NCKU Programming Contest Training Course
Time Complexity & Sorting
2018/02/22

Syuan Yi, Lin (petermouse)
petermouselin@gmail.com

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

made by petermouse & DADA

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- **補充**
  - Quick sort
  - qsort()

Time complexity

A+B Problem

Time Limit: 1000MS **Memory Limit:** 10000K
Total Submissions: 342704 **Accepted:** 190305

Run ID	User	Problem	Result
12350938	106580	1840	Time Limit Exceeded
12350931	hncu110610230	3903	Time Limit Exceeded
12350906	20112685	3233	Time Limit Exceeded
12350904	20112685	3233	Time Limit Exceeded
12350899	hncu793116483	1833	Time Limit Exceeded
12350889	xk2741	3016	Time Limit Exceeded
12350859	superstarzhu	3461	Time Limit Exceeded
12350840	davidlee1999WTK	1251	Time Limit Exceeded
12350835	altair21	1811	Time Limit Exceeded
12350797	altair21	1811	Time Limit Exceeded
12350786	hncu793116483	1833	Time Limit Exceeded
12350773	block3	3993	Time Limit Exceeded
12350770	clbq2012	1273	Time Limit Exceeded

made by petermouse & DA

Time complexity

- Technical analysis

- Examples
- `for(i=0; i<n; i++)`

  `if(...) else ...`

- $O(n)$

Time complexity

- Technical analysis

- Examples
- ```
  for(i=0; i<n ;i++)
      for(j=0; j<n; j++)
          if(...) else ....
  ```

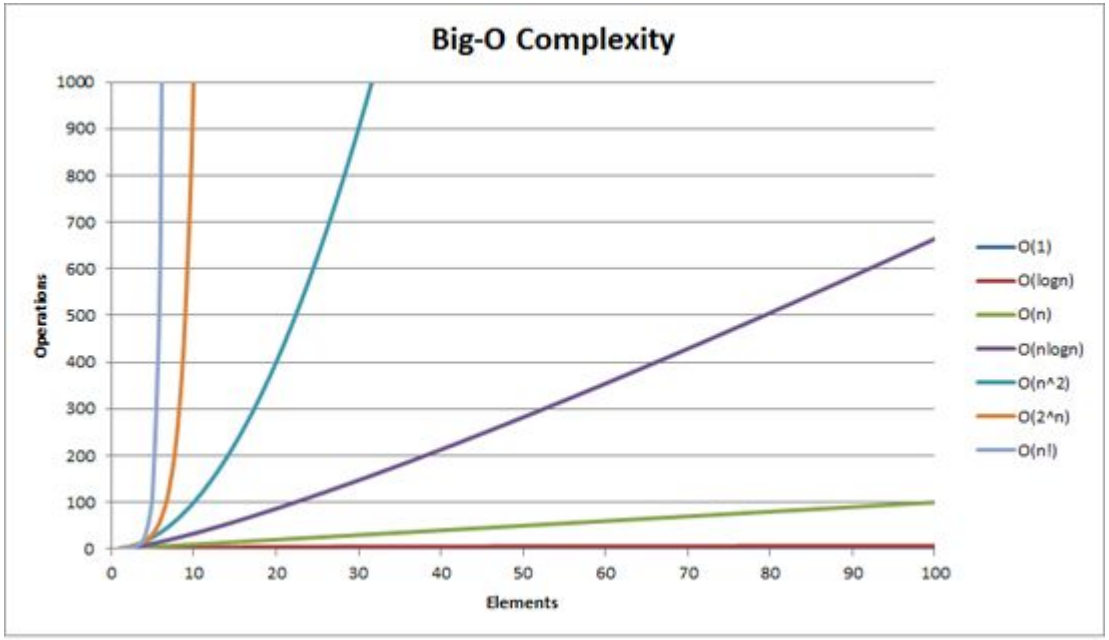
- $O(n^2)$
  ```

Time complexity

- Technical analysis

- Examples
- ```
  int two(int n){
          if(n < 2) return 1 << n;
      return two(n - 1)+ two(n - 1);
  }
  /* maximum n=M */
  ```

- $O(2^{M})$
  ```

Time complexity

- Given Input Size n = 1,000:
  - $O(n) = O(1{,}000)$
  - $O(n^2) = O(1{,}000{,}000)$
  - $O(nlgn)$ 近似於 $O(9965)$

- $O(1{,}000{,}000)$ 近似於 1 sec

Time complexity



圖片來源:
https://medium.com/journey-of-one-thousand-apps/complexity-and-big-o-notation-in-swift-478a67ba20e7

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- 補充
  - Quick sort
  - qsort()

Sorting

- 「排序演算法」
- 顧名思義就是將一串資料，依照特定的規則排序的演算法。
- 以下我們使用一個 int 數組 a[n]，由 a[0] 至 a[n-1] 共 n 項，欲將之由小到大排序。

穩定排序 v.s. 不穩定排序

- 比較時相同之資料(又或稱為鍵值相同)
- 在排序後相對位置與排序前相同時，稱穩定排序(Stable sort)
- 若在排序後相對位置與排序前「可能不相同」時，稱不穩定排序(Unstable sort)。
- 例如:2 3 5 1 3 4
  - 穩定排序:1 2 3 3 4 5
  - 不穩定排序: 1 2 3 3 4 5

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - Quick sort
  - STL
- **補充**
  - Quick sort
  - qsort()

Selection Sort

- 「選擇排序法」
- 和日常生活中會用到的方法一樣，不斷地找到最小值，再做移動，非常直觀。
- 時間複雜度 $O(n^2)$
- 為不穩定排序。

Selection Sort

- 1. 從 a[0] 跑到 a[n-1] 找出其中的最小值，再與 a[0] 交換，a[0] 便確定了。
- 2. 重複步驟1，從 a[1] 跑到 a[n-1] 找出其中的最小值，再與 a[1] 交換……
- 3. 如此跑完 n 次後就完成了排序。

Selection Sort

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
8	6	2	7	3	1	4	5
1	6	2	7	3	*8*	4	5
1	2	*6*	7	3	8	4	5
1	2	3	7	*6*	8	4	5
1	2	3	4	5	8	*7*	6
1	2	3	4	5	8	7	6
1	2	3	4	5	6	7	*8*
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

底線：已排序好的資料
灰底底線：上一個被排序好的資料
綠色斜體：被交換的資料。

made by petermouse & DA

Code

```
// Selection Sort
for (int i = 0; i < n - 1; i++) {
    int min_index = i;
    // Find index of minimum value
    for (int j = i + 1; j < n; j++) {
        if (array[j] < array[min_index]) {
            min_index = j;
        }
    }
    // Swap two values
    int temp = array[min_index];
    array[min_index] = array[i];
    array[i] = temp;
}
```


Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- **補充**
  - Quick sort
  - qsort()

Bubble Sort

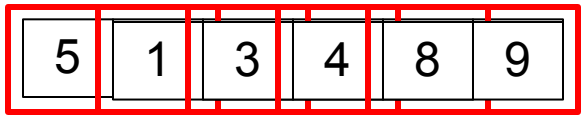
- 「氣泡排序法」(or 泡沫排序法, 冒泡排序法... whatever)
- 重複地走訪過要排序的數列, 一次比較兩個元素, 如果他們的順序錯誤就把他們交換過來。
- 時間複雜度 $O(n^2)$
- 為穩定排序。

Bubble Sort

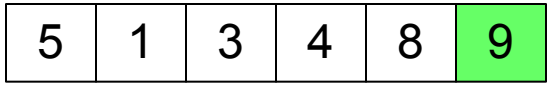
- 1. 從第一個元素開始，比較相鄰的兩個元素大小
- 2. 若第一元素大於第二元素，則交換位置
- 3. 每回合遞減需要比較的元素個數

Bubble Sort

- 範例

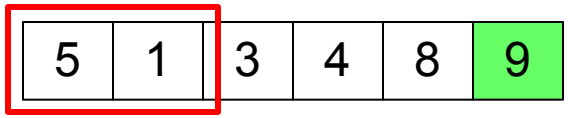


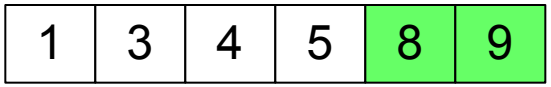
第一回合
1st iteration



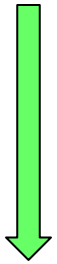
Bubble Sort

- 範例





第二回合
2nd iteration



第五回合
5th iteration

made by petermouse & DA

Bubble Sort

Code

```
// Bubble Sort
for (int i = n - 1; i > 0; i--) {
    for (int j = 0; j < i; j++) {
        if (array[j] > array[j + 1]) {
            int temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
        }
    }
}
```

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- **補充**
  - Quick sort
  - qsort()

Insertion Sort

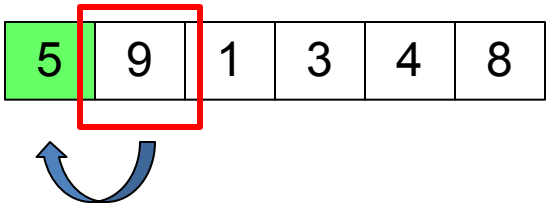
- 「插入排序法」
- 每次都只為一個元素找到目前(已排序的數列中)的最佳位置
- 時間複雜度 $O(n^2)$
- 為穩定排序。

Insertion Sort

- 1. 從第一個元素開始，該元素可以認為已經被排序
- 2. 取出下一個元素，在已經排序的元素序列中從後向前掃描
- 3. 如果該元素(已排序)大於新元素，將該元素移到下一位置
- 4. 重複步驟3，直到找到已排序的元素小於或者等於新元素的位置
- 5. 將新元素插入到該位置後
- 6. 重複步驟2~5

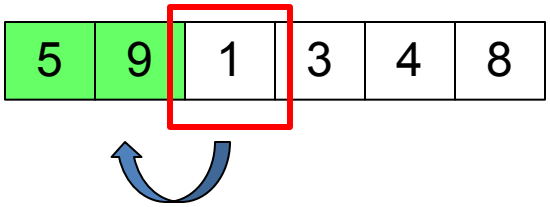
Insertion Sort

- 範例



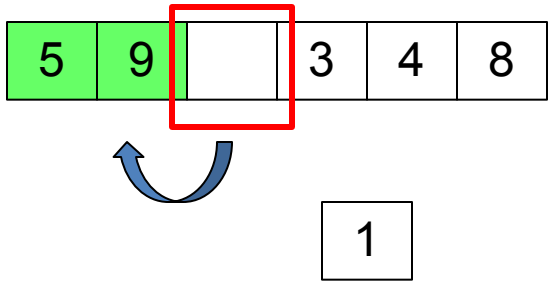
Insertion Sort

- 範例



Insertion Sort

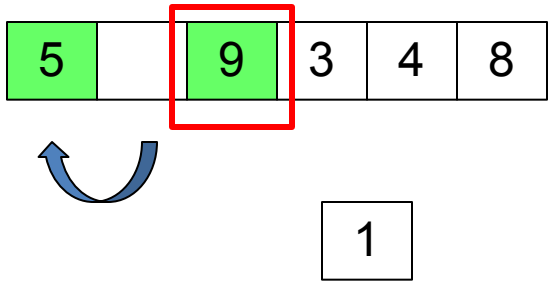
- 範例



made by petermouse & DA

Insertion Sort

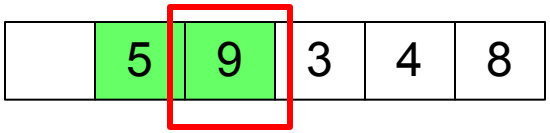
- 範例

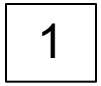


made by petermouse & DA

Insertion Sort

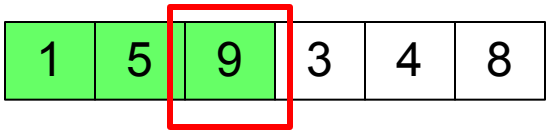
- 範例





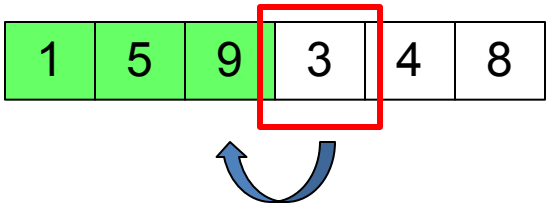
Insertion Sort

- 範例



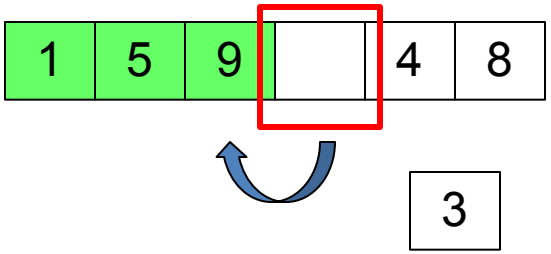
Insertion Sort

- 範例



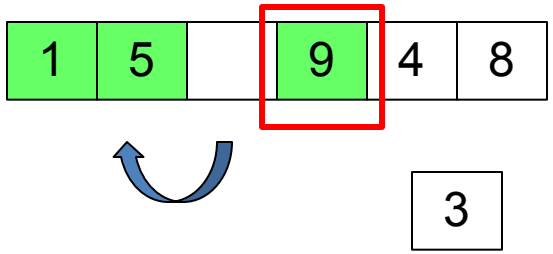
Insertion Sort

- 範例



Insertion Sort

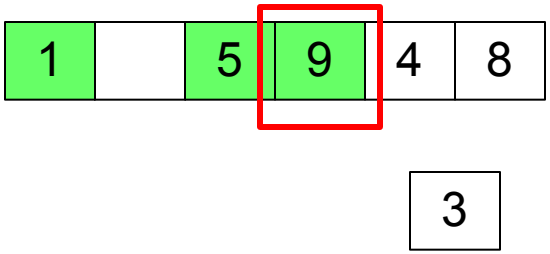
- 範例



made by petermouse & DA

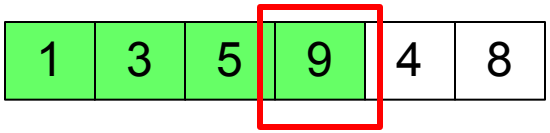
Insertion Sort

- 範例



Insertion Sort

- 範例



Result

Insertion Sort

- Code

```
// Insertion Sort
for (int i = 1; i < n; i++) {
    int temp = array[i];
    for (int j = i - 1; j >= 0; j--) {
        if (array[j] > temp)
            array[j + 1] = array[j];
        else
            break;
    }
    array[j + 1] = temp;
}
```

Example 1

Uva10327 – Flip Sort

Sorting in computer science is an important part. Almost every problem can be solved efficiently if sorted data are found. There are some excellent sorting algorithm which has already achieved the lower bound $nlgn$. In this problem we will also discuss about a new sorting approach. In this approach only one operation (Flip) is available and that is you can exchange two adjacent terms. If you think a while, you will see that it is always possible to sort a set of numbers in this way.

The Problem

A set of integers will be given. Now using the above approach we want to sort the numbers in ascending order. You have to find out the minimum number of flips required. Such as to sort "1 2 3" we need no flip operation whether to sort "2 3 1" we need at least 2 flip operations

Example 1

The Input

The input will start with a positive integer N (N<=1000). In next few lines there will be N integers. Input will be terminated by EOF.

The Output

For each data set print "Minimum exchange operations : M" where M is the minimum flip operations required to perform sorting. Use a seperate line for each case.

Sample Input

```
3
1 2 3
3
2 3 1
```

Sample Output

```
Minimum exchange operations : 0
Minimum exchange operations : 2
```

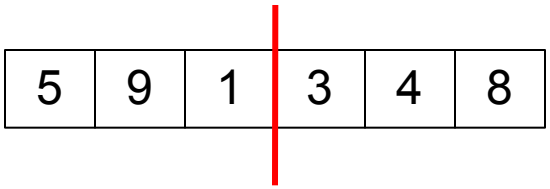

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- **補充**
  - Quick sort
  - qsort()

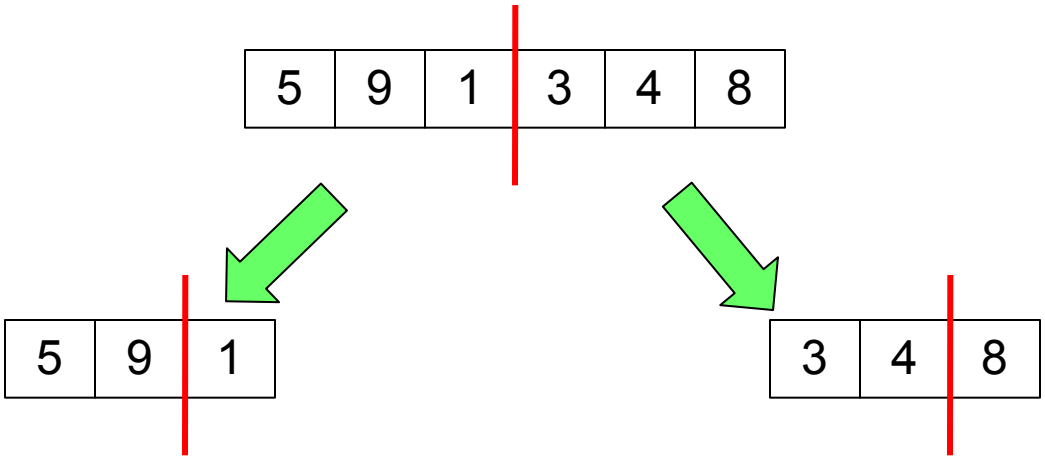
Merge Sort

- 「合併排序法」
- 利用分治法（Divide and Conquer）將要排序的資料均分成兩組，分別對兩邊做排序，然後再將兩邊「已排序的資料」再做合併，即完成排序。
- 時間複雜度 $O(n \lg n)$
- 還需要額外的記憶體做合併，空間複雜度$O(n)$
- 為穩定排序

Merge Sort

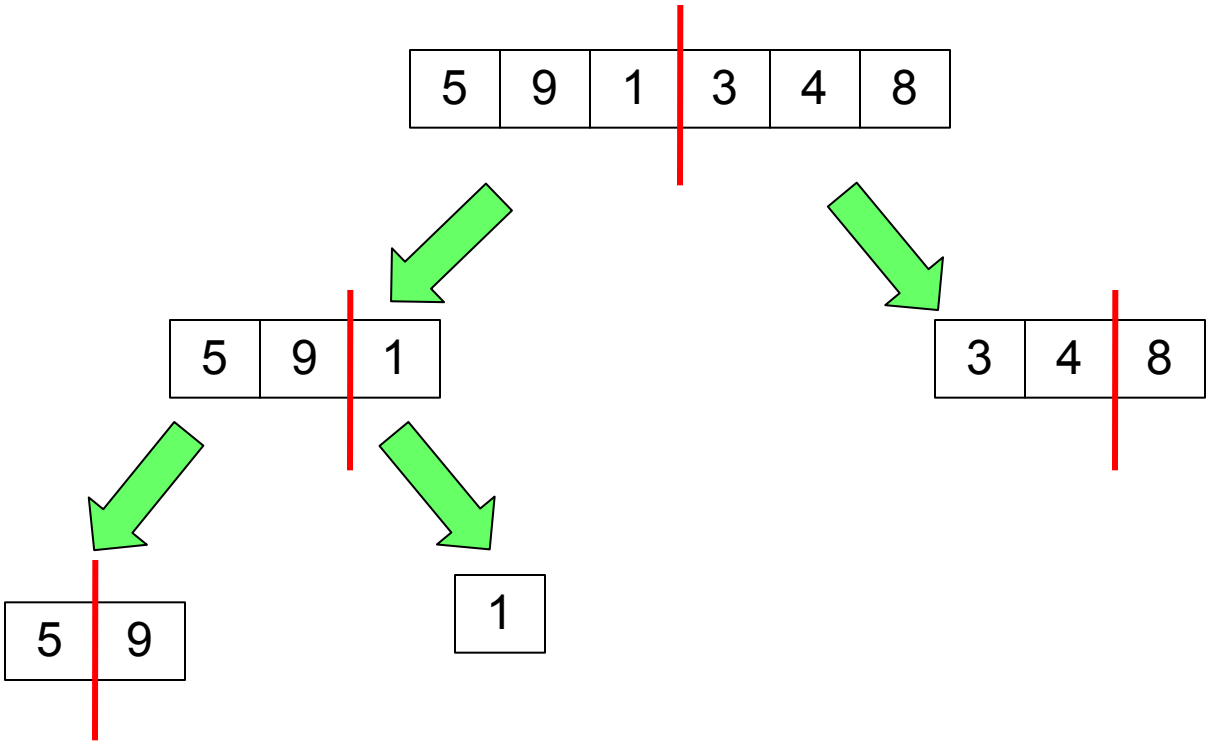


Merge Sort



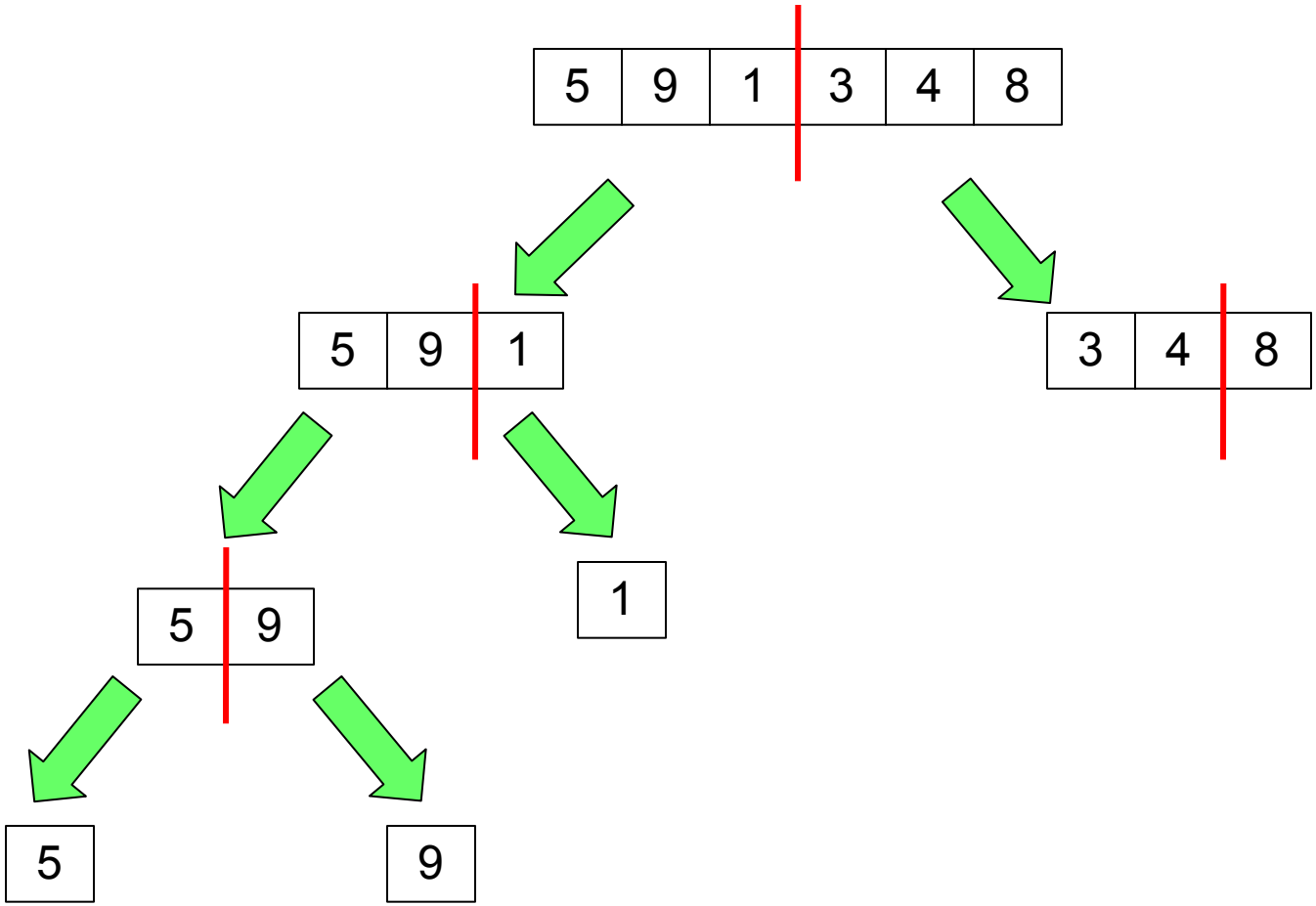
made by petermouse & DA

Merge Sort



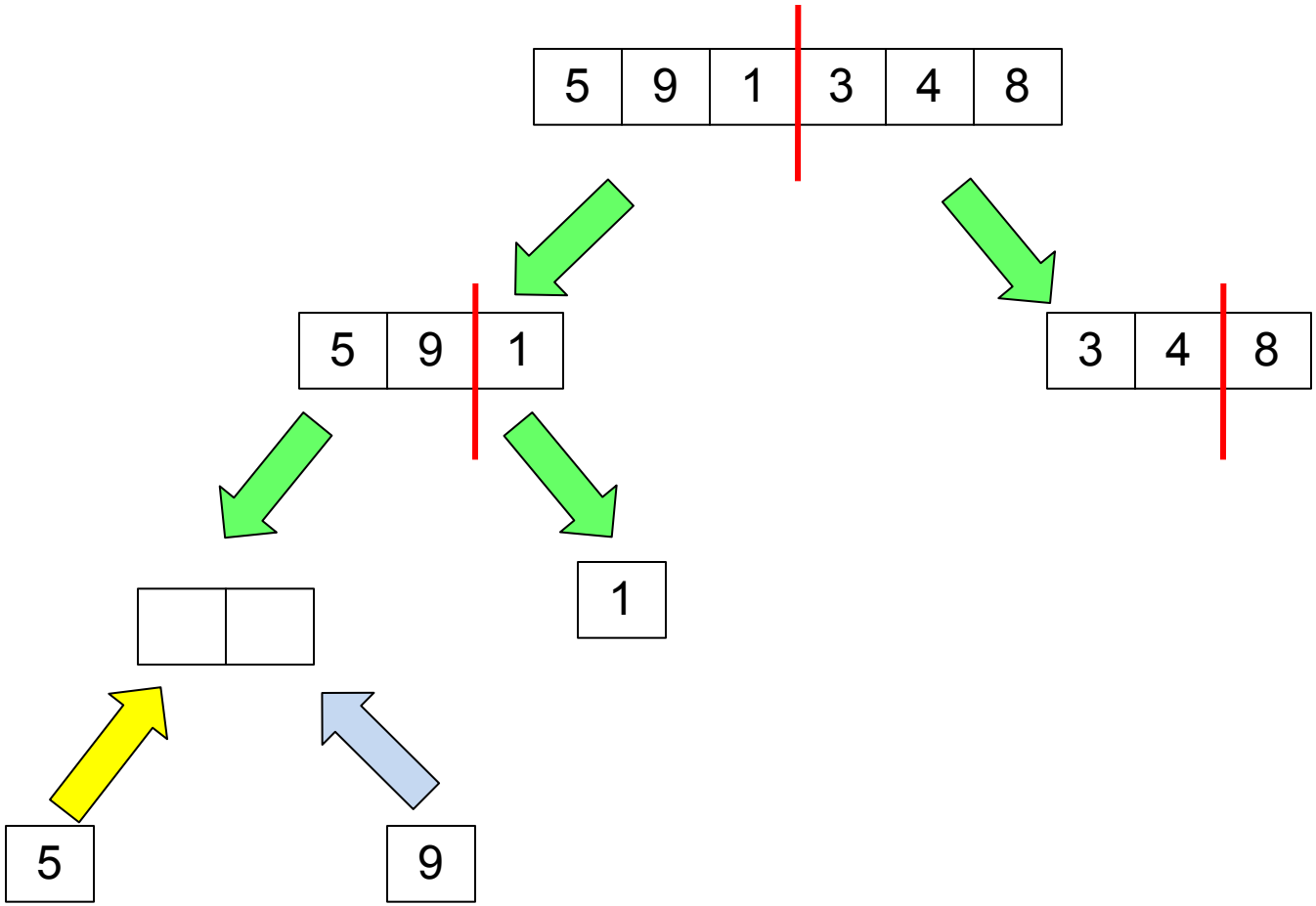
made by petermouse & DA

Merge Sort



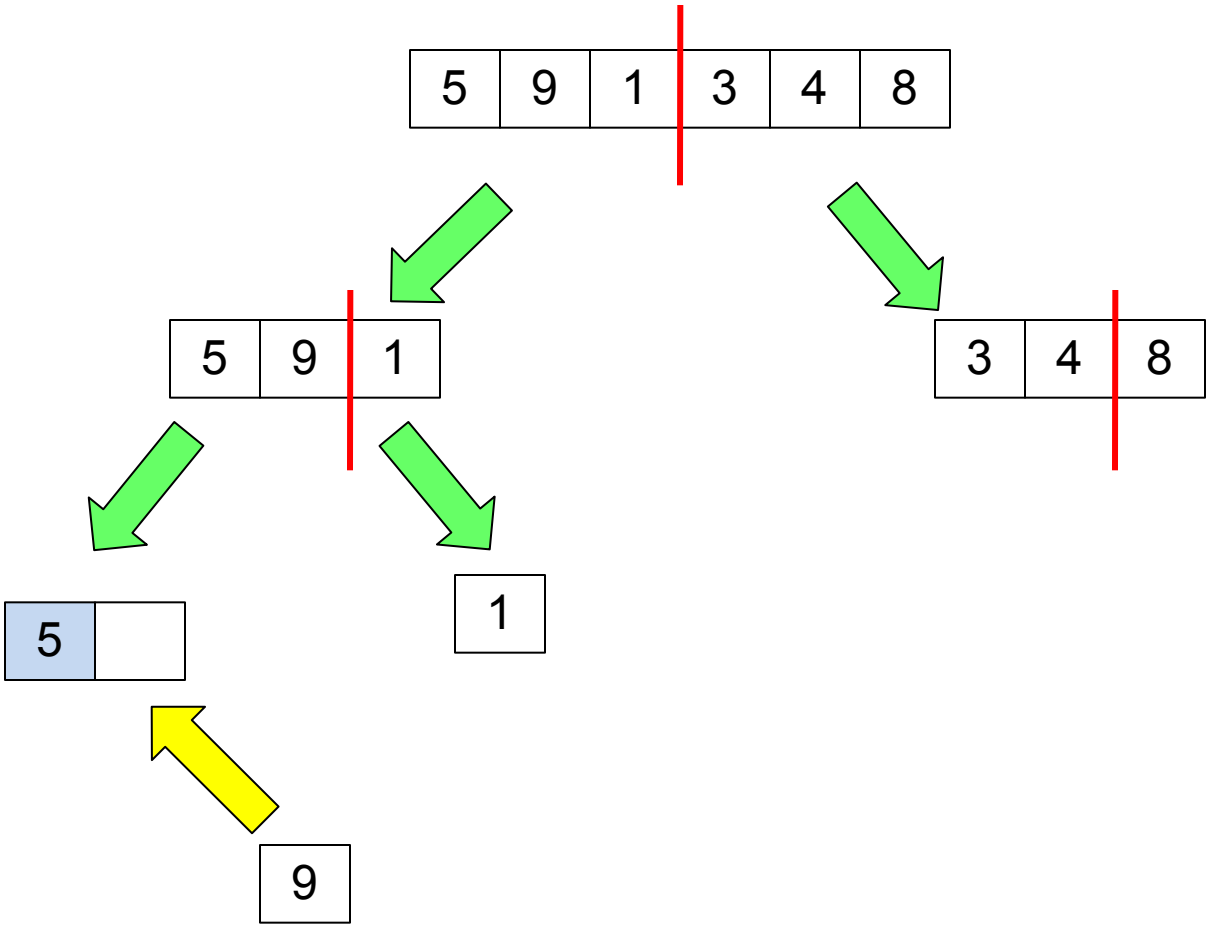
made by petermouse & DA

Merge Sort



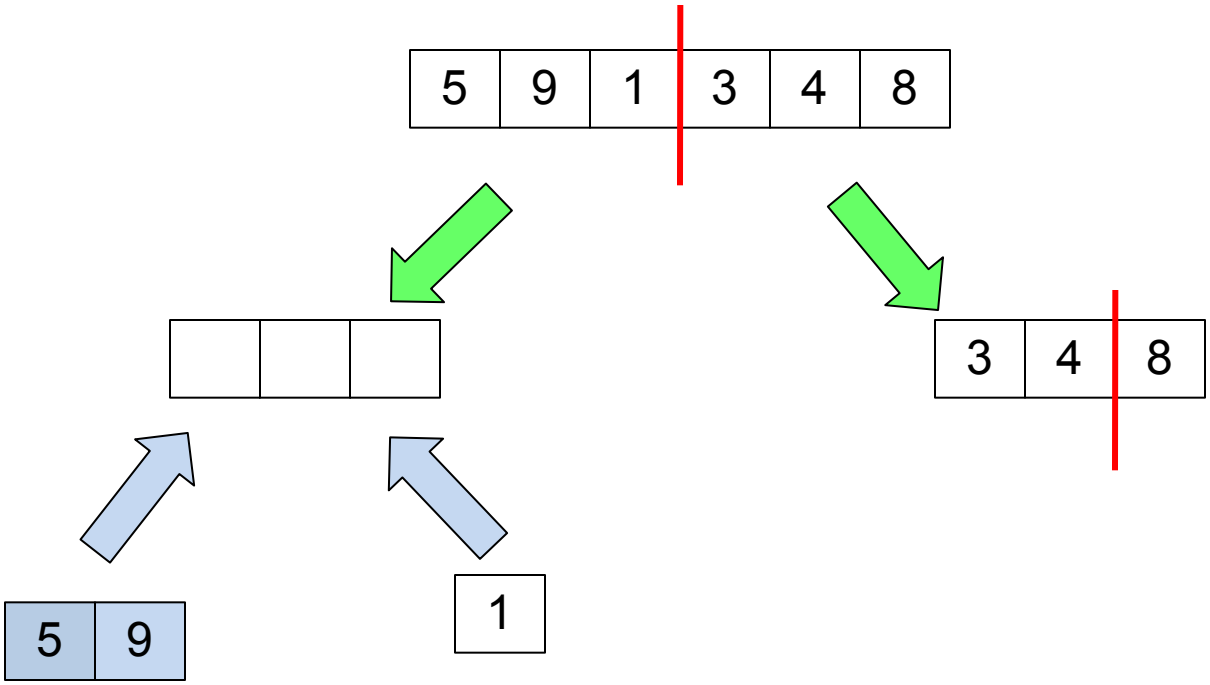
made by petermouse & DA

Merge Sort



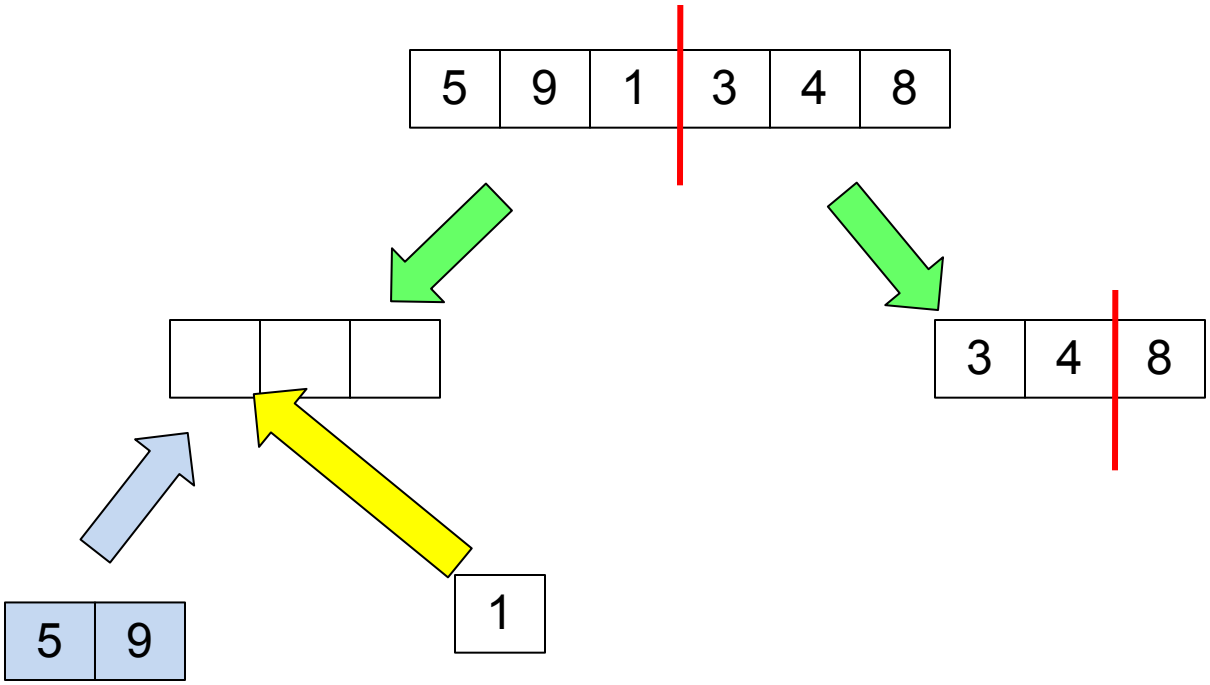
made by petermouse & DA

Merge Sort



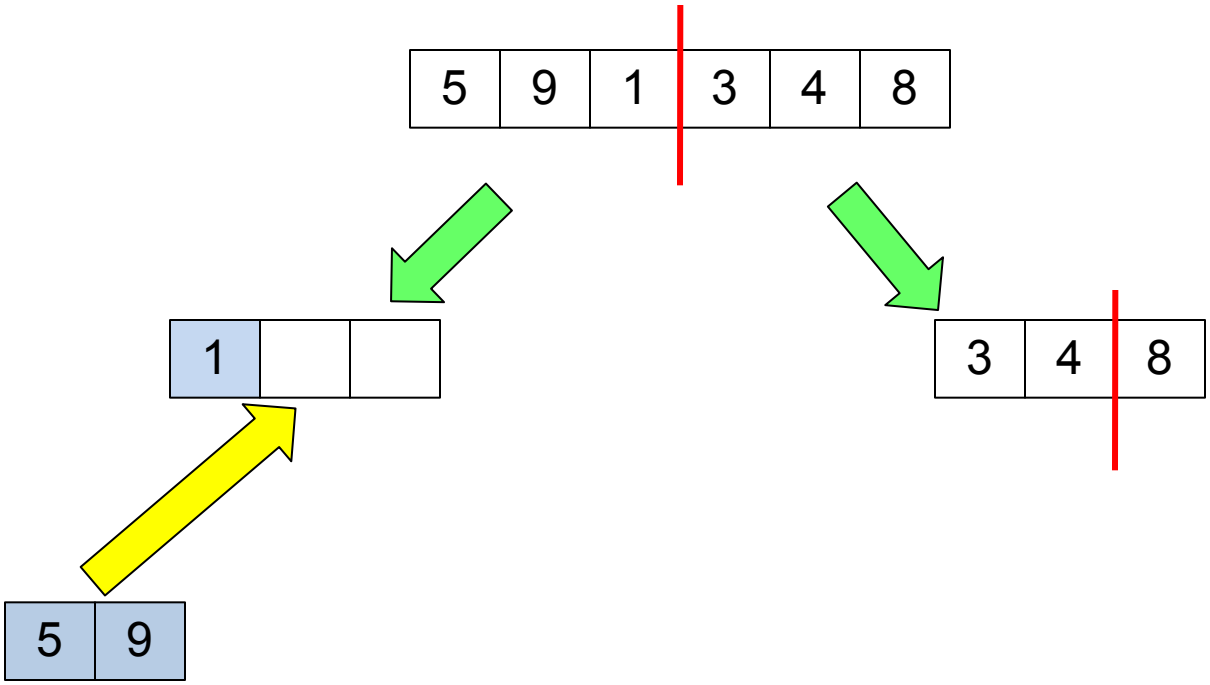
made by petermouse & DA

Merge Sort



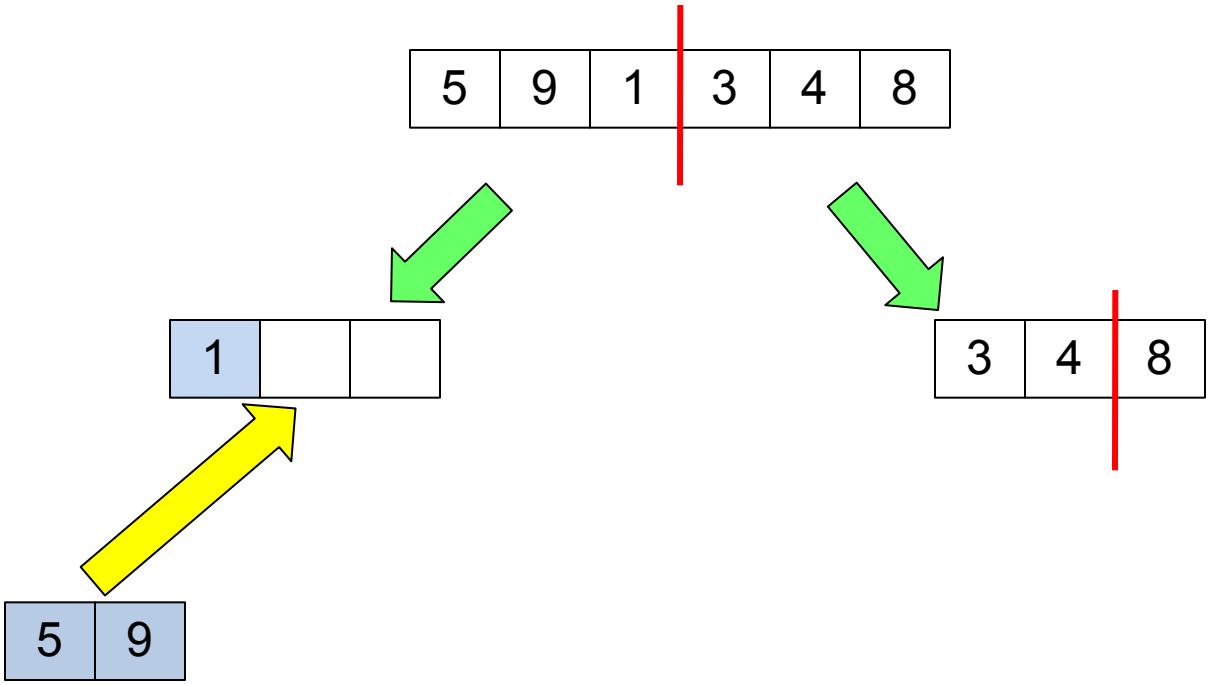
made by petermouse & DA

Merge Sort



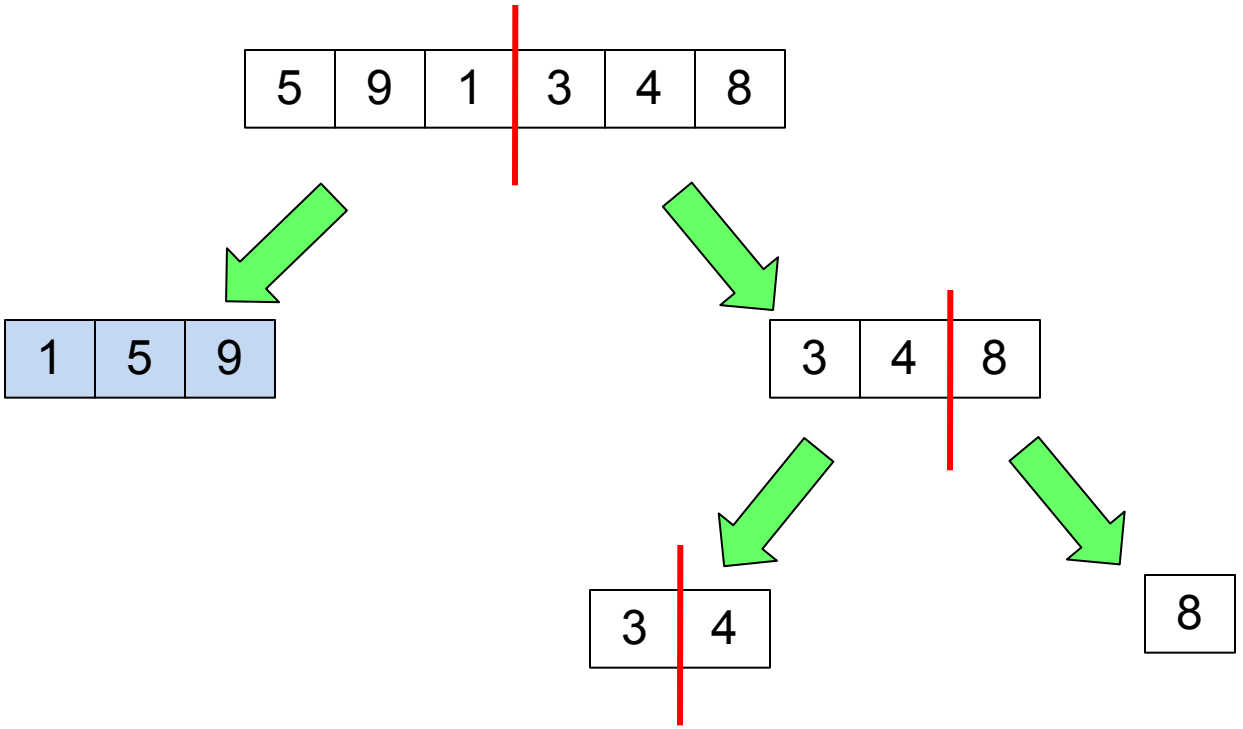
made by petermouse & DA

Merge Sort



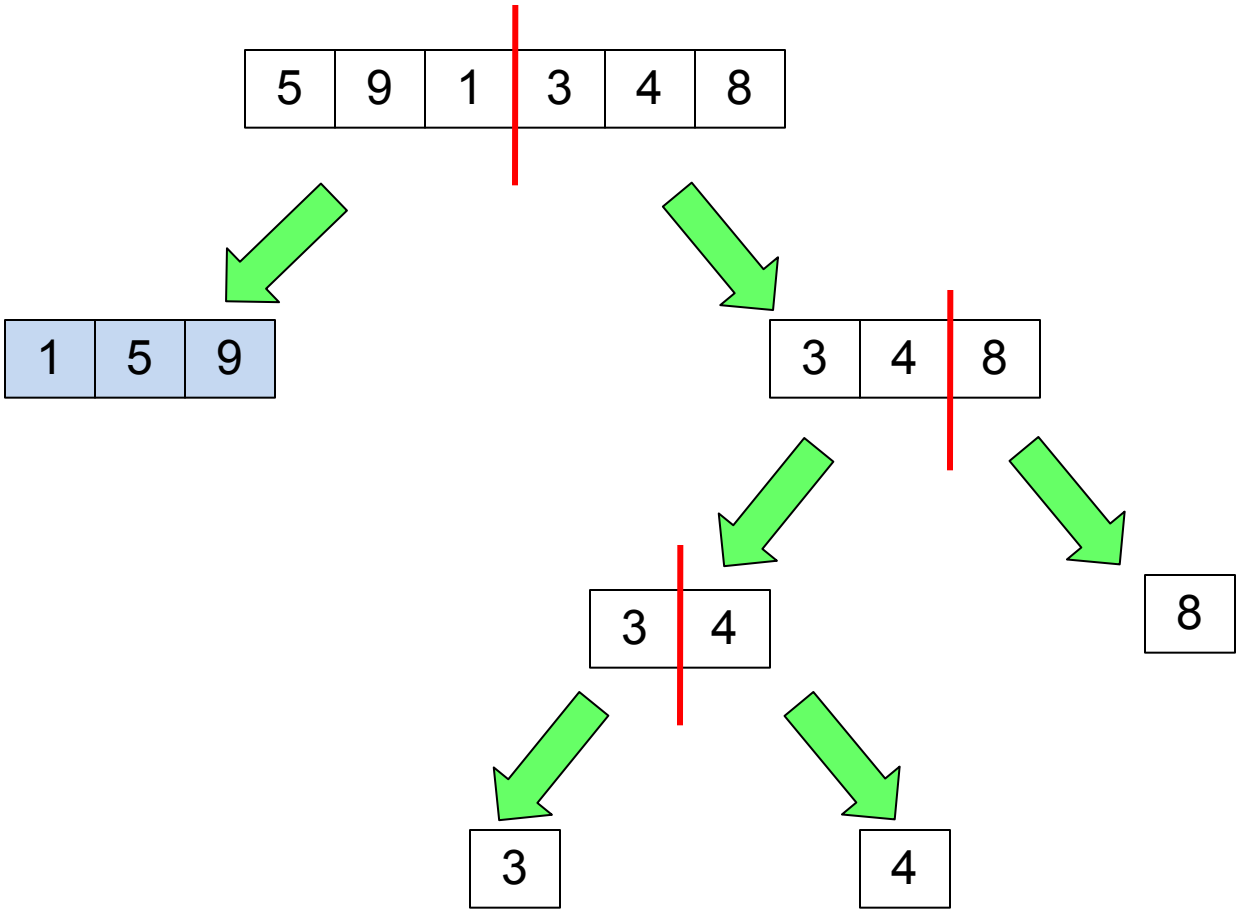
made by petermouse & DA

Merge Sort



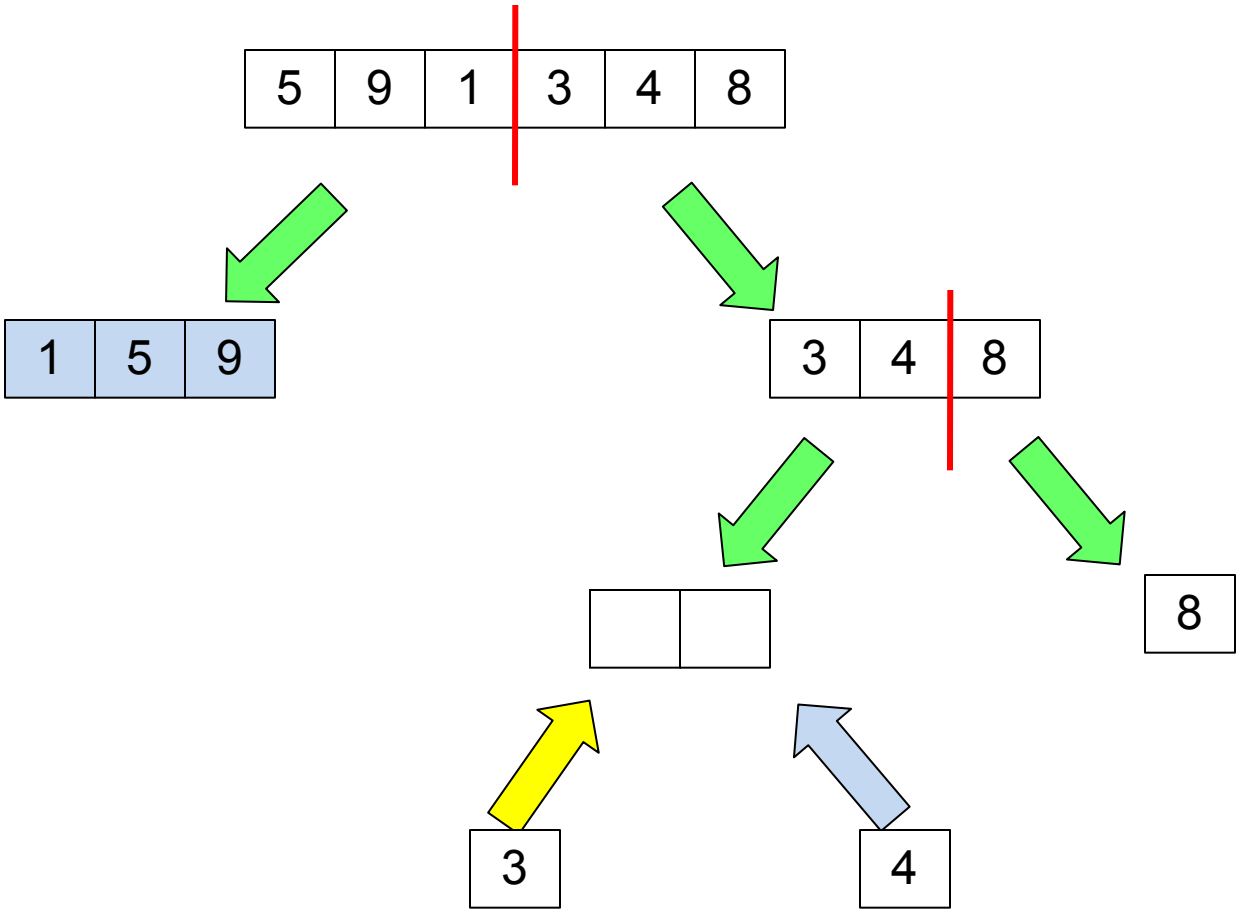
made by petermouse & DA

Merge Sort



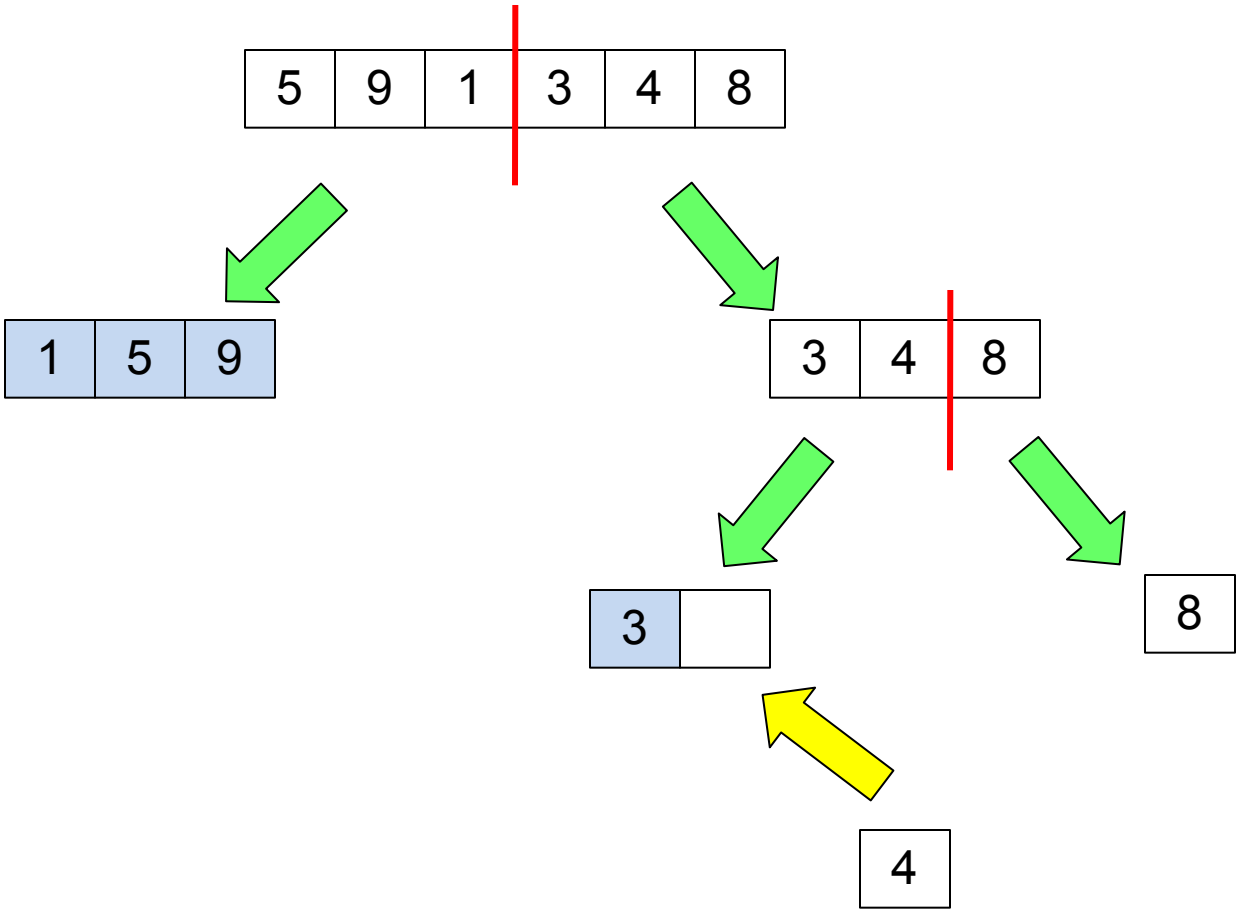
made by petermouse & DA

Merge Sort



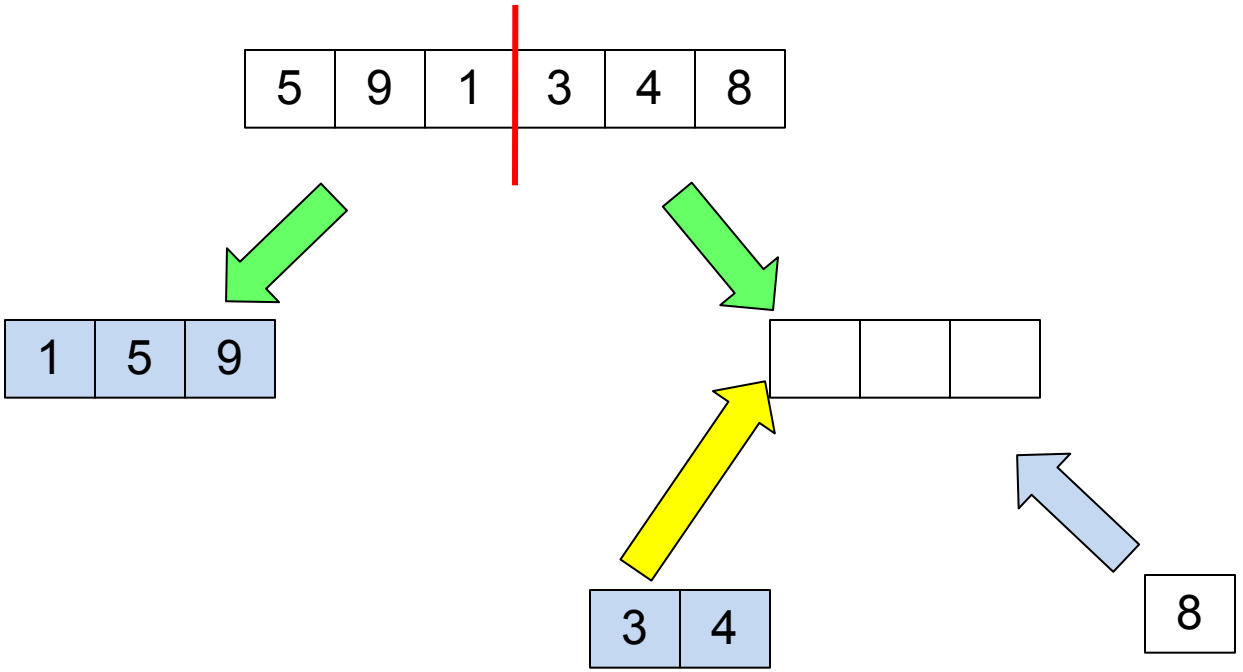
made by petermouse & DA

Merge Sort



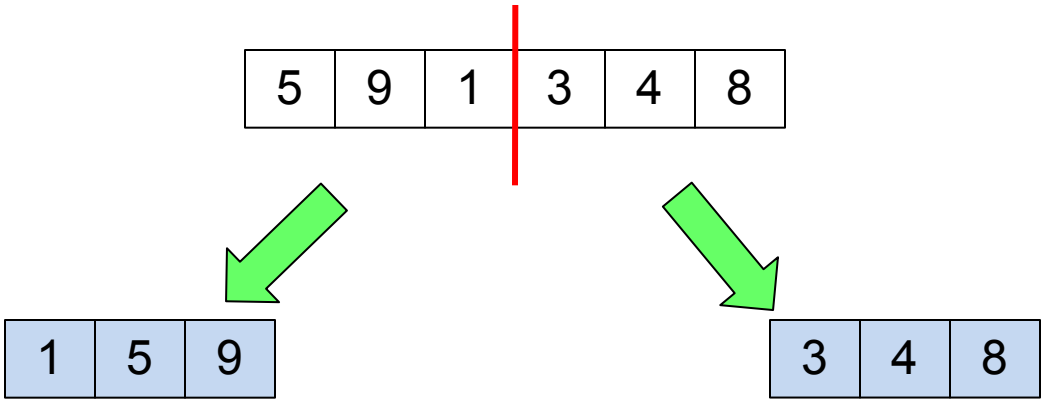
made by petermouse & DA

Merge Sort



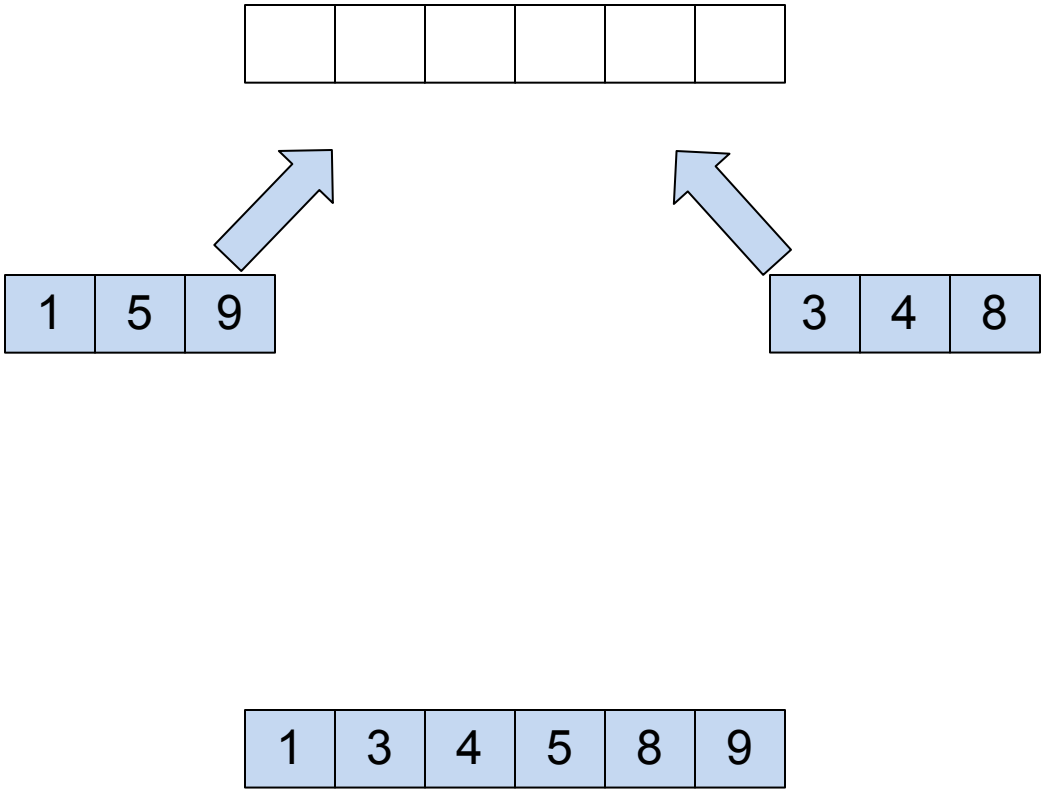
made by petermouse & DA

Merge Sort



made by petermouse & DA

Merge Sort



made by petermouse & DA

Merge Sort

- 1.　先將a 分割成兩半a[0]~a[n/2]和a[n/2+1]~a[n-1]，然後分別對它們做Merge Sort，如此遞迴下去。
- 2.　如果遞迴到陣列元素只有一個的時候，則要return，不必再分割。
- 3.　兩邊分別都做完Merge Sort 之後，便對兩組資料進行「合併」動作。

Merge Sort

- Divide 分

```
void Mergesort(int list[], int low, int high) {
    if (high > low) {
        Mergesort(list, low, (low + high) / 2);
        Mergesort(list, (low + high) / 2 + 1, high);
        Merge(list, low, high);
    }
}
```

Merge Sort

- Conquer 合

```
void Merge(int list [], int low, int high) {
    int combined[MAX_SIZE], i, j;
    int k = -1, mid = (low + high) / 2;
    for (i = low, j = mid + 1; i <= mid || j <= high;) {
        if (i > mid) combined[++k] = list[j++];
        else if (j > high) combined[++k] = list[i++];
        else if (list[i] >= list[j]) combined[++k] = list[j++];
        else combined[++k] = list[i]++;
    }
    k = 0;
    for (i = low; i <= high; i++)
        list[i] = combined[k++];
}
```

Example 2

Uva 10810 – Ultra-QuickSort

In this problem, you have to analyze a particular sorting algorithm. The algorithm processes a sequence of *n* distinct integers by swapping two adjacent sequence elements until the sequence is sorted in ascending order. For the input sequence9 1 0 5 4 ,Ultra-QuickSort produces the output0 1 4 5 9 .Your task is to determine how many swap operations Ultra-QuickSort needs to perform in order to sort a given input sequence.The input contains several test cases. Every test case begins with a line that contains a single integer $n < 500,000$ -- the length of the input sequence. Each of the the following *n* lines contains a single integer $0 \le a[i] \le 999,999,999$, the *i*-th input sequence element. Input is terminated by a sequence of length $n = 0$. This sequence must not be processed.

For every input sequence, your program prints a single line containing an integer number *op*, the minimum number of swap operations necessary to sort the given input sequence.

Example 2

- **Sample Input**

  5
  9
  1
  0
  5
  4
  3
  1
  2
  3
  0

  Output for Sample Input

  6
  0

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- **補充**
  - Quick sort
  - qsort()

STL

- STL = **S**tandard **T**emplate **L**ibrary **標準模板庫**
- 包含了 container(容器), iterator(迭代器), algorithm

```
#include <algorithm>
```

- <algorithm>
  - sort()
  - stable_sort()
  - ……
  - http://www.cplusplus.com/reference/algorithm/

STL

- sort()
  - 內部實作以 Quick sort 為原型
  - 非穩定排序
  - 非必要的情況下，我們都不會自己寫而使用 STL 的sort。

- sort (ary, ary + n)
  - ascending order
  - 排序 ary[0] 至 ary [n - 1]，共 n 筆資料

- sort (ary, ary + n, cmp)
  - cmp : comparision function

STL

- comparision function 的使用時機
  - 不按照預設的 ascending order 做排序

```
bool cmp(int a, int b) {
    return a > b; // 改以 descending order 排序
}
```

  - 為 custom data type 做排序

```
struct Point {
    int x, y;
} p[10000];

bool cmp(struct Point p1, struct Point p2)
{
    return p1.x < p2.x;
    // 以 Point 的 x 值做 ascending order排序
}
```

STL

- 使用 operator overloading 為 custom data type 做排序

```
struct Point {
    int x;
    int y;
    bool operator <(const struct Point &p) const {
        return x < p.x;
    }
} p[10000];
```

STL

- Example (built-in type)

```cpp
#include <algorithm> // 使用 sort()
using namespace std;
// 用比較函式來自訂排序
bool cmp(int a, int b) {
    return a > b; // 改以降冪排序
}
int main() {
    int a[8] = {8, 6, 2, 7, 3, 1, 4, 5};
    sort(a, a + 4);       // (2 6 7 8) 3 1 4 5
    sort(a + 4, a + 8);   // 2 6 7 8 (1 3 4 5)
    sort(a, a + 8);       // (1 2 3 4 5 6 7 8)
    sort(a, a + 8, cmp); // (8 7 6 5 4 3 2 1)
    return 0;
}
```

STL

- Example (custom data type)

```
struct Point {
    int x, y;
    bool operator <(const struct Point &p2) const {
        return x < p2.x;
    }
} p[10000];

int main() {
    int n, x, y;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &x, &y);
        p[i].x = x; p[i].y = y;
    }
    sort(p, p + n);
    for (int i = 0; i < n; i++)
        printf("x= %d y= %d\n", p[i].x, p[i].y);
    return 0;
}
```

STL

- stable_sort()
  - 與 sort() 相同用法
  - 穩定排序

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- **補充**
  - Quick sort
  - qsort()

補充:Quick Sort

- 一樣利用分治法（Divide and Conquer）
- 將資料分成兩組分別再做排序。
- 平均時間複雜度$O(n \lg n)$
- 為不穩定排序。

補充:Quick Sort

- 1. 挑選資料中的一個元素做為「基準」
  （以下皆以序列末端元素作為基準）
- 2. 然後將比它小的放在前面部分，比它大的放後面部分
- 3. 將「基準」與比它大的序列的第一個元素交換，則「基準」在序列中的位置就確定了。
- 4. 然後對前後兩個序列分別再遞迴做Quick Sort

補充:Quick Sort

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
8	6	2	7	3	1	4	5

8	6	2	7	3	1	4	***5***
2	6	8	7	3	1	4	***5***
2	3	8	7	6	1	4	***5***
2	3	1	7	6	8	4	***5***
2	3	1	4	6	8	7	***5***
2	3	1	4	***5***	8	7	6

紅色斜體：基準

灰底：前面部分序列

底線：當前被交換的資料

補充:Quick Sort

2	3	1	*4*	5	基準為 4		
2	3	*1*	4	4,4 交換，基準為 1			
1	3	*2*		1,2 交換，基準為 2			
	2	*3*		2,3 交換，基準為 3			
		3		遞迴到底，回傳			
1	2	3	4	5	8	7	6

前面部分排序完畢，換後面部分遞迴

基準為 6				5	8	7	*6*
6,8 交換，基準為 8					6	7	*8*
8,8 交換，基準為 7						*7*	8
遞迴到底，回傳						7	
1	2	3	4	5	6	7	8

紅色斜體：基準

粗框：當前被排序好的資料

e by petermouse & DA

Outline

- **Time Complexity**
- **Sorting**
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - STL
- **補充**
  - Quick sort
  - qsort()

補充:qsort()

- qsort
  - C library
  - 非穩定排序

```
#include <stdlib.h> // C   使用 <stdlib.h>
#include <cstdlib>  // C++ 使用 <cstdlib>
```

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

- qsort()
  - base: 指標起始位置
  - size: 元素資料寬度
  - num: 幾個元素
  - compar: 比較函數

補充:qsort()

- 比較函式的參數型態是 (const void *)
- a 比 b 前 -> return 負數
- a 與 b 相等 -> return 0
- a 比 b 後 -> return 正數

```
int cmp (const void * a, const void * b)
{
  if ( *(int*)a <  *(int*)b ) return -1;
  if ( *(int*)a == *(int*)b ) return 0;
  if ( *(int*)a >  *(int*)b ) return 1;
}

int ary[1000];
qsort(ary, n, sizeof(int), cmp);
```

Thank you for your listening!

made by petermouse & DA

Problem List

- Uva (7)
  10327, 10810, 10107, 10026, 11727, 10420

- POJ (9)
  3067, 1002, 1007, 2231, 2371, 2388, 1318, 1971, 3663