; ++i) {
        node *p = new node(A[i], i);
        tree[i + (n - 1)] = p;
    }
    for (int i = n - 2; i >= 0; --i) {
        node *p1 = (++ts, tree[2 * i + 1]->data < tree[2 * i + 2]->data) ? tree[2
* i + 1] : tree[2 * i + 2];
        node *p2 = new node(p1->data, p1->id);
        tree[i] = p2;
    }
    for (int i = 0; i < n - 1; ++i) {
        A[i] = tree[0]->data;
        int index = tree[0]->id;
        tree[index + (n - 1)]->data = INFINITY;
        for (int j = (index + (n - 1) - 1) / 2; j >= 0; j = (j - 1) / 2) {
            tree[j] = (++ts, tree[2 * j + 1]->data < tree[2 * j + 2]->data) ? tree
[2 * j + 1] : tree[2 * j + 2];
            if (j == 0) break;
        }
    }

    if (!flag)
        A[n - 1] = tree[0]->data;
    for (int i = 0; i < n; ++i)
        delete tree[i + (n - 1)];

    B = A;
    return;
}
```

**冒泡排序**

```
void bubblesort(vector<int> A, vector<int> &B, int n, int &ts) {

    bool sorted = false;
    while (!sorted) {
        sorted = true;
        for (int i = 1; i < n; ++i) {
            if ((++ts, A[i - 1] > A[i])) {
                swap(A[i - 1], A[i]);
                sorted = false;
            }
        }
        n--;
    }
    B = A;
    return;
}
```

### 快速排序

```
void quicksort(vector<int> A, vector<int> &B, int n, int &ts) {

    if (n <= 1)
        return;
    stack<int> ranges;
    int l = 0;
    int r = n - 1;
    ranges.push(l);
    ranges.push(r);
    while (!ranges.empty()) {
        int tr = r = ranges.top(); ranges.pop();
        int tl = l = ranges.top(); ranges.pop();
        int pivot = A[tl];
        while (tr > tl) {
            while (tr > tl && (++ts, A[tr] > pivot))
                --tr;
            while (tr > tl && (++ts, A[tl] < pivot))
                ++tl;
            swap(A[tr], A[tl]);
        }
        A[tl] = pivot;
        if (tl - l >= 1) {
            ranges.push(l); ranges.push(tl);
        }
        if (r - (tl + 1) >= 1) {
            ranges.push(tl + 1); ranges.push(r);
        }
    }
    B = A;
    return;
}
```

### 归并排序

```
void mergesort(vector<int> A, vector<int> &B, int n, int &ts) {

    vector<int>b(n);
    for (int seg = 1; seg < n; seg += seg) {
        for (int start = 0; start < n; start += seg + seg) {
            int low = start, mid = min(start + seg, n), high = min(start + seg + s
eg, n);
            int start1 = low, end1 = mid;
            int start2 = mid, end2 = high;
            int k = low;
            while (start1 < end1 && start2 < end2)
                b[k++] = (++ts, A[start1] < A[start2]) ? A[start1++] : A[start2++]
;
            while (start1 < end1) {
                b[k++] = A[start1++];
            }
            while (start2 < end2) {
                b[k++] = A[start2++];
            }
        }
        auto temp = A;
        A = b;
        b = temp;
    }
    B = A;
    return;
}
```

### 基数排序

```
void radixsort(vector<int> A, vector<int> &B, int n, int &ts) {

    int d = maxbit(A, n);
    vector<int>temp(n);
    vector<int>count(10);
    int radix = 1;
    int i, j, k;
    for (i = 0; i < d; ++i) {
        for (j = 0; j < 10; ++j) {
            count[j] = 0;
        }
        for (j = 0; j < n; ++j) {
            k = (A[j] / radix) % 10;
            ++count[k];
        }
        for (int j = 1; j < 10; ++j) {
            count[j] = count[j - 1] + count[j];
        }
        for (j = n - 1; j >= 0; --j) {
            k = (A[j] / radix) % 10;
            temp[count[k] - 1] = A[j];
            --count[k];
        }
        A = temp;
        radix *= 10;
    }
    B = A;
    return;
}
```