NCKU Programming Contest Training Course
Time Complexity & Sorting
2017/02/16

Jingfei Yang

e84016184@mail.ncku.edu.tw

http://myweb.ncku.edu.tw/~e84016184/sorting.pdf

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

Time Complexity

Time Complexity

- How to evaluate the execution time?

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

Time Complexity

- Technical Analysis!!

Time Complexity

- k nested loops with n iterations each:
  - $O(n^k)$

Time Complexity

- k nested loops with n iterations each:
  - $O(n^k)$

- b recursive calls per level with maximum L levels:
  - $O(b^L)$

Time Complexity

- k nested loops with n iterations each:
  - $O(n^k)$
- b recursive calls per level with maximum L levels:
  - $O(b^L)$
- Process 2D n*m matrix with k op. each cell:
  - O(**n** x **m** x **k**)

Time Complexity

- Examples
- for(i=0;i<n;i++)
  if(.....) ....
- O(n)

Time Complexity

- Examples

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

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

Time Complexity

- 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) =
  - $O(n^2)$ =
  - O(nlgn) =

- Given Input Size n = 1,000,000:
  - O(n) =
  - $O(n^2)$ =
  - O(nlgn) =

Time Complexity

- Given Input Size n = 1,000:
  - O(n) = O(1,000) **OK**
  - $O(n^2)$ = O(1,000,000) **OK**
  - O(nlgn) ≒ O(9965) **OK**

- Given Input Size n = 1,000,000:
  - O(n) = O(1,000,000) **OK**
  - $O(n^2)$ = O(1,000,000,000,000) **Not good. Why…?**
  - O(nlgn) ≒ O(9,965,784) **OK**

Time Complexity

n	Worst AC Algorithm
≦[10..11]	$O(n!)$, $O(n^6)$
≦[15..18]	$O(2^n \times n^2)$
≦[18..22]	$O(2^n \times n)$
≦100	$O(n^4)$
≦400	$O(n^3)$
≦2K	$O(n^2 \log_2 n)$
≦10K	$O(n^2)$
≦1M	$O(n \log_2 n)$
≦100M	$O(n)$, $O(\log_2 n)$, $O(1)$

*A typical year 2013 CPU can process 100M operations in few seconds.

*Referenced from Competitive Programming, 3ed.

Time Complexity

- But!!
- The actual running time depends on your actual number of operations and CPU power.
- For safety: 10^6 - 10^7 => ≦ 3 seconds
  - Modern computers

Sorting

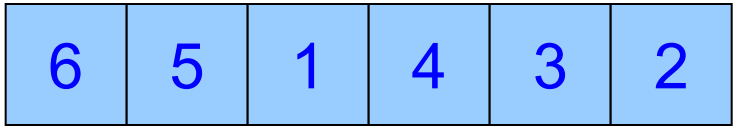
What is sorting?

- Order the sequence by some rules

- Ex: ascending order
  - 6 5 1 4 3 2
  - 1 2 3 4 5 6

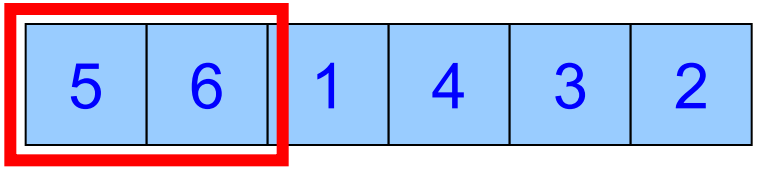
Bubble Sort

- Original



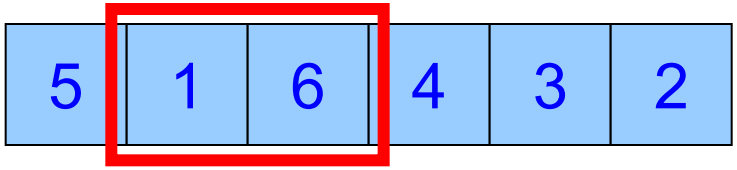
Bubble Sort

- Swap



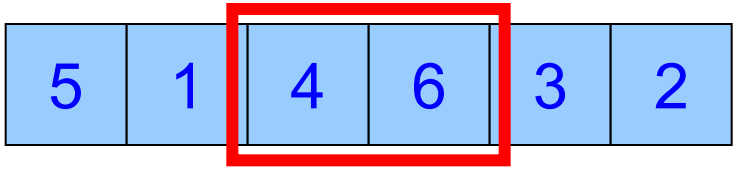
Bubble Sort

- Swap



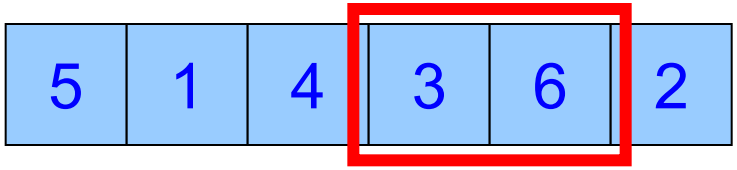
Bubble Sort

- Swap



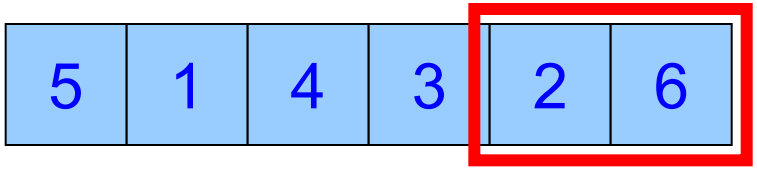
Bubble Sort

- Swap



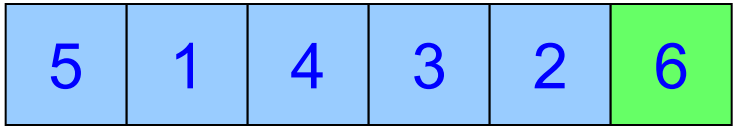
Bubble Sort

- Swap



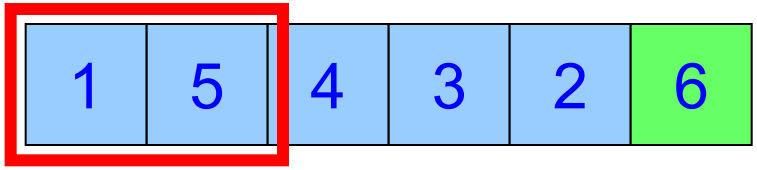
Bubble Sort

- End of First Iteration



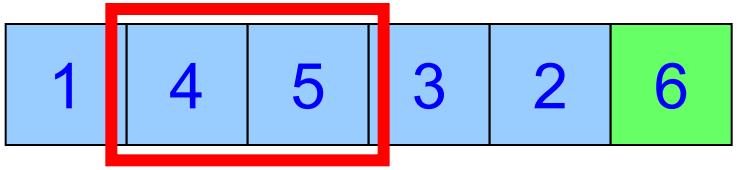
Bubble Sort

- Swap



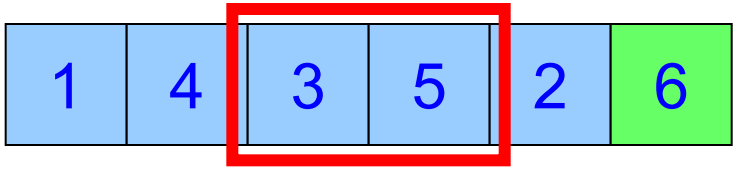
Bubble Sort

- Swap



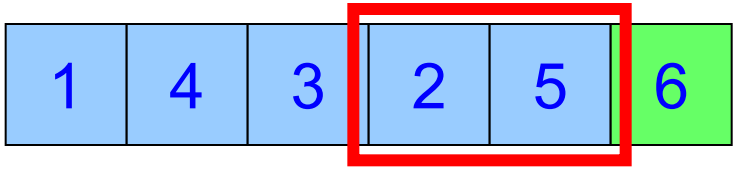
Bubble Sort

- Swap



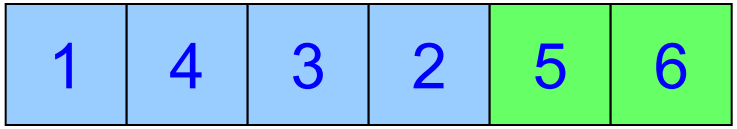
Bubble Sort

- Swap



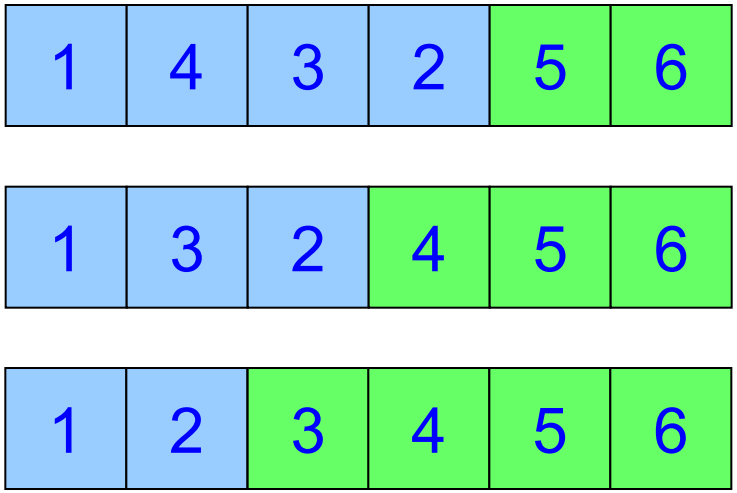
Bubble Sort

- End of Second Iteration



Bubble Sort

- And so on….



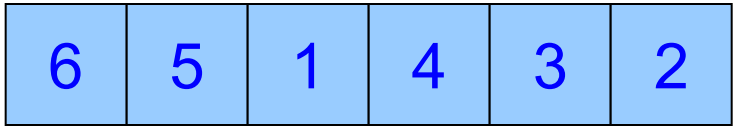
Bubble Sort

- Code

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

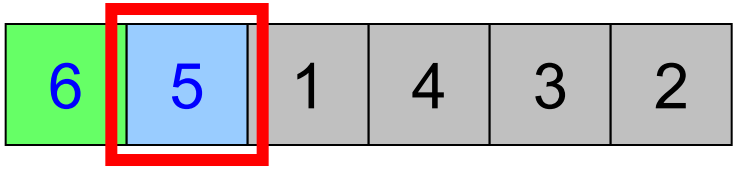
Insertion Sort

- Original



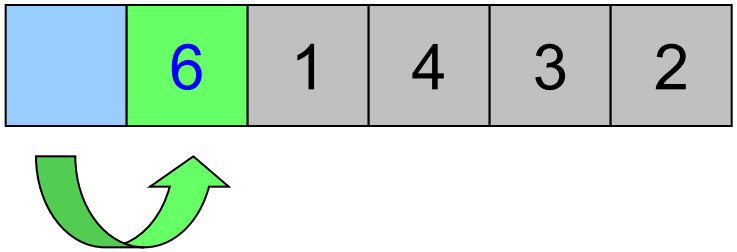
Insertion Sort

- Choose one to insert



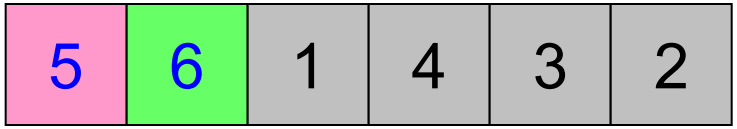
Insertion Sort

- If value larger than chosen, then shift



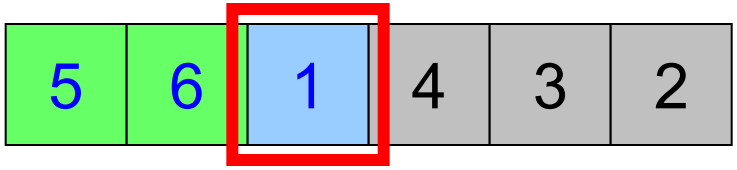
Insertion Sort

- Insert



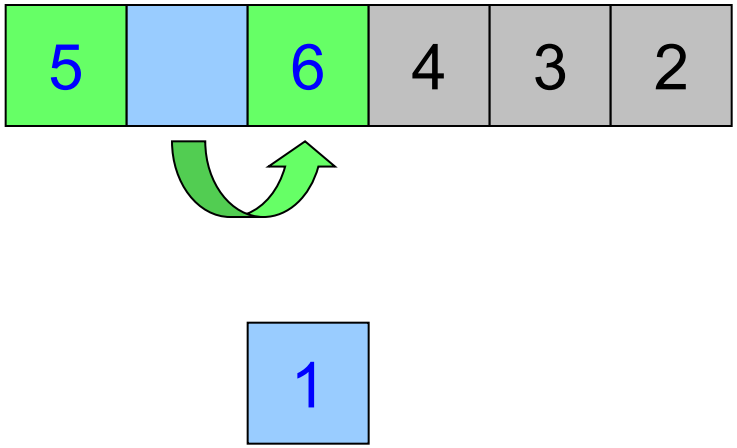
Insertion Sort

- Choose one to insert



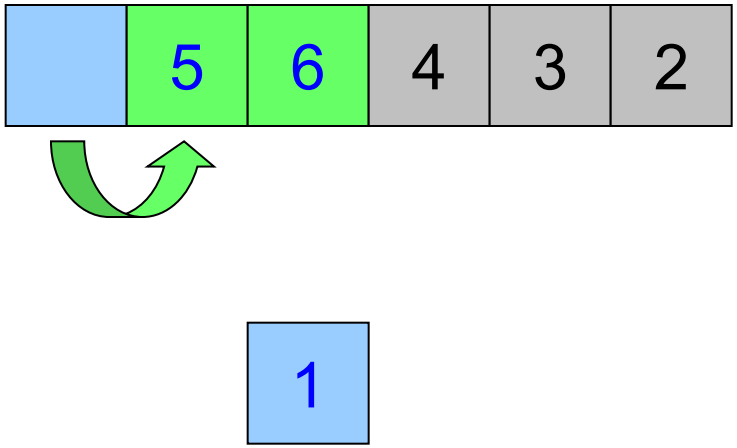
Insertion Sort

- Shift



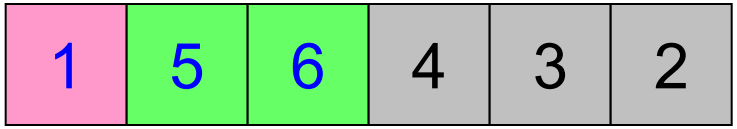
Insertion Sort

- Shift



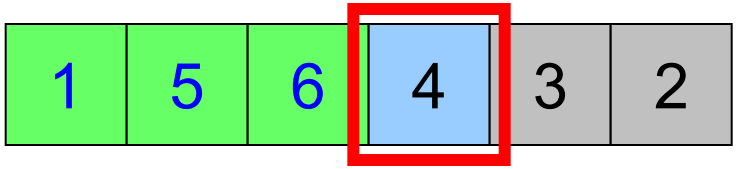
Insertion Sort

- Insert



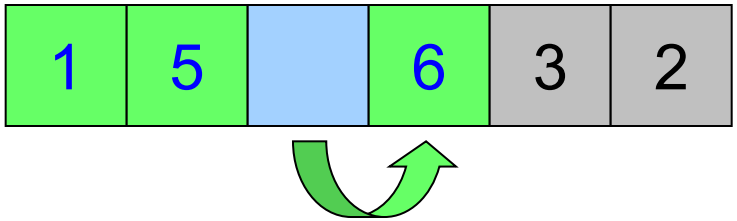
Insertion Sort

- Choose one to insert



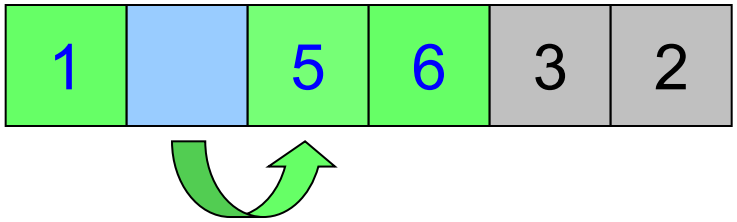
Insertion Sort

- Shift



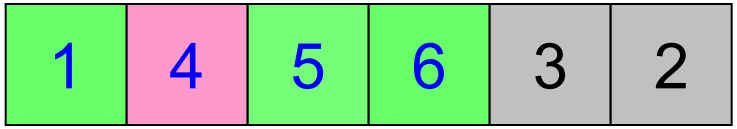
Insertion Sort

- Shift



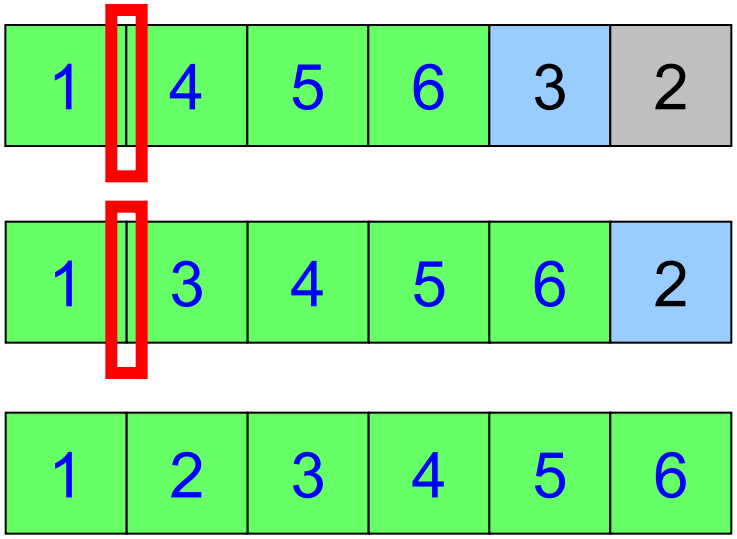
Insertion Sort

- Insert



Insertion Sort

- And so on…



Done.

Insertion Sort

- Code

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

Example 1

UVa 10327 - Flip Sort

Sorting in computer science is an important part. Almost every problem can be solved effeciently if sorted data are found. There are some excellent sorting algorithm which has already acheived 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.

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

逆序數

- 逆序對
  - ary[i] > ary[j] for some i < j

- 逆序數
  - Sequence中所有逆序對的個數

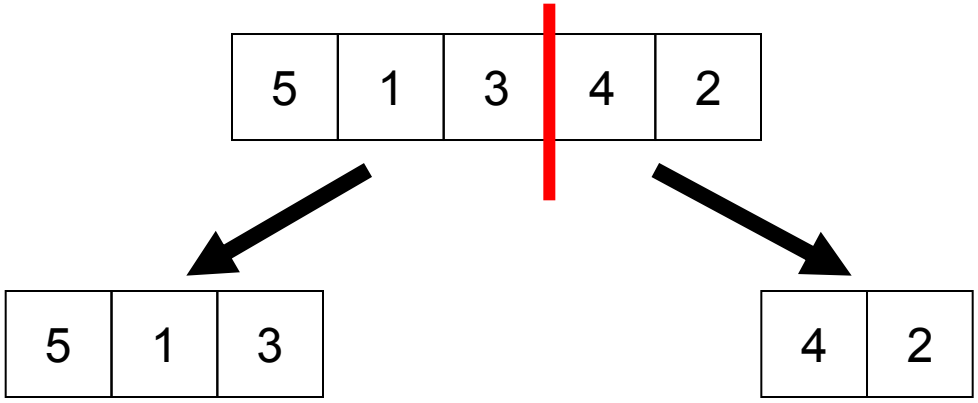
Merge Sort

- Divide & Conquer
- O(nlgn)

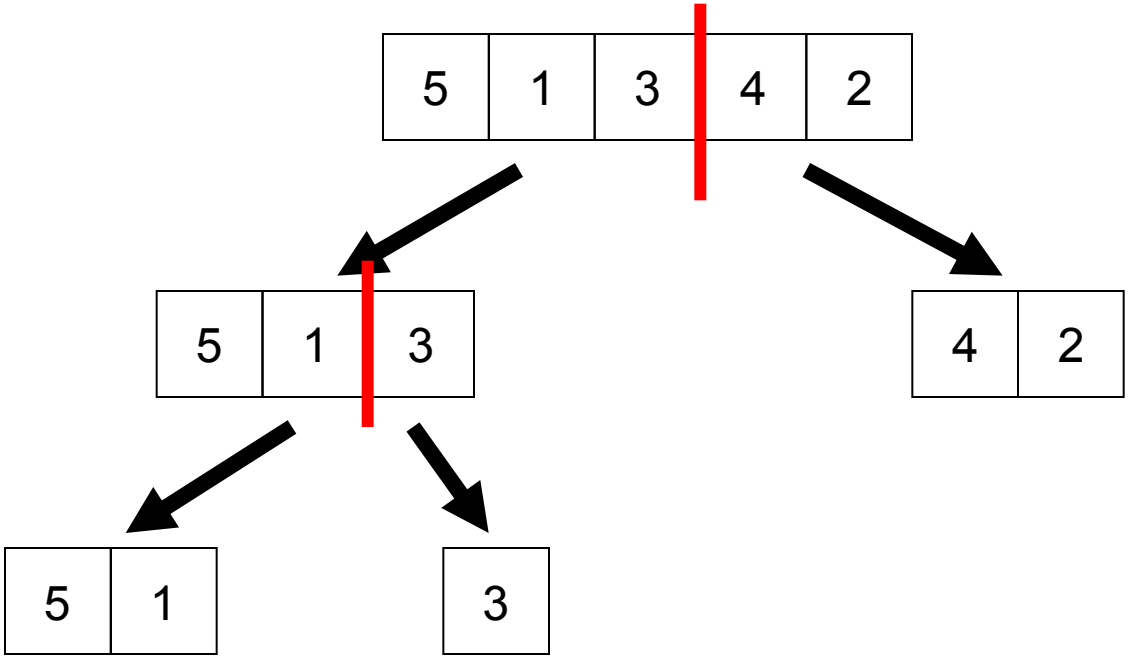
Merge Sort

5	1	3	4	2

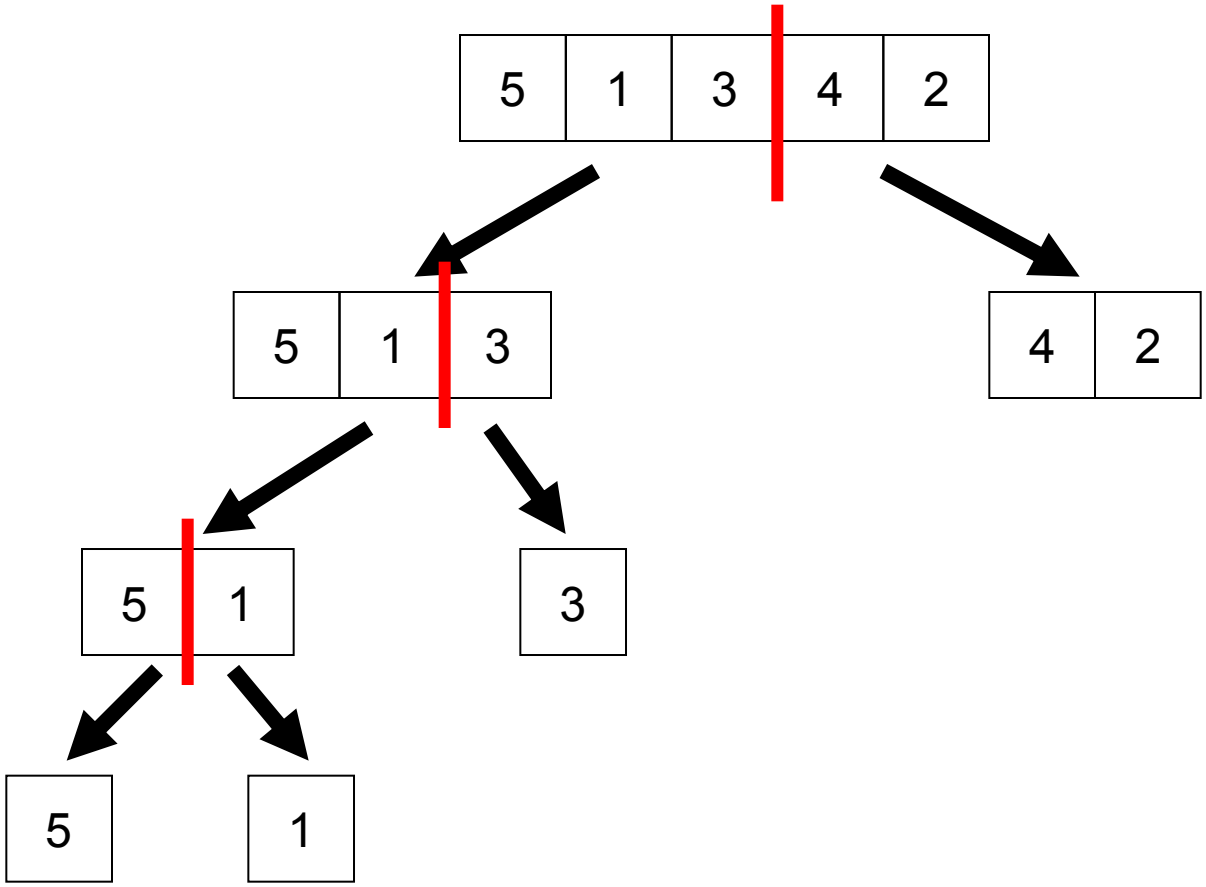
Merge Sort



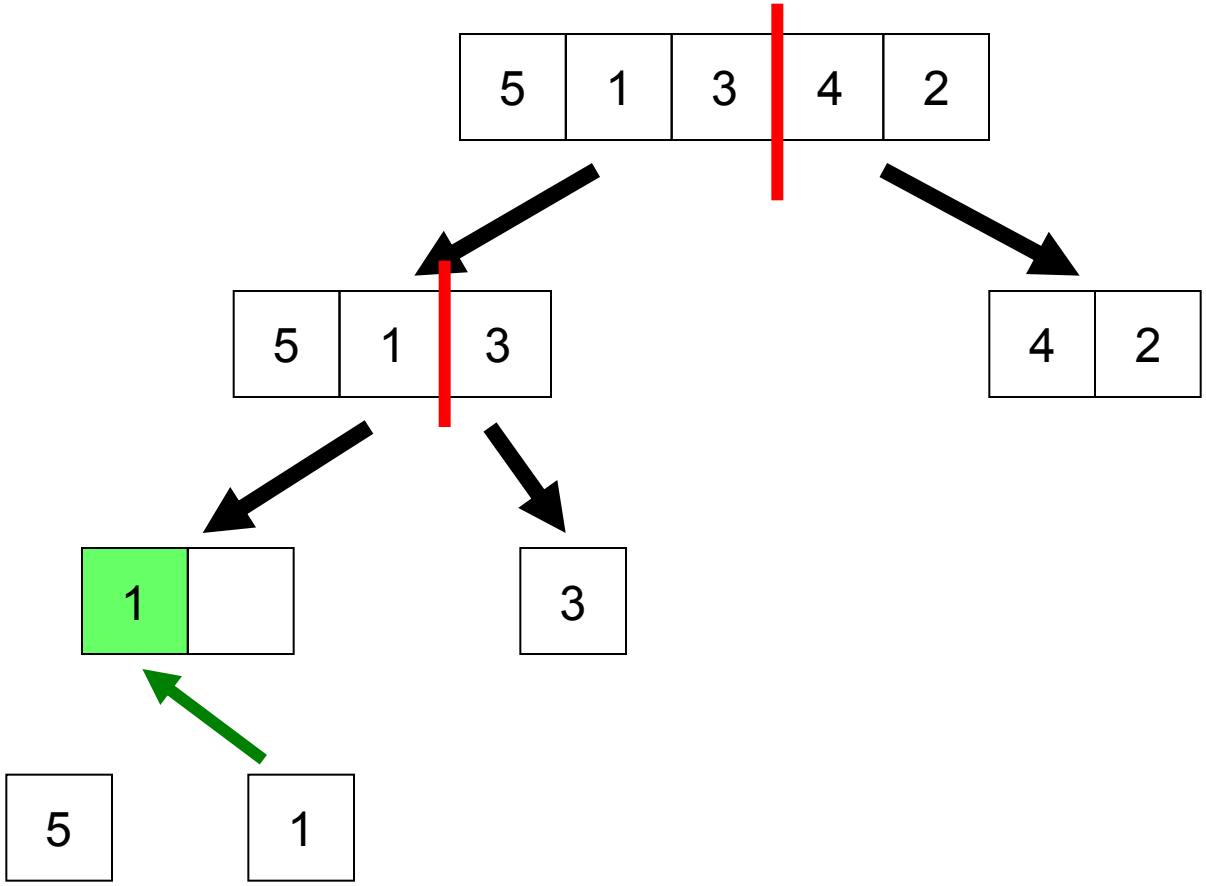
Merge Sort



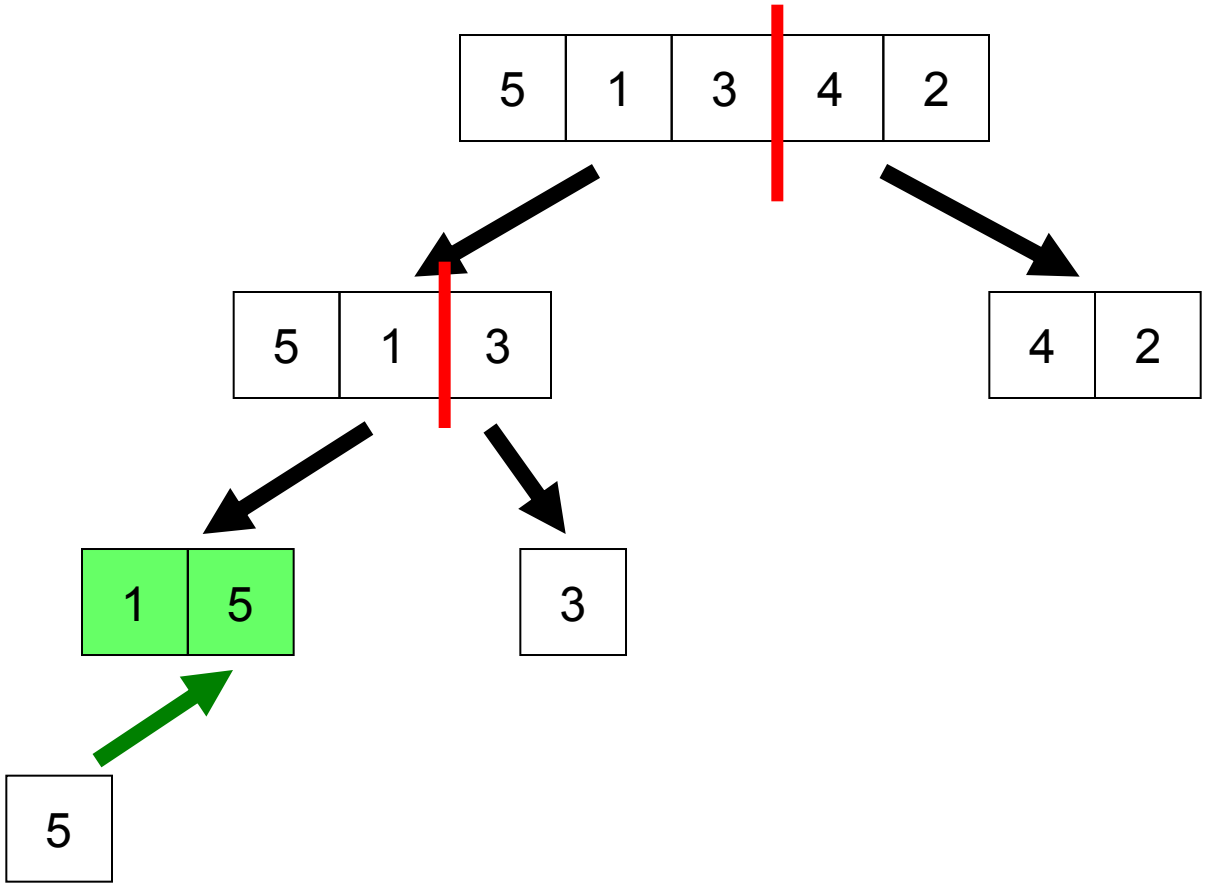
Merge Sort



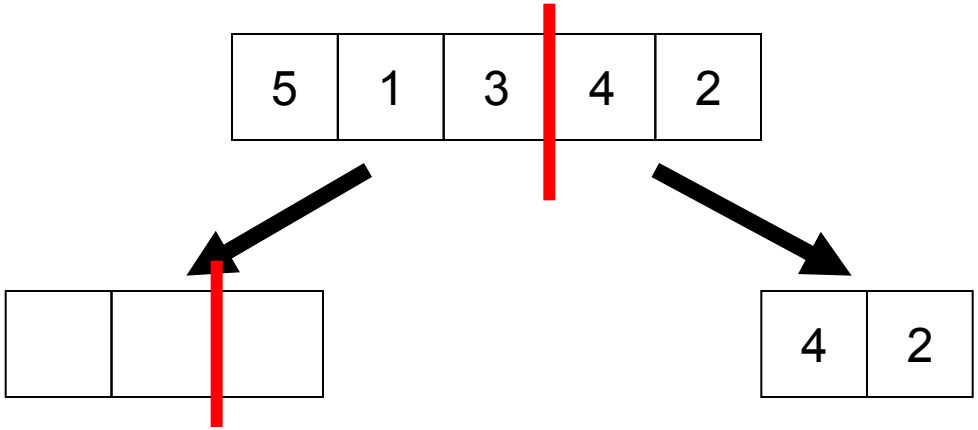
Merge Sort



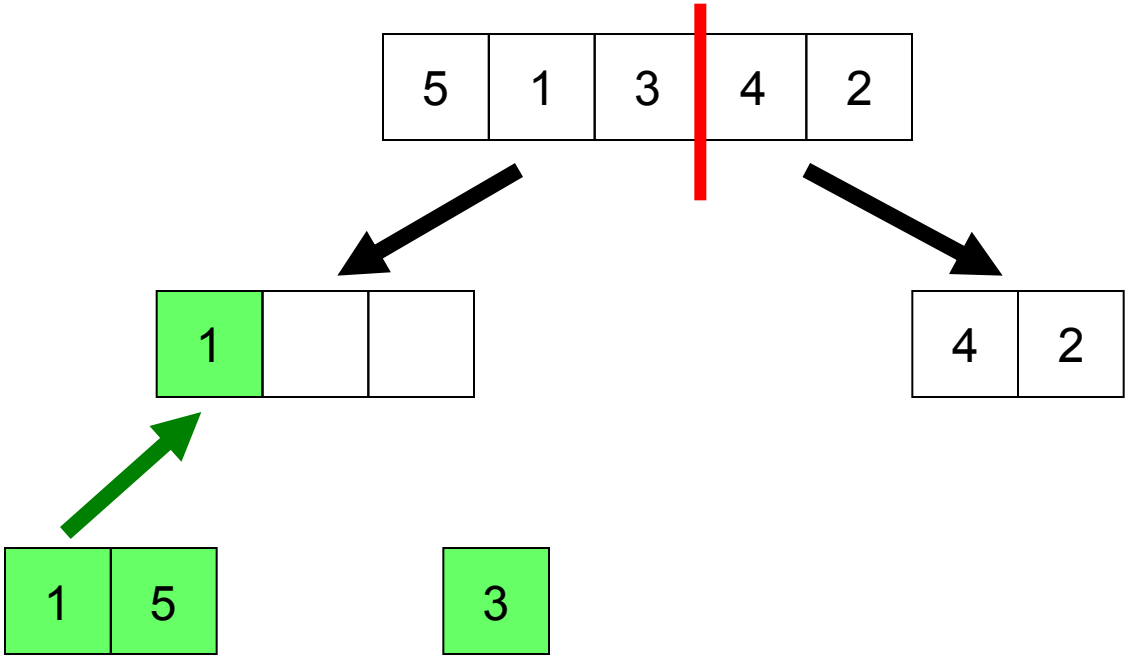
Merge Sort



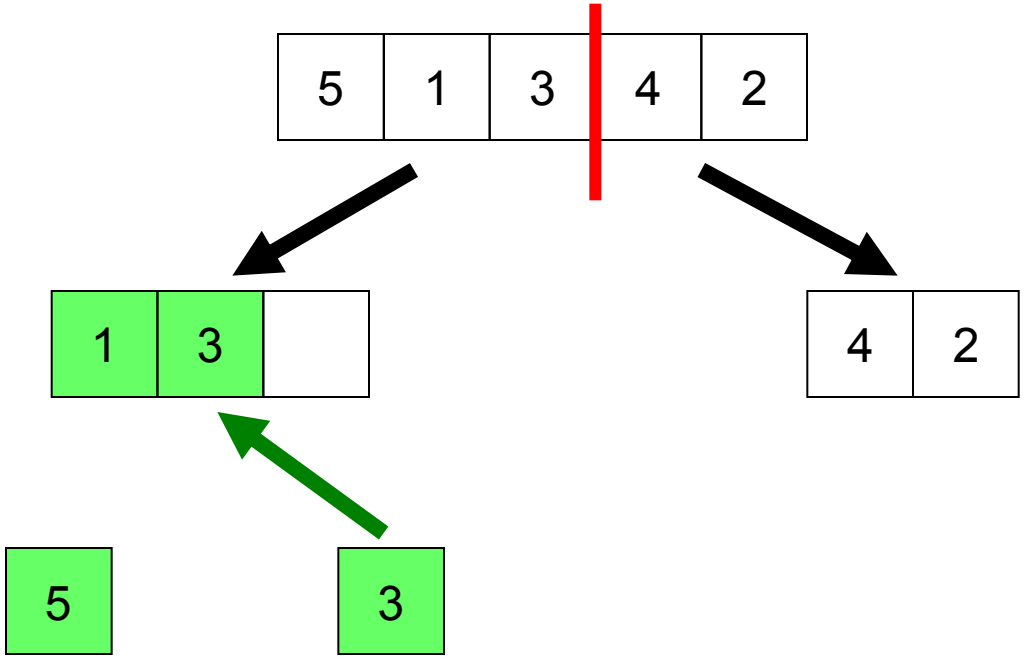
Merge Sort



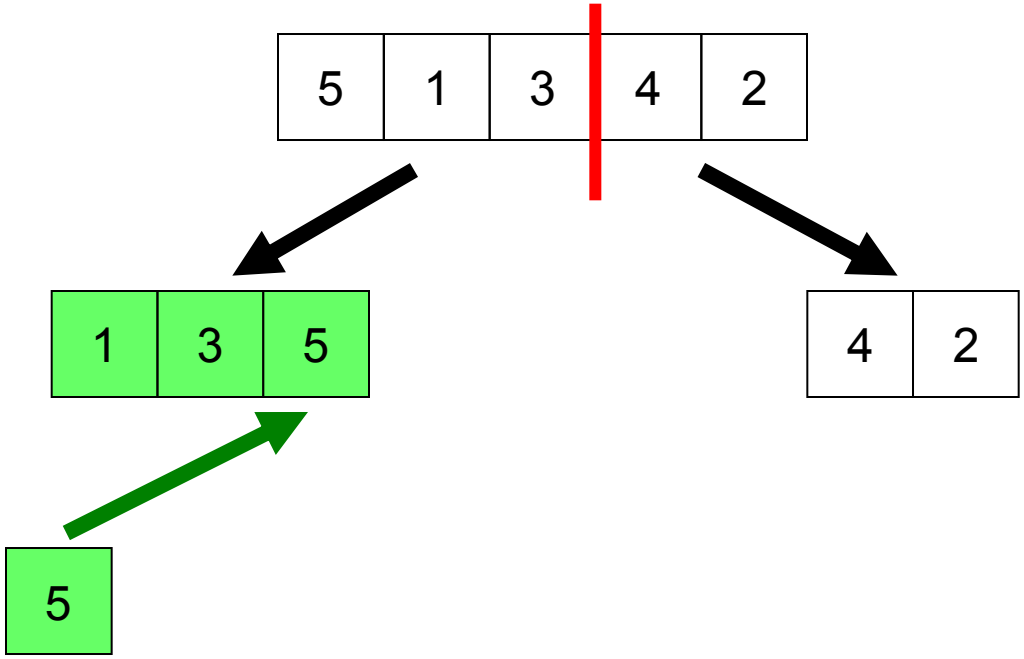
Merge Sort



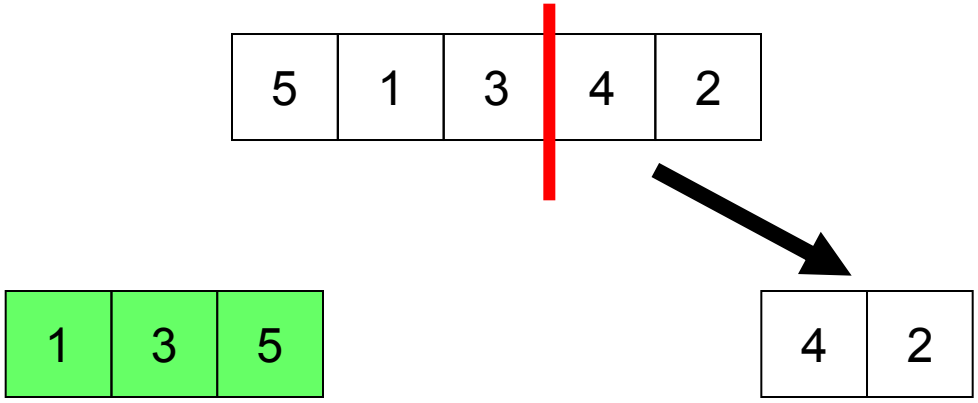
Merge Sort



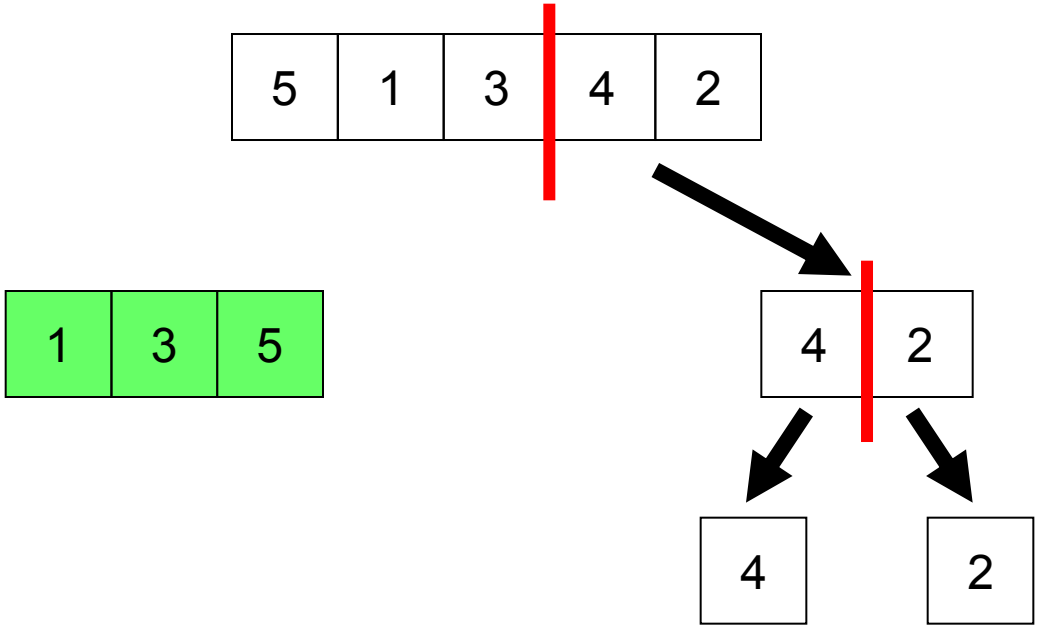
Merge Sort



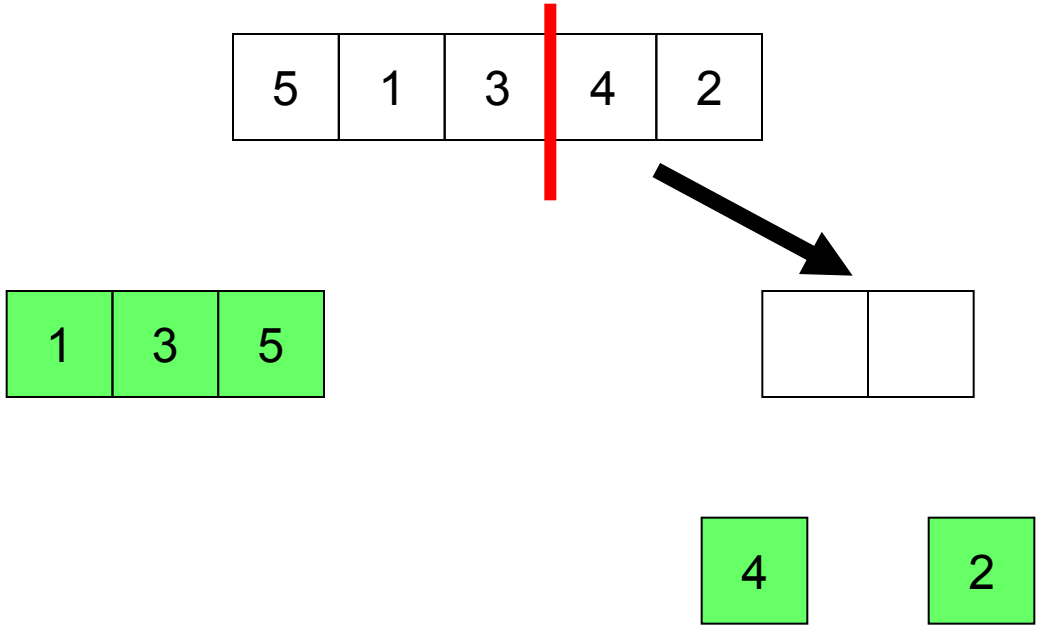
Merge Sort



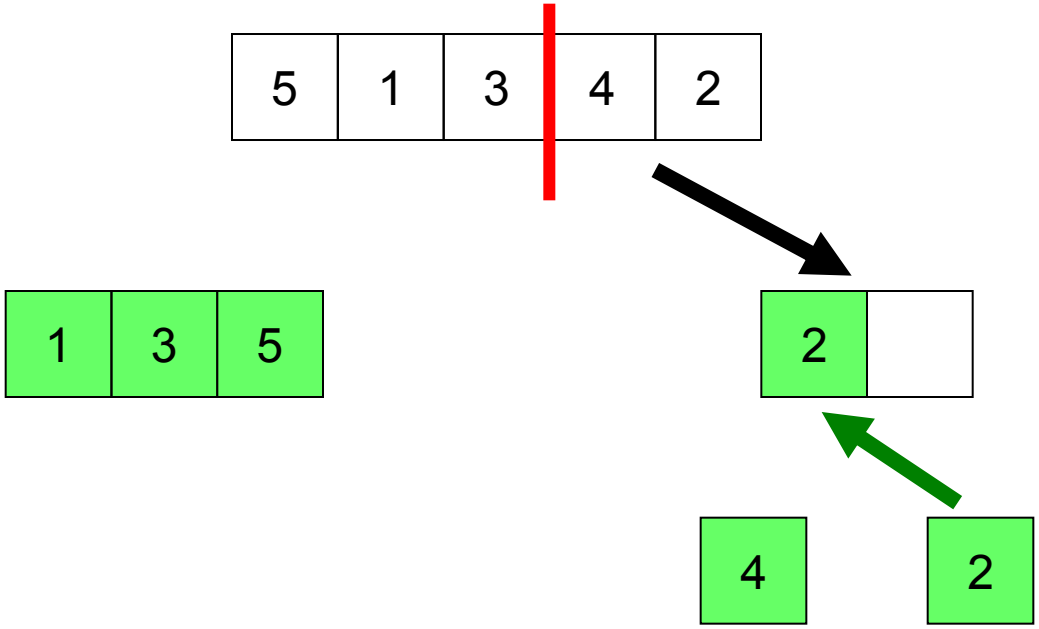
Merge Sort



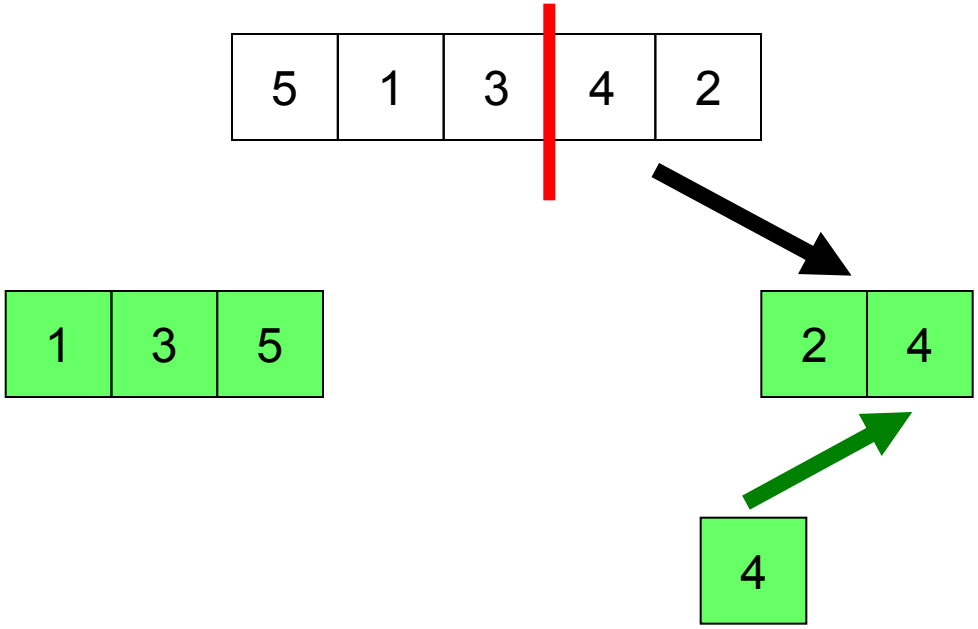
Merge Sort



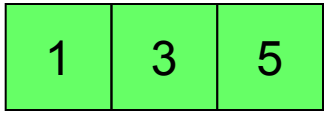
Merge Sort

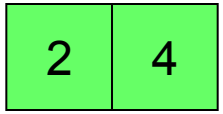


Merge Sort

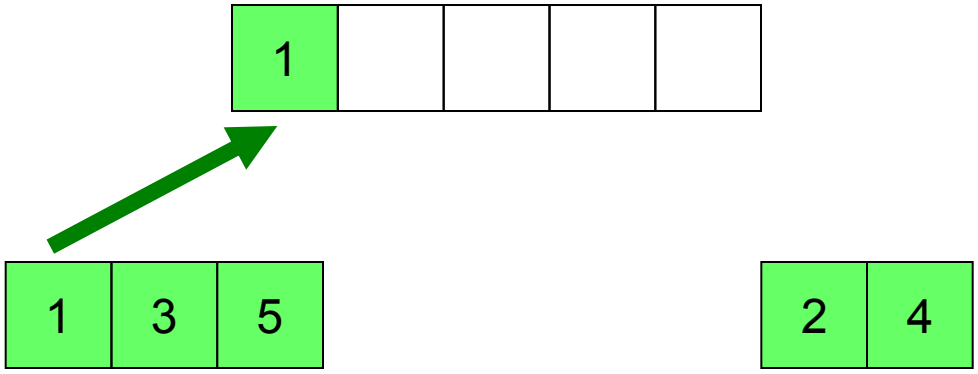


Merge Sort

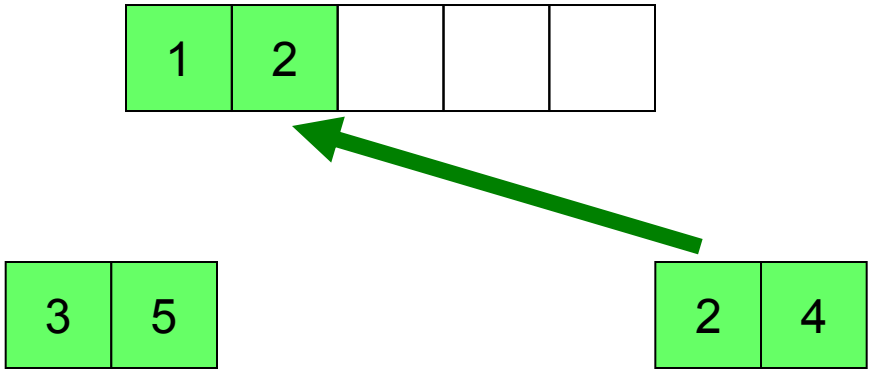




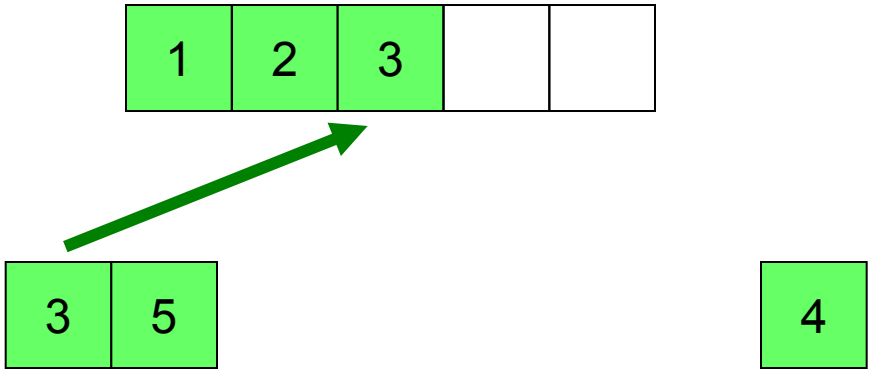
Merge Sort



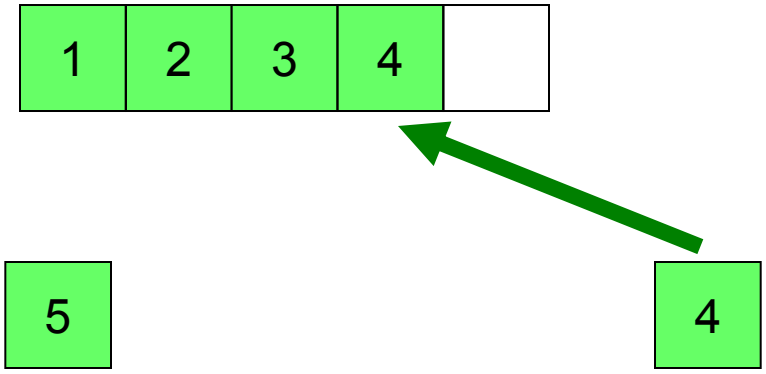
Merge Sort



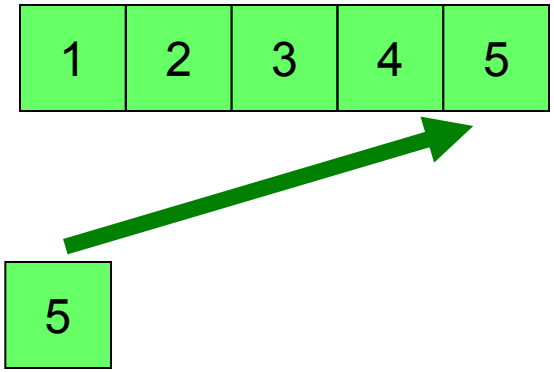
Merge Sort



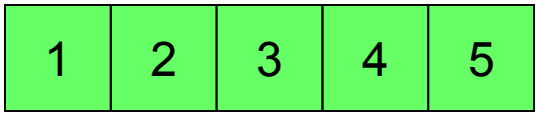
Merge Sort



Merge Sort



Merge Sort



Done!!

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

STL - sort

- #include<algorithm> // C++
- sort(ary, ary + n);
  - Ascending order
- sort(ary, ary + n, cmp);
  - Comparison Function ‘cmp’

Example (builtin type)

```
int main() {
  int n, N[ 10010 ];
  while ( scanf( “%d”, &n ) != EOF ) {
    int x;
    for ( int i = 0; i < n; ++i ) {
      scanf( “%d”, &x );
      N[ i ] = x;
    }
    sort( N, N + n );
  }
  return 0;
}
```

Customized Data Type

Function prototype for operator <:

```
bool operator< ( const type_name &p ) const;
```

return value	means
TRUE	this < p
FALSE	this >= p

Example (custom type)

```
struct T {
  int x, y;
  bool operator< ( const struct  T &p ) const {
    return x == p.x ? y < p.y : x < p.x;
  }
} pt[ 10010 ];

int main() {
  int n;
  while ( scanf( “%d”, &n ) != EOF ) {
    int x, y;
    for ( int i = 0; i < n; ++i ) {
      scanf( “%d %d”, &x, &y );
      pt[ i ].x = x, pt[ i ].y = y;
    }
    sort( pt, pt + n );
  }
  return 0;
}
```

Customized Data Type

Function prototyp:

```
bool function_name ( type_name p1, type_name p2 );
```

return value	means
TRUE	p1 < p2
FALSE	p1 >= p2

Example (descending)

```
bool descending( int p1, int p2 ) {
  return p1 >= p2;
}

int main() {
  int n, N[ 10010 ];
  while ( scanf( “%d”, &n ) != EOF ) {
    int x;
    for ( int i = 0; i < n; ++i ) {
      scanf( “%d”, &x );
      N[ i ] = x;
    }
    sort( N, N + n, descending );
  }
  return 0;
}
```

Example 3

POJ 3664 - Election Time

The cows are having their first election after overthrowing the tyrannical Farmer John, and Bessie is one of N cows ($1 \le N \le 50{,}000$) running for President. Before the election actually happens, however, Bessie wants to determine who has the best chance of winning.

The election consists of two rounds. In the first round, the K cows ($1 \le K \le N$) cows with the most votes advance to the second round. In the second round, the cow with the most votes becomes President.

Given that cow i expects to get Ai votes ($1 \le Ai \le 1{,}000{,}000{,}000$) in the first round and Bi votes ($1 \le Bi \le 1{,}000{,}000{,}000$) in the second round (if he or she makes it), determine which cow is expected to win the election. Happily for you, no vote count appears twice in the Ai list; likewise, no vote count appears twice in the Bi list.

Example 3

Input

* Line 1: Two space-separated integers: *N* and *K*
* Lines 2..*N*+1: Line i+1 contains two space-separated integers: *Ai* and *Bi*

Output

* Line 1: The index of the cow that is expected to win the election.

Sample Input

```
5 3
3 10
9 2
5 6
8 4
6 5
```

Sample Output

```
5
```

Thank you for your listening!

made by Jingfei & kevinx6000

Practice

- Uva (6)
  - 10327, 10810, 10107, 10026, 10420
- POJ (11)
  - 3664, 3067, 3262, 1002, 1007, 2231, 2371, 2388, 1318, 1971, 3663

made by Jingfei & kevinx6000