一些很有用的思维方法与算法

- function_1

(一) 尺取法

尺取法又称双指针法

- 顾名思义，像尺子一样取一段，借用挑战书上面的话说，尺取法通常是对**数组保存一对下标**，即**所选取的区间的左右端**点，**然后根据实际情况不断地推进区间左右端点**以得出答案。之所以需要掌握这个技巧，是因为尺取法比直接暴力枚举区间效率高很多，尤其是数据量大的。
- 这种方法很类似于==蚯蚓的蠕动==。
- 图大概是这样的：

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14

---

case_1_方生气

题目描述`

方可元得到了员工们藏身的位置信息，有体力x点，他移动一格就会消耗一点体力，现在，他需要抓到最多的员工

输入格式`

第一行，包含两个整数n（0<n<=10000），x（0<x<=10000），len（0<len<=10000）分别表示藏身的员工数和方总的体力以及街区的长度。 接下来n行，每行一个整数，表示第i个藏身的员工的位置。

输出格式`

共两行，第一行包含两个整数，表示方可元从第几格走到第几格（第一格必须为员工的藏身点），第二行包含一个整数，表示抓到的员工数。

MyCode

---

*case_2_求和

题目

给定一个数组和一个数s，在这个数组中找一个区间，使得这个区间之和等于s

- 如：数组int x[14] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};和一个s = 15。那么，可以找到的区间就应该有0到4， 3到5， 6到7.（注意这里的下标从0开始）

正常的思维：

```
//**先用一个数组sum[i]存放前i个元素的和，其实现用的是"递推思想"**
sum[0] = x[0];//x为给定的原数组
for(int i = 1; i < n; i++){
   sum[i] += sum[i-1];//递推思想
}
//**然后通过两层循环求解**
for(int i = 0; i < n; i++)
    for(int j = n-1; j >= 0; j--)
    {
        if(sum[j]-sum[i]==s)
            printf("%d---%d\n", i, j);
    }
```

- 那么，用"尺取法"做上面这道题思路应该是这样的：

1. 用一对脚标i, j。**最开始都指向第一个元素。**
2. **如果区间i到j之和比s小，就让j往后挪一位**，并把sum的值加上这个新元素。相当于蚯蚓的头向前伸了一下。
3. **如果区间i到j之和比s大，就让sum减掉第一个元素。**相当于蚯蚓的尾巴向前缩了一下。
4. 如果i到j之和刚好等于s，则输入。

用一张图来表示就是这样的，每一行的黄色部分代表本次循环选中的区间:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	**2**	3	4	5	6	7	8	9	10	11	12	13	14
1	**2**	**3**	4	5	6	7	8	9	10	11	12	13	14
1	**2**	**3**	**4**	5	6	7	8	9	10	11	12	13	14
1	**2**	**3**	**4**	5	6	7	8	9	10	11	12	13	14
1	**2**	**3**	**4**	**5**	6	7	8	9	10	11	12	13	14
1	**2**	**3**	**4**	**5**	**6**	7	8	9	10	11	12	13	14
1	2	**3**	**4**	**5**	**6**	7	8	9	10	11	12	13	14
1	2	3	**4**	**5**	**6**	7	8	9	10	11	12	13	14
1	2	3	4	**5**	**6**	7	8	9	10	11	12	13	14
1	2	3	4	**5**	**6**	**7**	8	9	10	11	12	13	14
1	2	3	4	5	**6**	**7**	8	9	10	11	12	13	14
1	2	3	4	5	**6**	**7**	**8**	9	10	11	12	13	14
1	2	3	4	5	6	**7**	**8**	9	10	11	12	13	14
1	2	3	4	5	6	7	**8**	9	10	11	12	13	14
1	2	3	4	5	6	7	**8**	**9**	10	11	12	13	14
1	2	3	4	5	6	7	8	**9**	10	11	12	13	14
1	2	3	4	5	6	7	8	**9**	**10**	11	12	13	14
1	2	3	4	5	6	7	8	9	**10**	11	12	13	14
1	2	3	4	5	6	7	8	9	**10**	**11**	12	13	14
1	2	3	4	5	6	7	8	9	10	**11**	12	13	14
1	2	3	4	5	6	7	8	9	10	**11**	**12**	13	14
1	2	3	4	5	6	7	8	9	10	11	**12**	13	14
1	2	3	4	5	6	7	8	9	10	11	**12**	**13**	14
1	2	3	4	5	6	7	8	9	10	11	12	**13**	14
1	2	3	4	5	6	7	8	9	10	11	12	**13**	**14**
1	2	3	4	5	6	7	8	9	10	11	12	13	**14**

other'sCode

(二) 背包

- 给体积V的背包,有n种物品,每种物品有k[i]个,价值w[i],占空间v[i]

- 想法 给一个参数f[l],当我的体积为l的时候,我放还是不放这个物品i
- 实现

**case_1_背包0.1*

**case_2_背包1.0*

题目描述``

有 N 件物品和一个容量为 M 的背包。第 i 件物品的重量是 W_i，价值是 D_i。求解将哪些物品装入背包可使这些物品的重量总和不超过背包容量，且价值总和最大。

输入格式``

第一行：物品个数 N 和背包大小 M。

第二行至第 $N+1$ 行：第 i 个物品的重量 W_i 和价值 D_i。

输出格式``

输出一行最大价值。

- **样例输入 #1**

```
4 6
1 4
2 6
3 12
2 7
```

- **样例输出 #1**

```
23
```

code

(三) 关于周围的计数

想要统计$ a_{ij} $的周围

想法: 定义一个$ vector_{ij} $表示上下左右或其他

- 如:表示上下左右可以:

```
int vector[4][2] = {{0,1},{0,-1},{1,0},{-1,0}}; //即上下左右
//运用:
for(int i = 0;i<n;i++)
    for(int j = 0;j<m;j++)
      for(int k = 0;k<4;k++)
        cnt[i][j] += a[i+vector[k][0]][j+vector[k][1]];
```

- 如:表示前后各两个可以:

```
int vector[4][2] = {{-2,0},{-1,0},{1,0},{2,0}}; //即前前后后
//运用:
for(int i = 0;i<n;i++)
    for(int j = 0;j<m;j++)
      for(int k = 0;k<4;k++)
        cnt[i][j] += a[i+vector[k][0]][j+vector[k][1]];
```

---

*case_1_扫雷

*case_2_数独

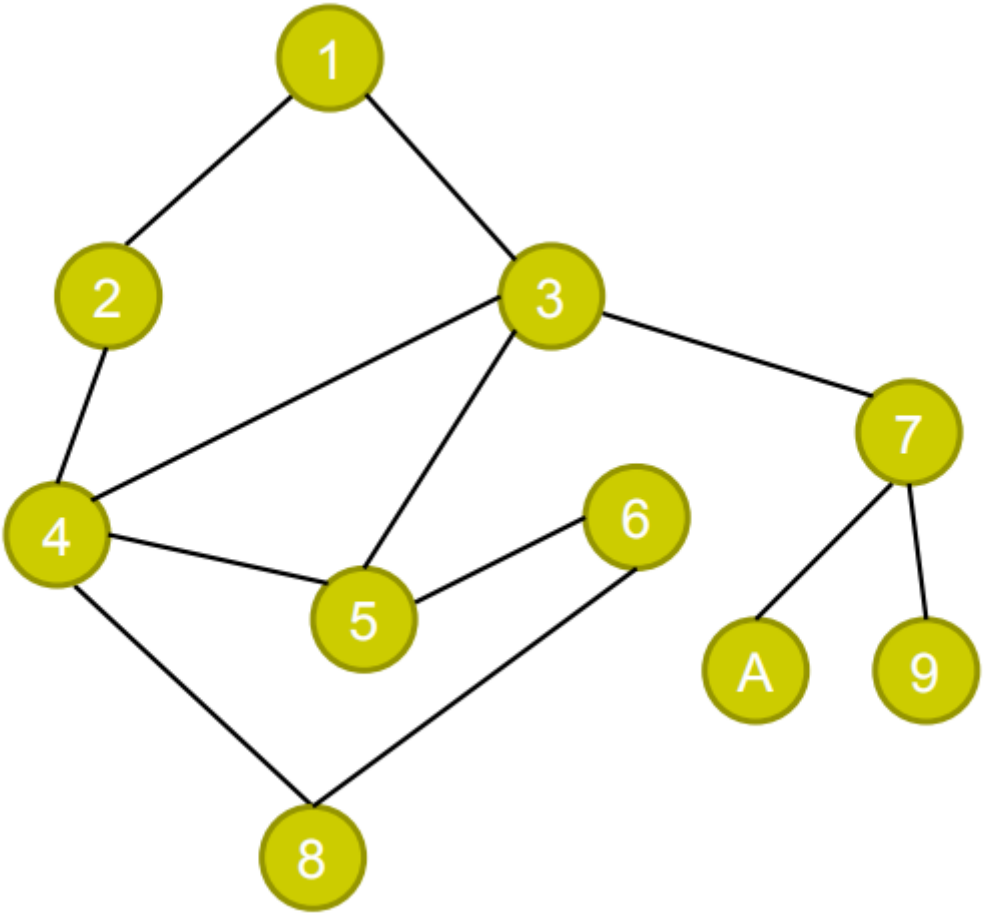
(四)遍历与排他的思考

ff

(五)DFS深度优先搜索算法

介绍

- 深度优先搜索算法（Depth First Search，简称DFS）：一种用于遍历或搜索树或图的算法。
- 沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点v的所在边都己被探寻过或者在搜寻时结点**不满足条件**，搜索将回溯到发现节点v的那条边的起始节点。整个进程反复进行直到**所有节点**都被访问为止。
- 属于盲目搜索,最糟糕的情况算法时间复杂度为$O(n!)$。



case_1

case_2

case_3

(六) 排序

菜鸟教程

需要问助教

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	O(n²)	O(n)	O(n²)	O(1)	In-place	稳定
选择排序	O(n²)	O(n²)	O(n²)	O(1)	In-place	不稳定
插入排序	O(n²)	O(n)	O(n²)	O(1)	In-place	稳定
希尔排序	O(n log n)	O(n log² n)	O(n log² n)	O(1)	In-place	不稳定
归并排序	O(n log n)	O(n log n)	O(n log n)	O(n)	Out-place	稳定
快速排序	O(n log n)	O(n log n)	O(n²)	O(log n)	In-place	不稳定
堆排序	O(n log n)	O(n log n)	O(n log n)	O(1)	In-place	不稳定
计数排序	O(n + k)	O(n + k)	O(n + k)	O(k)	Out-place	稳定
桶排序	O(n + k)	O(n + k)	O(n²)	O(n + k)	Out-place	稳定
基数排序	O(n × k)	O(n × k)	O(n × k)	O(n + k)	Out-place	稳定

名称	数据对象	稳定性	时间复杂度		额外空间复杂度	描述
			平均	最坏		
冒泡排序	数组	✓	$O(n^2)$		$O(1)$	（无序区，有序区）。 从无序区透过交换找出最大元素放到有序区前端。
选择排序	数组	✗	$O(n^2)$		$O(1)$	（有序区，无序区）。 在无序区里找一个最小的元素跟在有序区的后面。对数组：比较得多，换得少。
	链表	✓				
插入排序	数组、链表	✓	$O(n^2)$		$O(1)$	（有序区，无序区）。 把无序区的第一个元素插入到有序区的合适的位置。对数组：比较得少，换得多。
堆排序	数组	✗	$O(n \log n)$		$O(1)$	（最大堆，有序区）。 从堆顶把根卸出来放在有序区之前，再恢复堆。
归并排序	数组	✓	$O(n \log^2 n)$		$O(1)$	把数据分为两段，从两段中逐个选最小的元素移入新数据段的末尾。 可从上到下或从下到上进行。
			$O(n \log n)$		$O(n) + O(\log n)$ 如果不是从下到上	
	链表				$O(1)$	
快速排序	数组	✗	$O(n \log n)$	$O(n^2)$	$O(\log n)$	（小数，基准元素，大数）。 在区间中随机挑选一个元素作基准，将小于基准的元素放在基准之前，大于基准的元素放在基准之后，再分别对小数区与大数区进行排序。
希尔排序	数组	✗	$O(n \log^2 n)$	$O(n^2)$	$O(1)$	每一轮按照事先决定的间隔进行插入排序，间隔会依次缩小，最后一次一定要是1。
计数排序	数组、链表	✓	$O(n + m)$		$O(n + m)$	统计小于等于该元素值的元素的个数i，于是该元素就放在目标数组的索引i位（i≥0）。
桶排序	数组、链表	✓	$O(n)$		$O(m)$	将值为i的元素放入i号桶，最后依次把桶里的元素倒出来。
基数排序	数组、链表	✓	$O(k \times n)$	$O(n^2)$		一种多关键字的排序算法，可用桶排序实现。

ver_n

---

ver_00_猴子

ver_000_睡眠纯娱乐

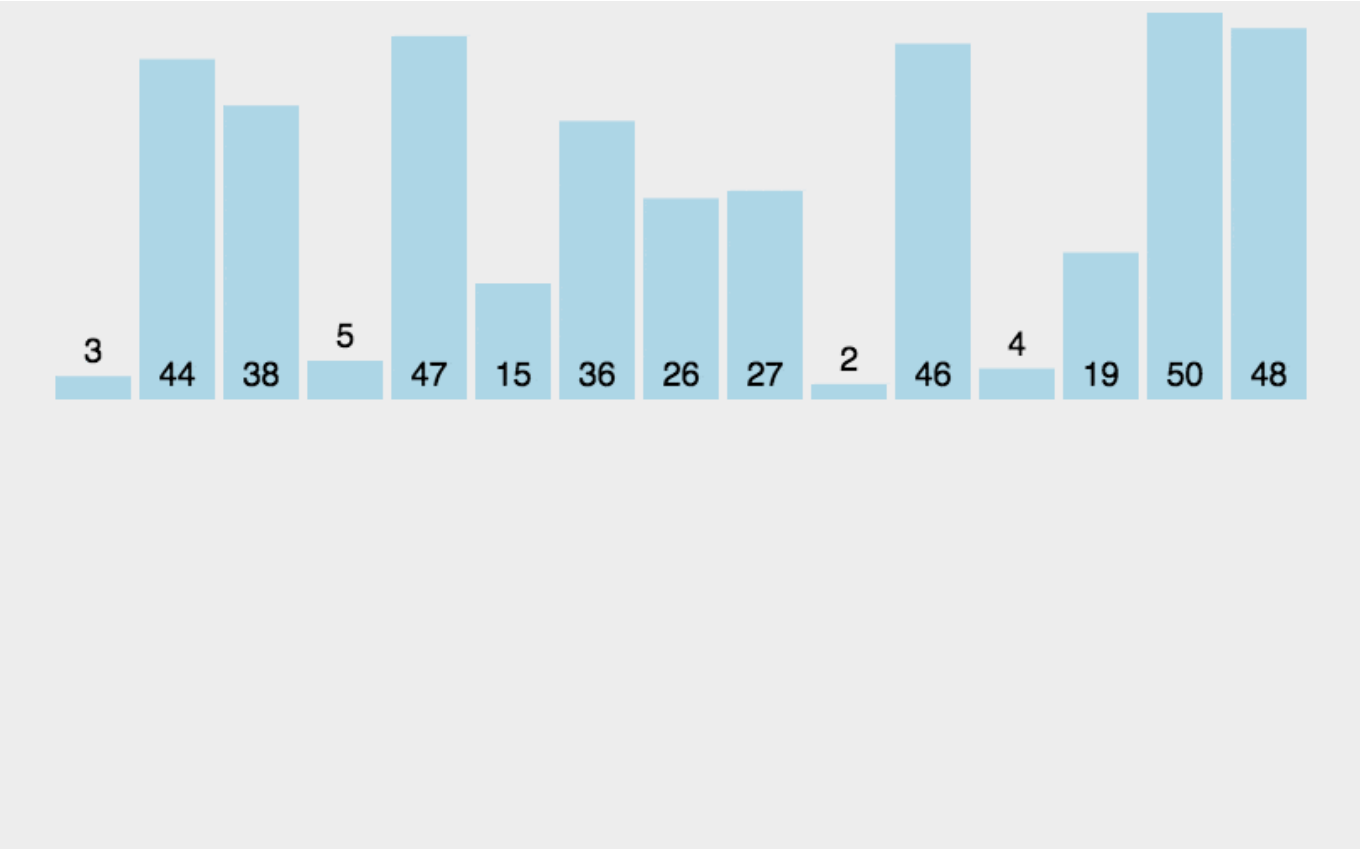
- 记录时间

---

ver_0_选择

ver_1_冒泡

ver_2_插入

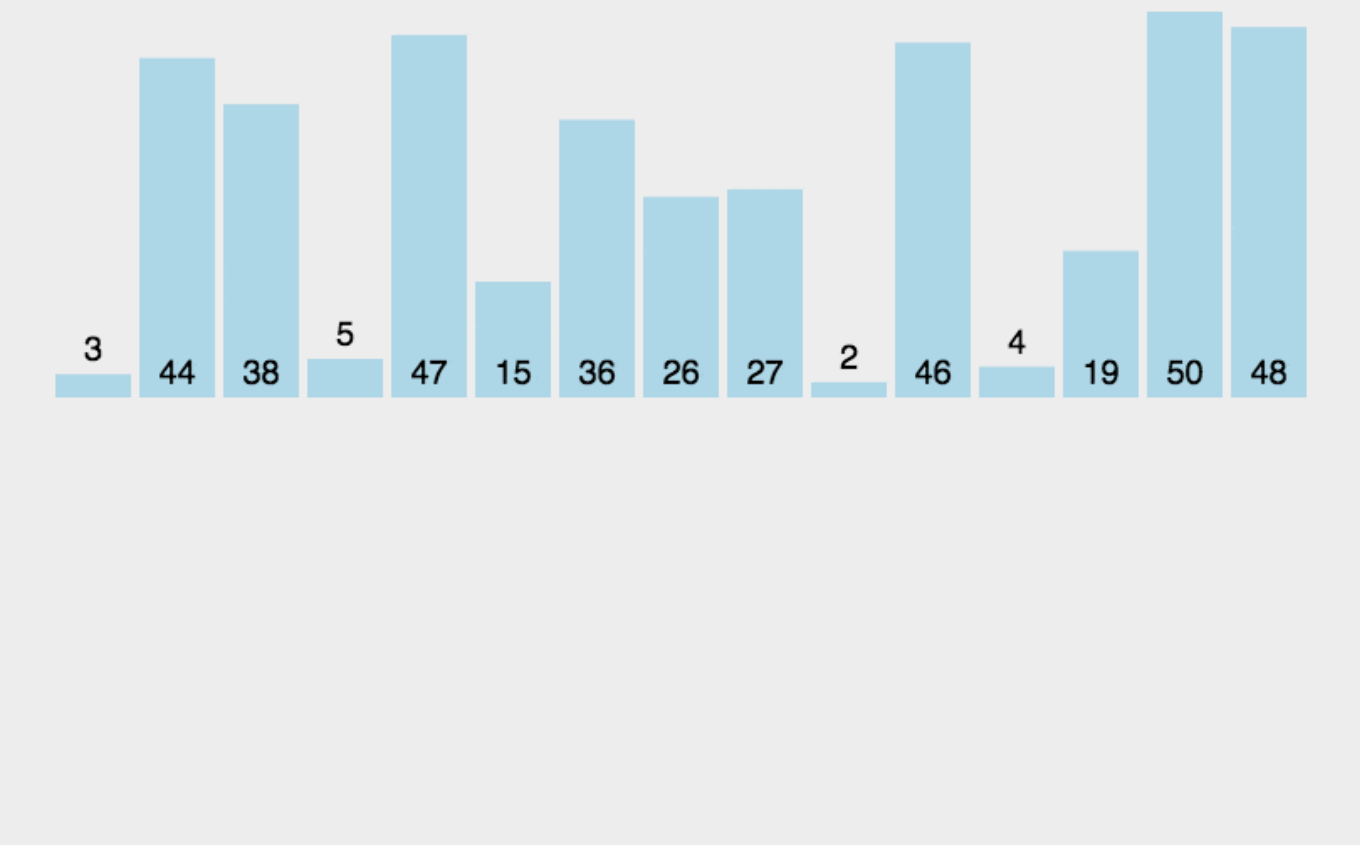


ver_2plus_希尔排序(非稳定)

- 可以理解成插入,从每次移一格的插入,变成每次移动x/2格,直到移动数为一格.

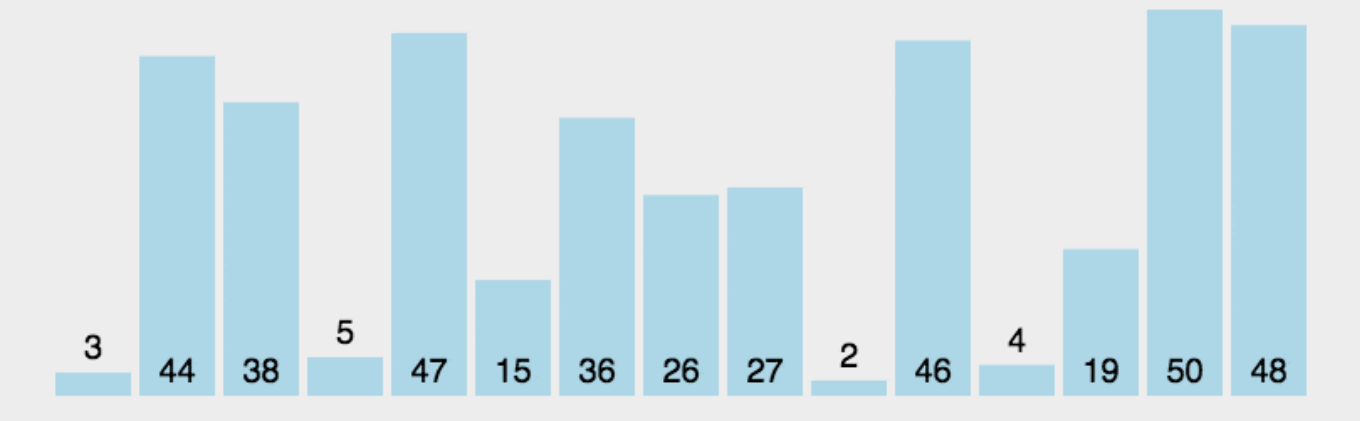
---

ver_3_归并



---

ver_4_快排 not code

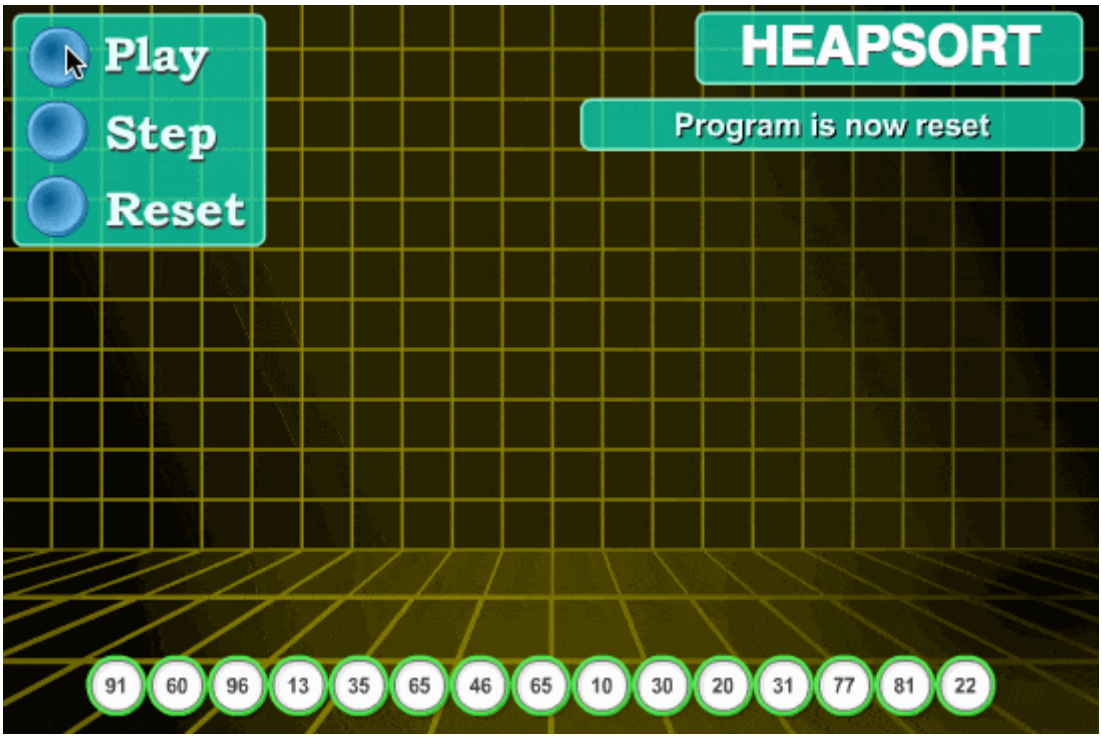


ver_4.1_二分快排他人

ver_4.2_二分快排助教思路

---

ver_5_堆排序



---

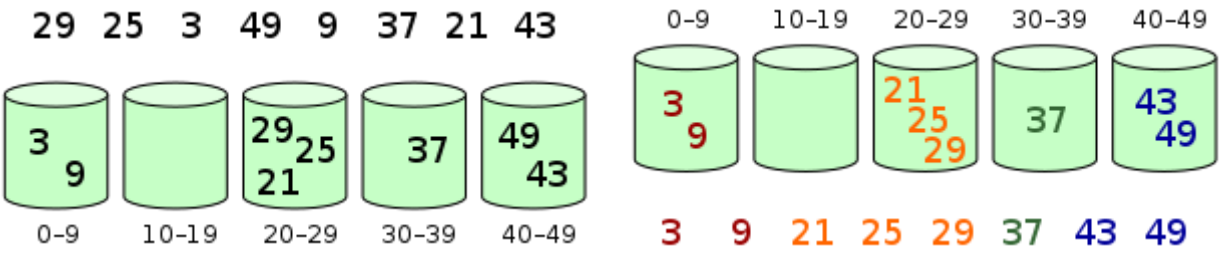
ver_6_非比较算法排序: 计数 桶 基数

ver_6.1_计数排序

- 就是下标计数

ver_6.2_桶排序

- 就是一个桶里面装的多一点



ver_6.3_基数排序

- 根据键值的每位数字来分配桶
