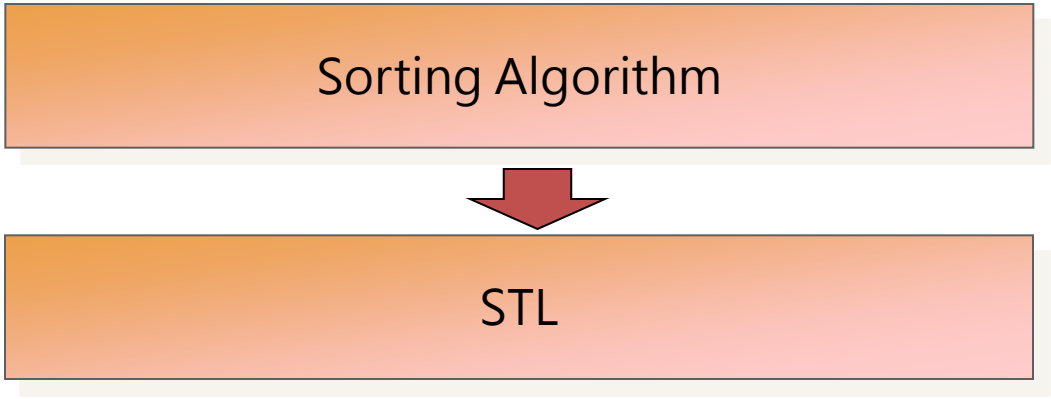
NCKU Programming Contest Training Course
Course 2
2012/12/26

Pin-Chieh Huang (free999)

Pinchieh.huang@gmail.com

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

Outline



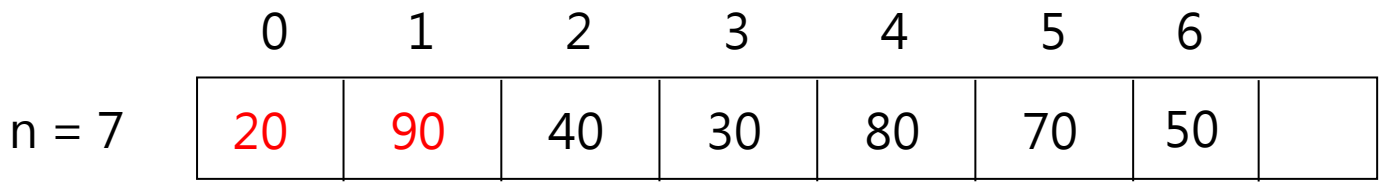
Sort - Bubble Sort

- Bubble sort is the basic sort algorithm
  - Scan + Swap

```
void bubble_sort(int in[], int n)
{
    int k, j;
    for(k=n-1; k>=1;k--)
       for(j=0; j<k; j++)
          if(in[j]>in[j+1])
              swap(in[j], in[j+1]);
}
```

  - $O(N^2)$ time complexity
  - Minimum Number of Adjacent Swapping to Sort a Sequence

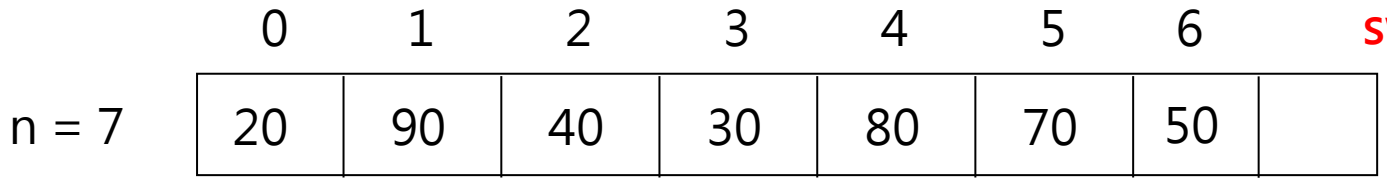
Sort - Bubble Sort



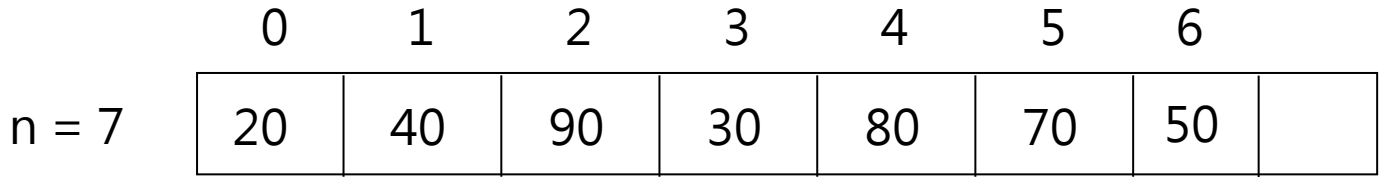
k = n-1 = 6

j = 0, a[0] = 20 < 90 = a[1]

```
for(j=0; j<k; j++)
    if(in[j]>in[j+1])
        swap(in[j], in[j+1]);
```

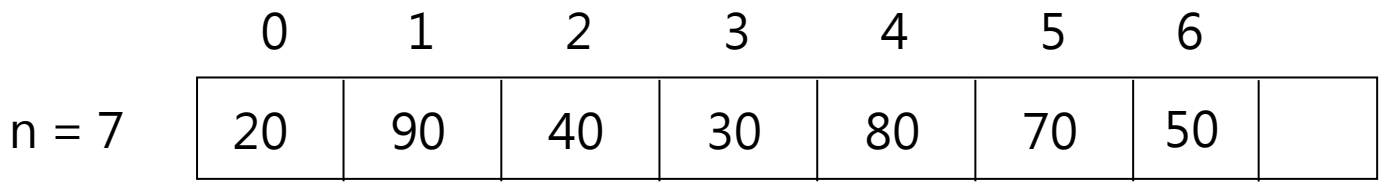


j = 1, a[1] = 90 > 40 = a[2]



made by electron & free999

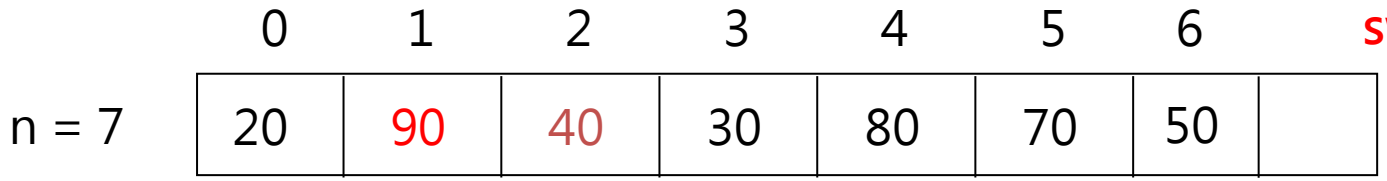
Sort - Bubble Sort



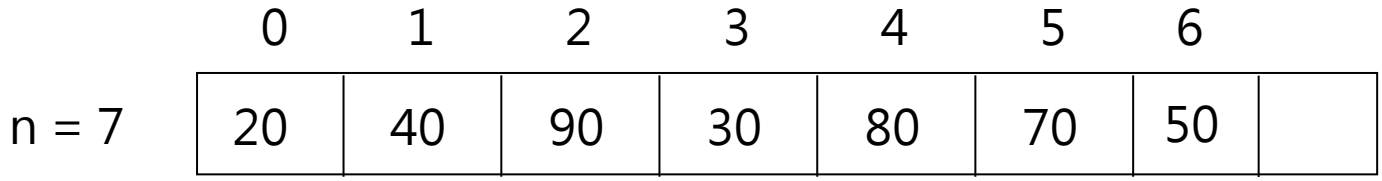
k = n-1 = 6

j = 0, a[0] = 20 < 90 = a[1]

```
for(j=0; j<k; j++)
    if(in[j]>in[j+1])
        swap(in[j], in[j+1]);
```

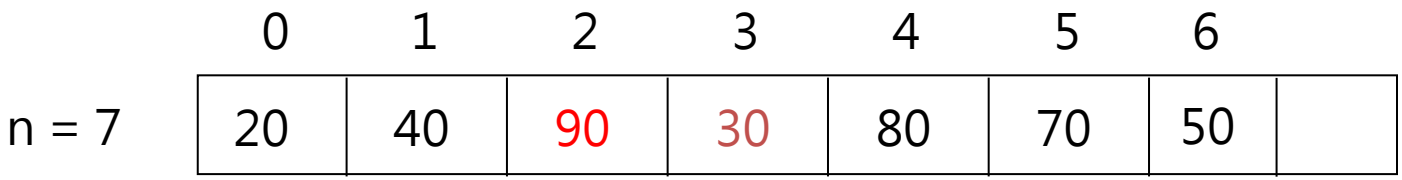


j = 1, a[1] = 90 > 40 = a[2]



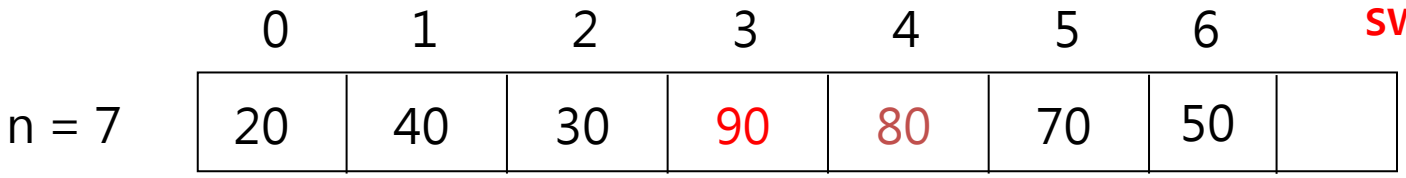
made by electron & free999

Sort - Bubble Sort

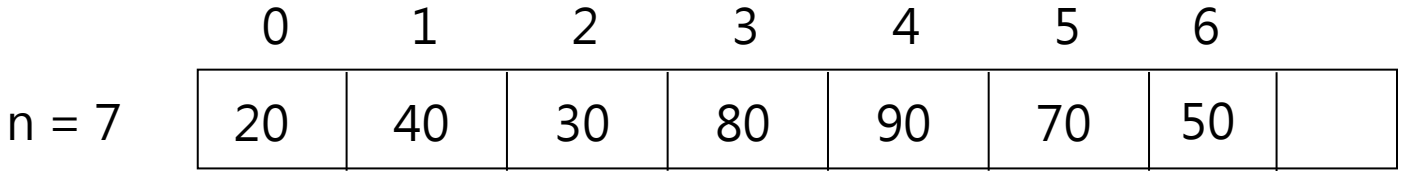


j = 2, a[2] = 90 > 30 = a[3]

```
for(j=0; j<k; j++)
    if(in[j]>in[j+1])
        swap(in[j], in[j+1]);
```

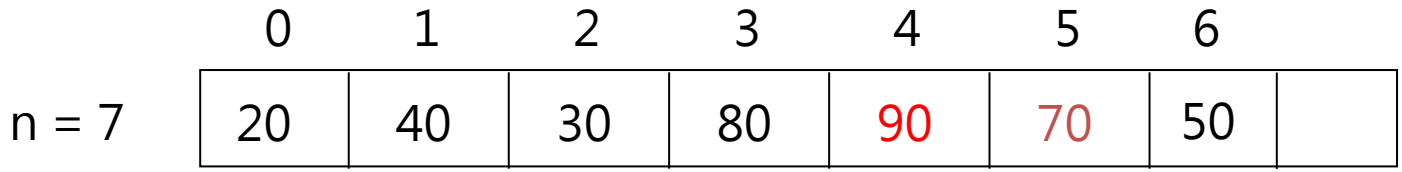


j = 3, a[3] = 90 > 80 = a[4]



made by electron & free999

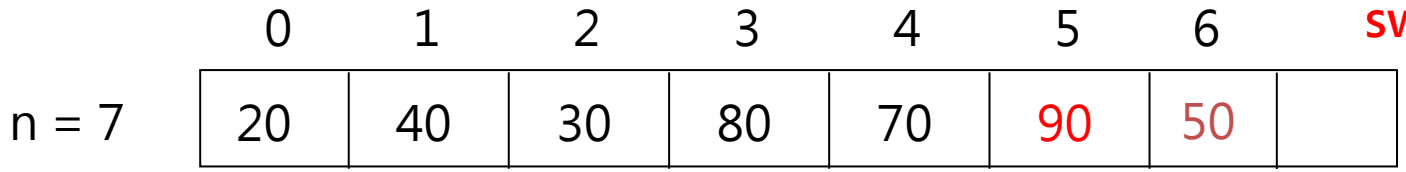
Sort - Bubble Sort



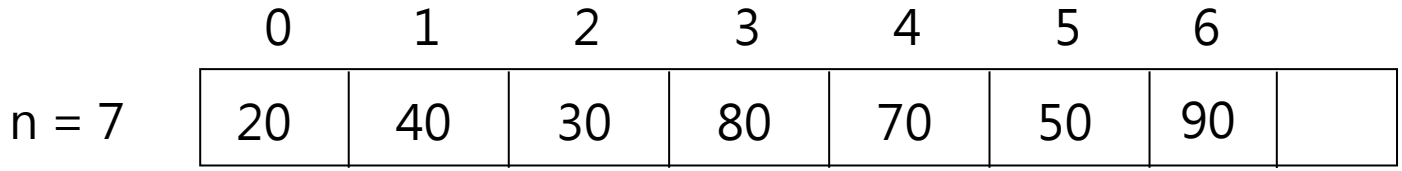
k = n-1 = 6

j = 4, a[4] = 90 > 70 = a[5]

```
for(j=0; j<k; j++)
    if(in[j]>in[j+1])
        swap(in[j], in[j+1]);
```



j = 5, a[5] = 90 > 50 = a[6]



made by electron & free999

Sort - Bubble Sort

for(k=n-1; k>=1;k--)

		0	1	2	3	4	5	6	
	k = 6	20	90	40	30	80	70	50	
		0	1	2	3	4	5	6	
pass 1:	k = 5	20	40	30	80	70	50	90	
		0	1	2	3	4	5	6	
pass 2:	k = 4	20	30	40	70	50	80	90	
		0	1	2	3	4	5	6	
pass 3:	k = 3	20	30	40	50	70	80	90	
		0	1	2	3	4	5	6	
pass 4:	k = 2	20	30	40	50	70	80	90	

made by electron & free999

STL

```
#include <algorithm>  ←記得!!!!
void bubble_sort(int in[], int n)
{
    int k, j;
    for(k=n-1; k>=1;k--)
        for(j=0; j<k; j++)
            if(in[j]>in[j+1])
              swap(in[j], in[j+1]);
}
```

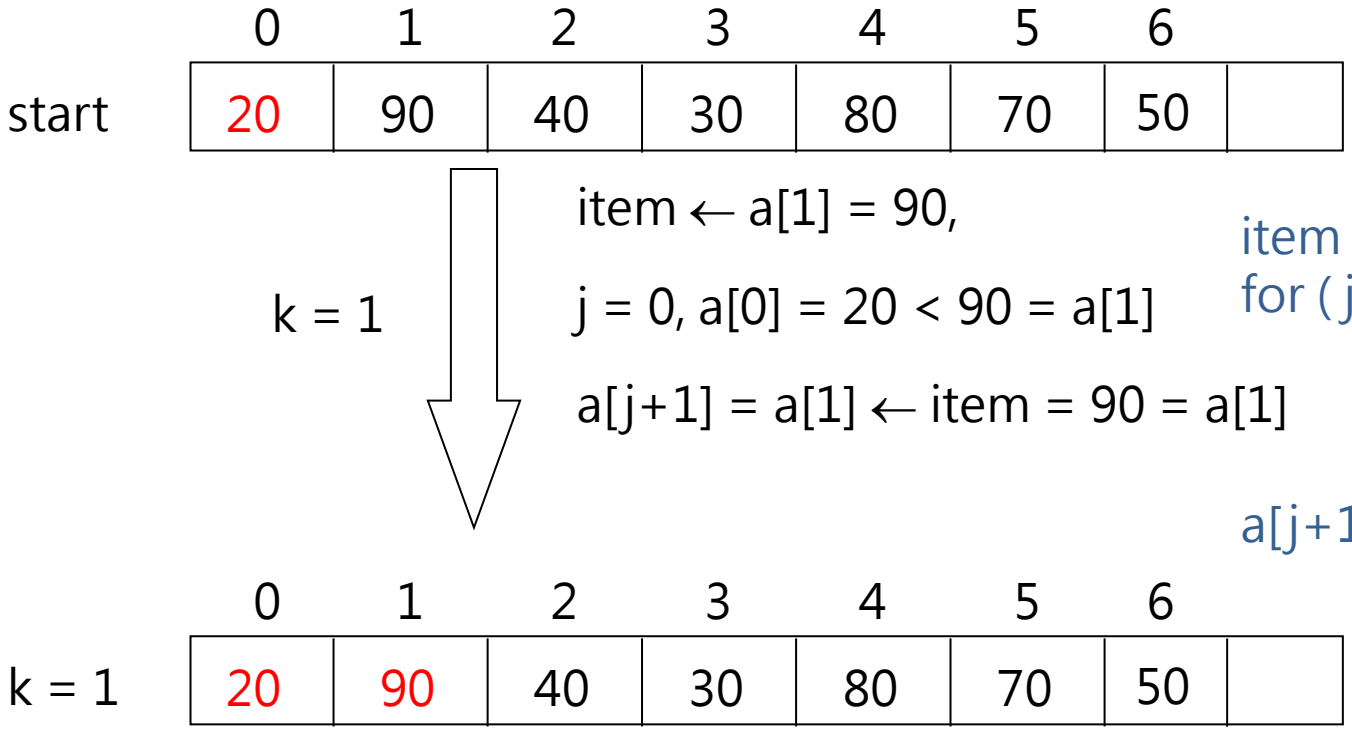
Sort - Insertion Sort

- Insert a element into a sorting list
  - Scan + Swap + branch & bound

```
void insertion_sort( int n, int a[])
{
        int k, j, item;
        for ( k = 1;  k<n;  k++)
        {
                item = a[k];
                for ( j = k-1; j>=0; j--)
                        if( a[j] > item )
                        {   a[j+1]=a[j]; }
                        else break;          // a[j] <= item
                a[j+1] = item;
        }
}
```

  - $O(N^2)$ time complexity

Sort - Insertion Sort



made by electron & free999

Sort - Insertion Sort

	0	1	2	3	4	5	6		
k = 2	20	40	90	30	80	70	50		n = 7
k = 3	20	30	40	90	80	70	50		n = 7
k = 4	20	30	40	80	90	70	50		n = 7
k = 5	20	30	40	70	80	90	50		n = 7
k = 6	20	30	40	50	70	80	90		n = 7

made by electron & free999

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.

- **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

Sort – Merge Sort

- Bubble Sort runs $O(N^2)$
- Insertion Sort runs $O(N^2)$
- Problem Occurs When Input Size >= 10000…

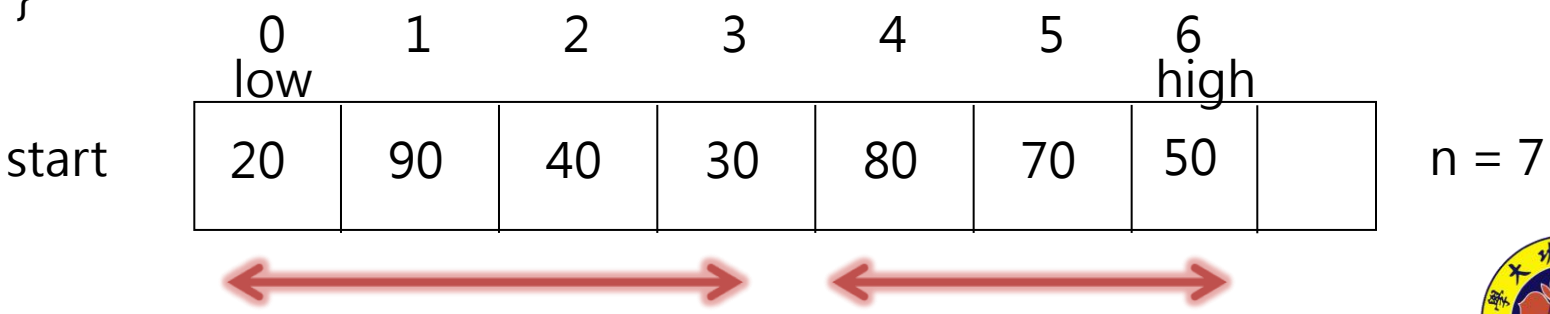
Sort – Merge Sort

- Merge Sort
  - A Divide and Conquer Based Algorithm
  - Run O(NlogN)
- Algorithm
  - **Divide**: If S has at leas two elements (nothing needs to be done if S has zero or one elements), remove all the elements from S and put them into two sequences, S_1 and S_2, each containing about half of the elements of S. (i.e. S_1 contains the first $\lceil n/2 \rceil$ elements and S_2 contains the remaining $\lfloor n/2 \rfloor$ elements.
  - **Recur**: Recursive sort sequences S_1 and S_2.
  - **Conquer**: Put back the elements into S by merging the sorted sequences S_1 and S_2 into a unique sorted sequence.

Sort – Merge Sort

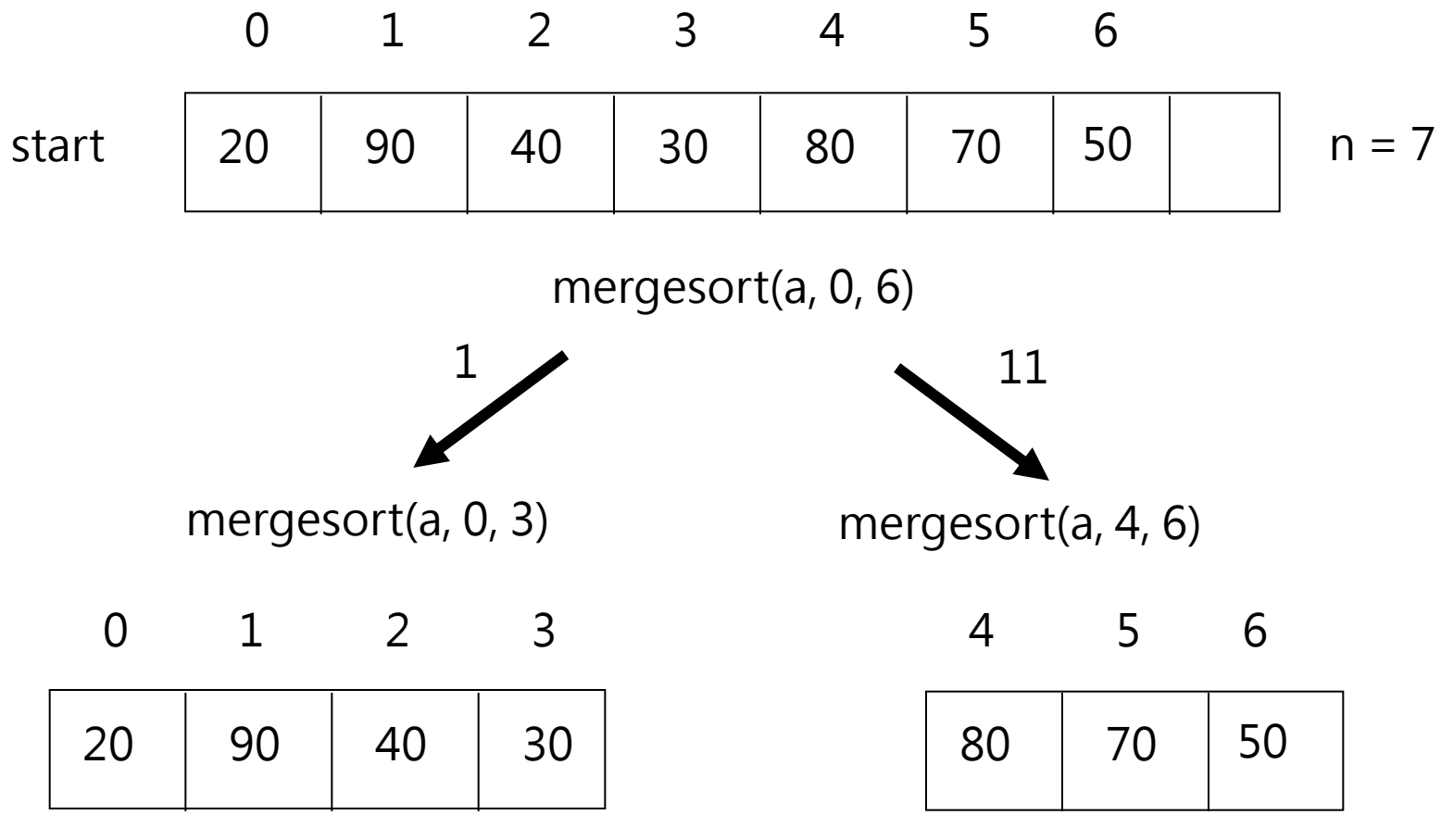
- Code 1 – Divide: mergeSort

```
void mergeSort(int a[], int low, int high)
{
        int k;
        if ( high > low)
        {
                mergeSort(a, low, (low+high)/2);
                mergeSort(a, 1+(low+high)/2, high);
                merge(a, low, high);
        }
}
```



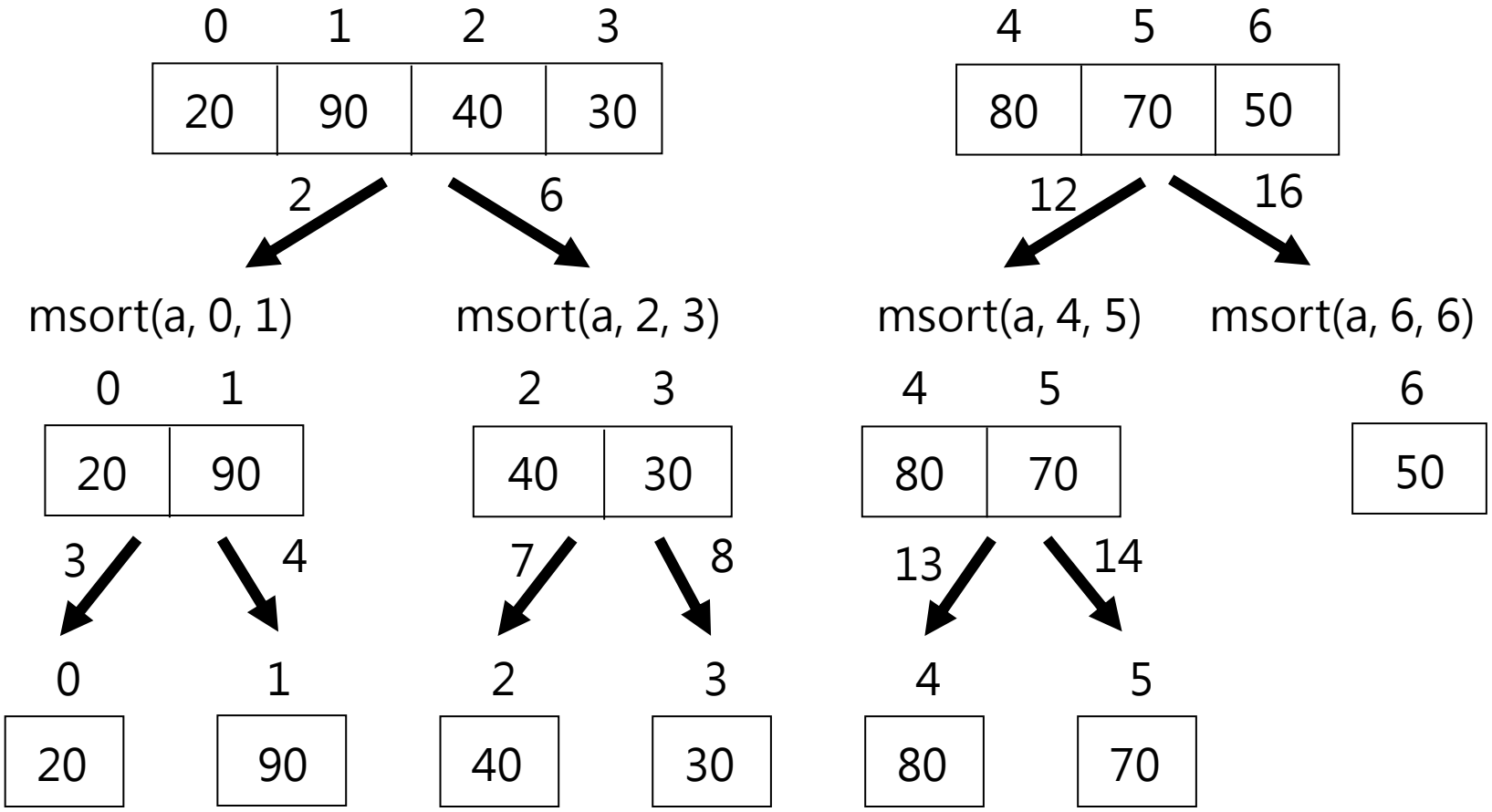
made by electron & free999

Sort – Merge Sort



made by electron & free999

Sort – Merge Sort



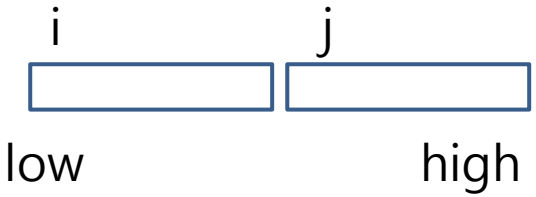

```
mergeSort(a, low, (low+high)/2);
mergeSort(a, 1+(low+high)/2, high);
merge(a, low, high);
```

made by electron & free999

Sort – Merge Sort

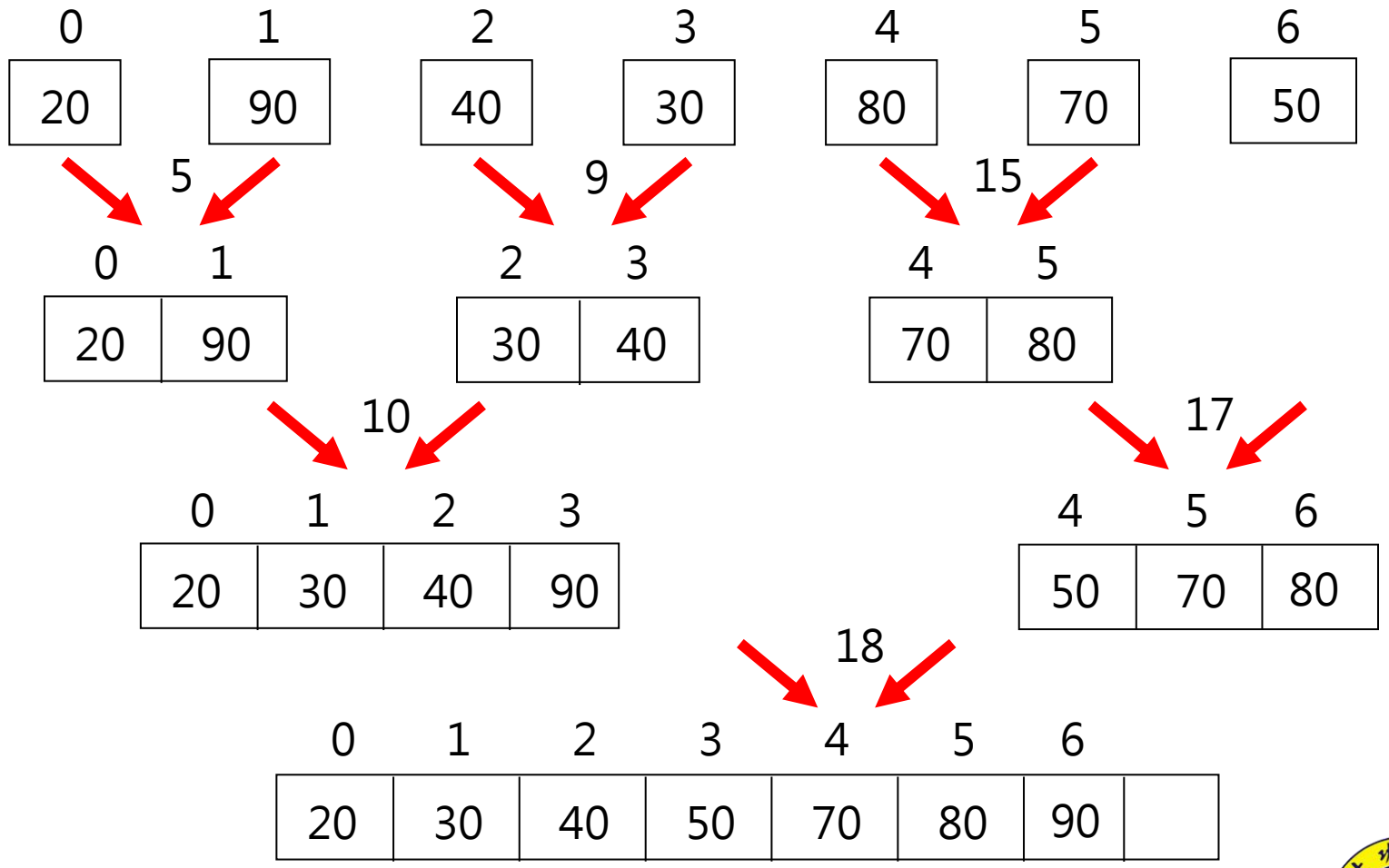
- Code 2 – Merge:

```
void merge (int a[], int low, int high)
{
        int b[MAXSIZE], i, j, k = -1, mid=(low+high)>>1;
        for (i=low, j=mid+1; i<=mid || j<=high;  )
        {
                if ( i>(low+high)/2 )  b[++k]=a[j++];
                else if (j>high)       b[++k]=a[i++];
                else if (a[i]>=a[j])   b[++k]=a[j++];
                else                   b[++k]=a[i++];
        }
        k=0;
        for(i=low; i<=high; i++) a[i]=b[k++];
}
```



made by electron & free999

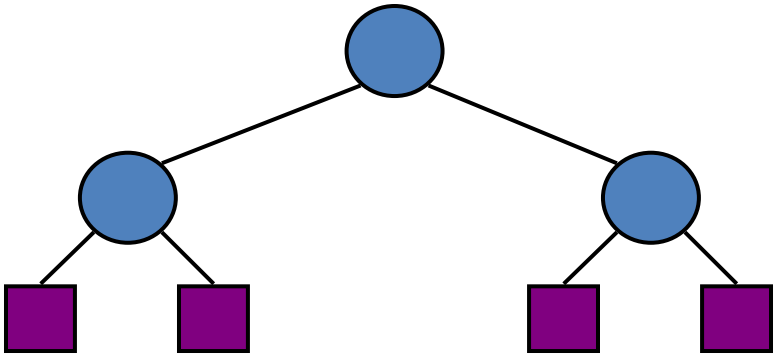
Sort – Merge Sort



made by electron & free999

Sort – Merge Sort

- Run Time Analysis
  - At each level in the binary tree created for Merge Sort, there are n elements, with O(1) time spent at each element
  - O(n) running time for processing one level
  - The height of the tree is O(log n)



- Therefore, the time complexity is $O(n \log n)$

Example 1

UVA-10810 Ultra Quick Sort

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 sequence 9 1 0 5 4 , Ultra-QuickSort produces the output 0 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 1

- Sample Input

  5
  9 1 0 5 4
  3
  1 2 3
  0

- Output for Sample Input

  6
  0

Sort – Quick Sort

- Quick Sort
  - A Divide and Conquer Based Algorithm
  - Run average O(NlogN)
- Algorithm
  - **Divide**: If the sequence S has 2 or more elements, select an element x from S to be your **pivot**. Any arbitrary element, like the last, will do. Remove all the elements of S and divide them into 3 sequences:
    - L, holds S' s elements less than x
    - E, holds S' s elements equal to x
    - G, holds S' s elements greater than x
  - **Recur**: Recursively sort L and G
  - **Conquer**: Finally, to put elements back into S in order, first inserts the elements of L, then those of E, and those of G.

made by electron & free999

Sort – Quick Sort

- Quick Sort Code

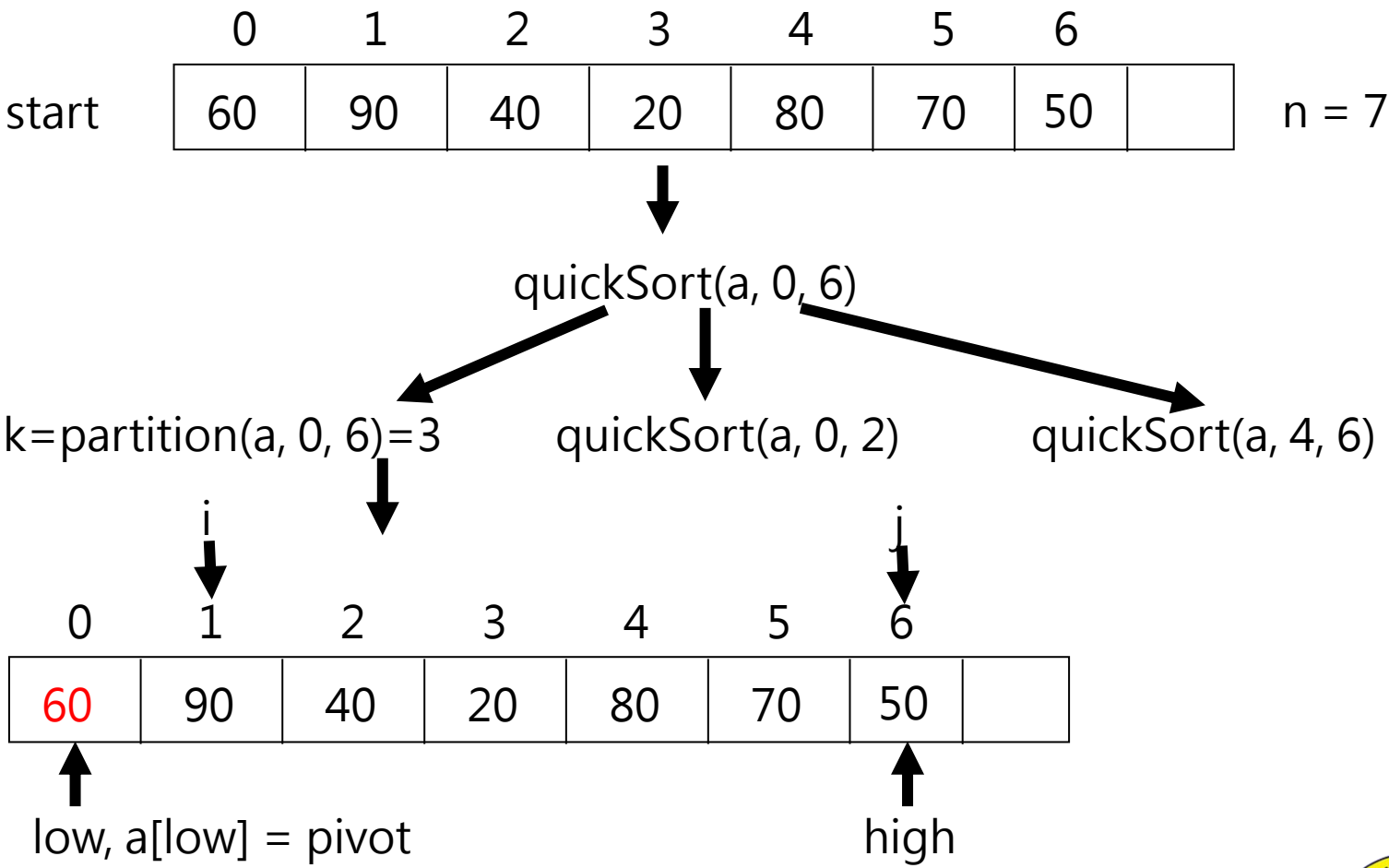
```
void quickSort(int a[], int low, int high)
{       int k;
        if ( high > low)
        {       k = partition(a, low, high);
                quickSort(a, low, k-1);
                quickSort(a, k+1, high);
        }
}
```

Sort – Quick Sort

- Quick Sort Code

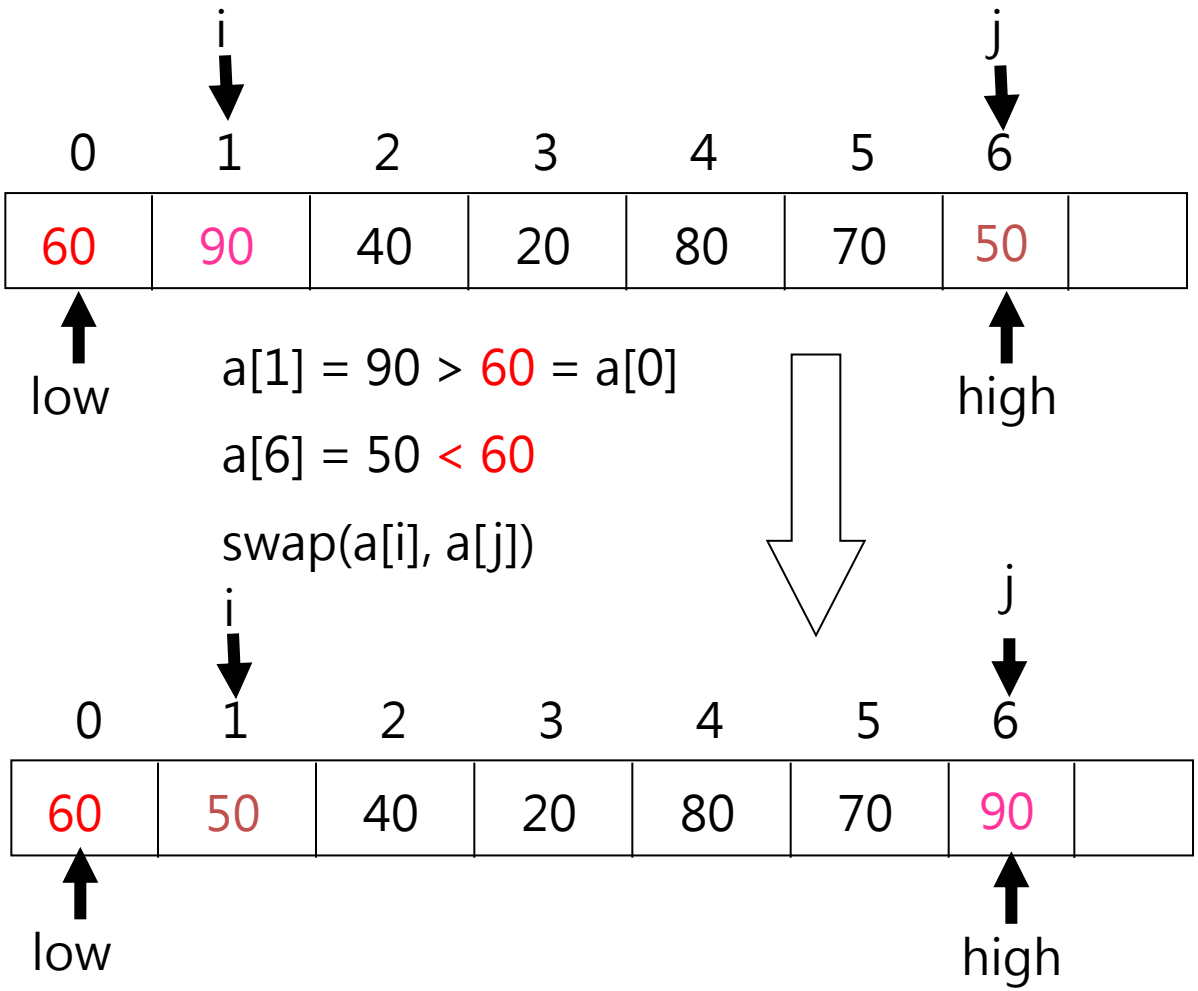
```
int partition(int a[], int low, int high)
{       int i, j;
        i = low+1;  j = high;
        while( 1 )
        {       for ( ; i<=j; i++ )  if (a[i] >= a[low]) break;
                for ( ; i<=j; j-- )  if (a[j] < a[low]) break;
                if ( i<j ) swap(a, i, j);
                else // i > j
                {       swap(a, low, j);  return j;    }
        }
}
```

Sort – Quick Sort



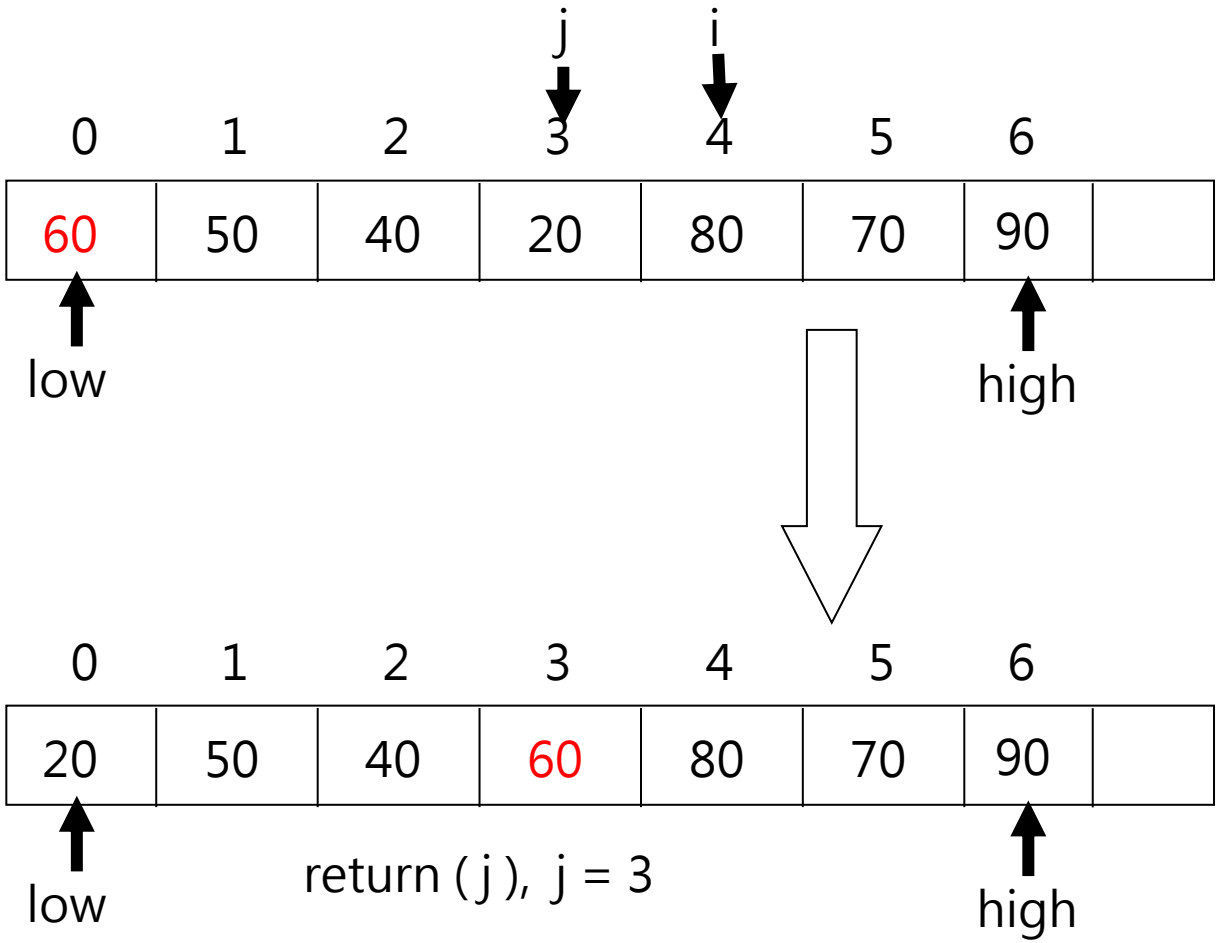
made by electron & free999

Sort – Quick Sort



made by electron & free999

Sort – Quick Sort



made by electron & free999

Sort – Quick Sort

- Quick Sort Usage

```
/* qsort example */
#include <stdio.h>
#include <stdlib.h>
int values[] = { 40, 10, 100, 90, 20, 25 };
int compare (const void * a, const void * b)
{
        return ( *(int*)a - *(int*)b );
}
 int main ()
{
        int n;
        qsort (values, 6, sizeof(int), compare);
        for (n=0; n<6; n++) printf ("%d ",values[n]);
        return 0;
}
```

Output: 10, 20, 25, 40, 90, 100

made by electron & free999

Example 1

UVA-10763 Foreign Exchange

Your non-profit organization (**iCORE** - **i**nternational **C**onfederation of **R**evolver **E**nthusiasts) coordinates a very successful foreign student exchange program. Over the last few years, demand has sky-rocketed and now you need assistance with your task.

The program your organization runs works as follows: All candidates are asked for their original location and the location they would like to go to. The program works out only if every student has a suitable exchange partner. In other words, if a student wants to go from A to B, there must be another student who wants to go from B to A. This was an easy task when there were only about 50 candidates, however now there are up to **500000** candidates!

Example 1

- Input
  - The input file contains multiple cases. Each test case will consist of a line containing **n** - the number of candidates (**1≤n≤500000**), followed by **n** lines representing the exchange information for each candidate. Each of these lines will contain **2** integers, separated by a single space, representing the candidate's original location and the candidate's target location respectively. Locations will be represented by nonnegative integer numbers. You may assume that no candidate will have his or her original location being the same as his or her target location as this would fall into the domestic exchange program. The input is terminated by a case where **n = 0**; this case should not be processed.
- Output
  - For each test case, print "**YES**" on a single line if there is a way for the exchange program to work out, otherwise print "**NO**".

Example 1

Sample Input	Output for Sample Input
10 1 2 2 1 3 4 4 3 100 200 200 100 57 2 2 57 1 2 2 1 10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 0	YES NO

made by electron & free999

STL

```
#include <algorithm> <- 記得!!!
int a[5] = { 3, 5, 4, 6, 2};
int main()
{
    sort ( a , a + 5 );
    for(int i=0;i<5;++i)
        cout<<a[i]<<  "   "; // 2, 3, 4, 5, 6
}
```

STL

```
#include <algorithm>   <- 記得!!!
struct NODE
{
    int x, y ;
    bool operator<(const NODE& t ) const
    {
        return x < t . x ;  (依據 x 由小到大排序)
    }
}p[MAXN];
sort( p , p + n ) ;
```

Example 2

UVA-10905 Children's Game

There are lots of number games for children. These games are pretty easy to play but not so easy to make. We will discuss about an interesting game here. Each player will be given **N** positive integer. (S)He can make a big integer by appending those integers after one another. Such as if there are 4 integers as 123, 124, 56, 90 then the following integers can be made – 1231245690, 1241235690, 5612312490, 9012312456, 9056124123 etc. In fact 24 such integers can be made. But one thing is sure that 9056124123 is the largest possible integer which can be made.

You may think that it's very easy to find out the answer but will it be easy for a child who has just got the idea of number?

Example 2

- **Input**
  - Each input starts with a positive integer **N** (≤ 50). In next lines there are **N** positive integers. Input is terminated by **N** = 0, which should not be processed.
- **Output**
  - For each input set, you have to print the largest possible integer which can be made by appending all the **N** integers.

Sample Input	Output for Sample Input
4 123 124 56 90 5 123 124 56 90 9 5 9 9 9 9 9 0	9056124123 99056124123 99999

made by electron & free999

Example 3

UVA-10026 Shoemaker's Problem

Shoemaker has N jobs (orders from customers) which he must make. Shoemaker can work on only one job in each day. For each i_{th} job, it is known the integer T_i ($1<=T_i<=1000$), the time in days it takes the shoemaker to finish the job. For each day of delay before starting to work for the i_{th} job, shoemaker must pay a fine of S_i ($1<=S_i<=10000$) cents. Your task is to help the shoemaker, writing a programm to find the sequence of jobs with minimal total fine.

Example 3

- The Input
  - **The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.**
  - First line of input contains an integer N (1<=N<=1000). The next N lines each contain two numbers: the time and fine of each task in order.
- The Output
  - **For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.**
  - You programm should print the sequence of jobs with minimal fine. Each job should be represented by its number in input. All integers should be placed on only one output line and separated by one space. If multiple solutions are possible, print the first lexicographically.

Example 3

- The Input
  - **The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.**
  - First line of input contains an integer N (1<=N<=1000). The next N lines each contain two numbers: the time and fine of each task in order.
- The Output
  - **For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.**
  - You programm should print the sequence of jobs with minimal fine. Each job should be represented by its number in input. All integers should be placed on only one output line and separated by one space. If multiple solutions are possible, print the first lexicographically.

Example 3

Sample Input

```
1

4
3 4
1 1000
2 2
5 5
```

Sample Output

```
2 1 3 4
```

Summary

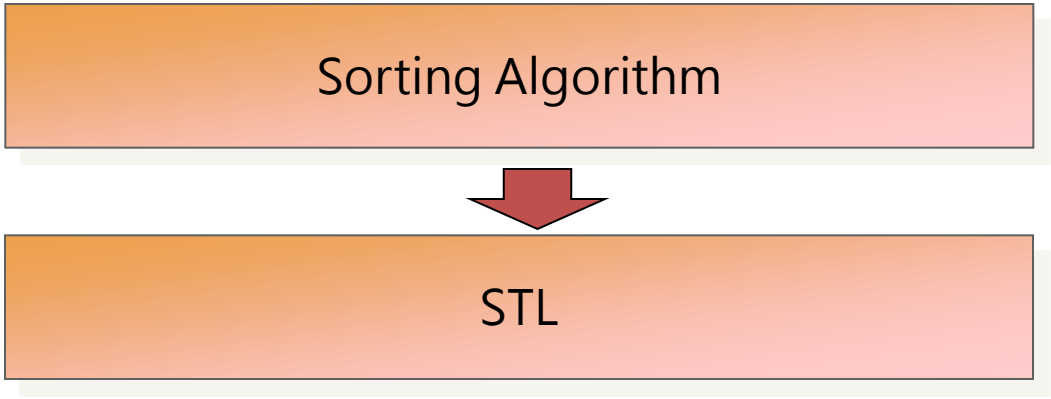
- Time Complexity

Bubble Sort	$O(n^2)$
Insert Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Quick Sort	**Average** O(n log n)
Merge Sort	O(n log n)
Heap Sort	O(n log n)
Radix Sort	O(n log n)

Take a Break

made by electron & free999

Outline



Stack

- Stack
  - A stack is an ordered list in which insertions and deletions are made at one end called the top.
  - If we add the elements *A*, *B*, *C*, *D*, *E* to the stack, in that order, then *E* is the first element we delete from the stack
  - A stack is also known as a ***Last-In-First-Out*** (***LIFO***) list.

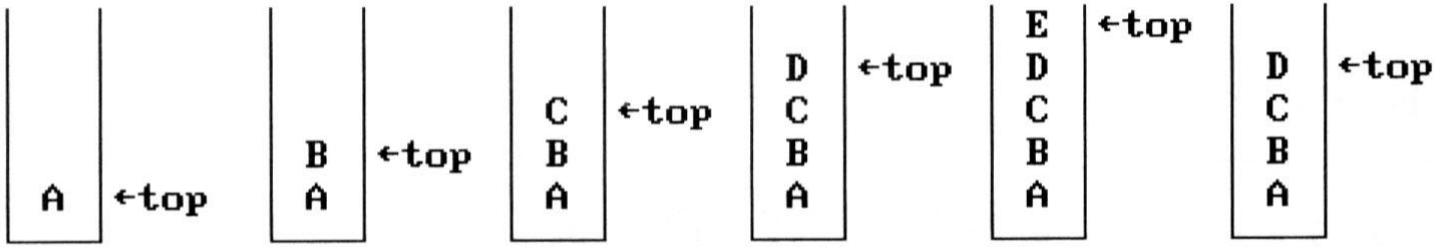
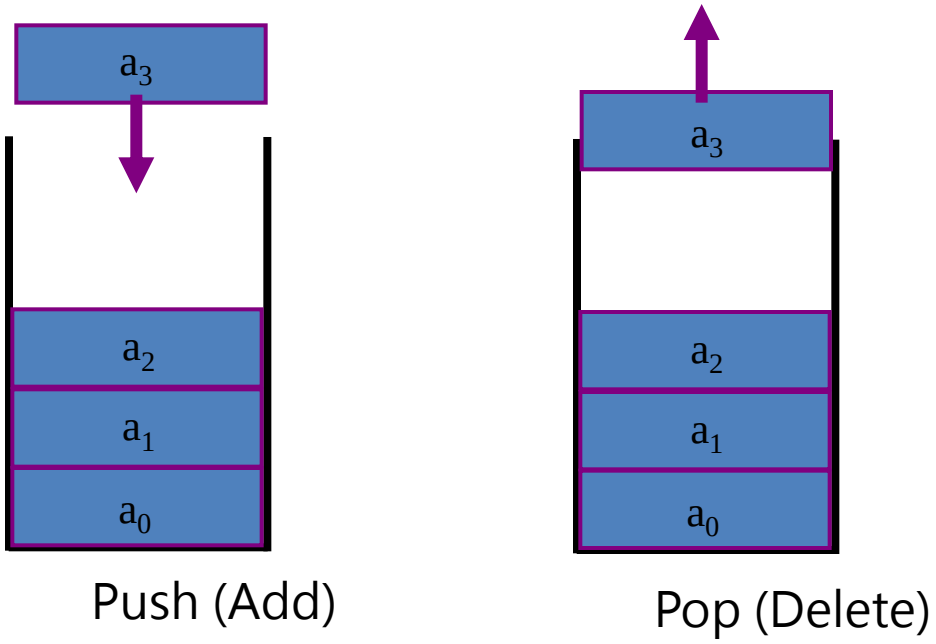


Figure 3.1: Inserting and deleting elements in a stack

made by electron & free999

Stack

- Main Subroutine
  - Push
  - Pop
  - Top
  - empty



Push (Add)

Pop (Delete)

made by electron & free999

Stack

- Stack Usage in STL

```
/* stack example */
#include <iostream>
#include <stack>
using namespace std;

int main()
{
        stack<int> stk;
        stk.push(1);
        stk.push(2);
        cout<<stk.top();   //2

        /* clear the stack */
        while(!stk.empty()) stk.pop();
}
```

Example 1

UVA-673 Parantheses

You are given a string consisting of parentheses () and []. A string of this type is said to be *correct*:

(a) if it is the empty string

(b) if A and B are correct, AB is correct,

(c) if A is correct, (A) and [A] is correct. Write a program that takes a sequence of strings of this type and check their correctness. Your program can assume that the maximum string length is 128.

Example 4

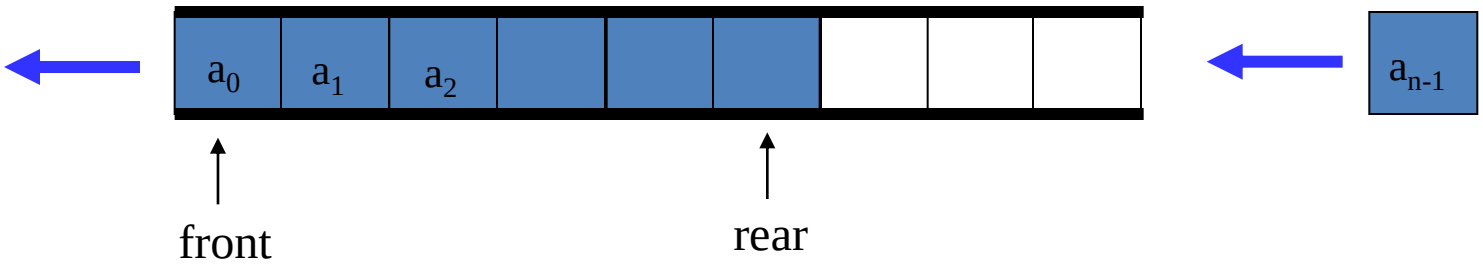
- **Input**
  - The file contains a positive integer *n* and a sequence of *n* strings of parentheses () and [], one string a line.
- **Output**
  - A sequence of Yes or No on the output file.
- **Sample Input**

  3
  ([])
  (([()]))
  ([()[]()])()
- **Sample Output**

  Yes
  No
  Yes

Queue

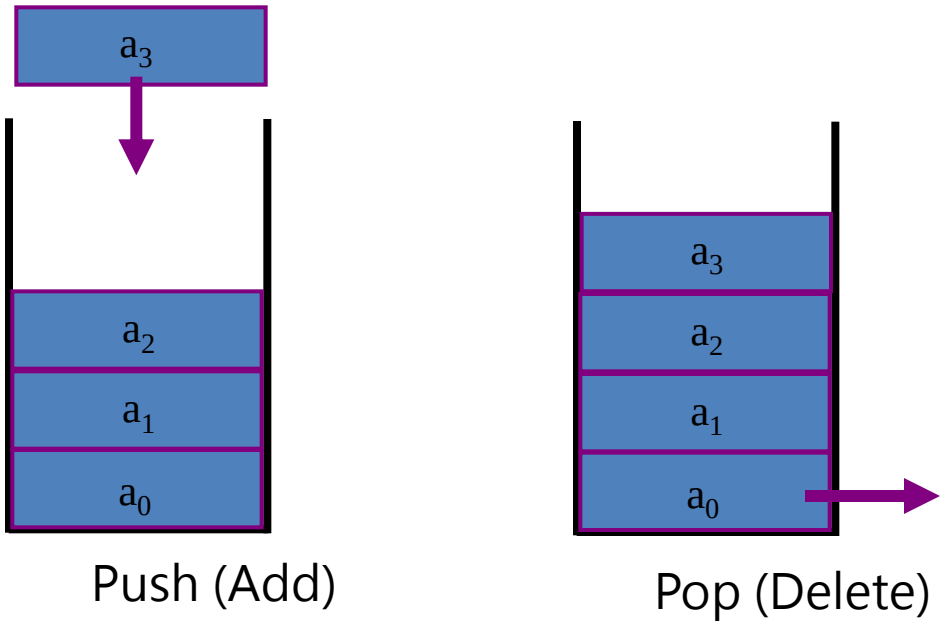
- Queue
  - A queue is an ordered list in which insertions and deletions are made at one end called the front
  - If we add the elements *A*, *B*, *C*, *D*, *E* to the stack, in that order, then *A* is the first element we delete from the queue
  - A stack is also known as a ***First-In-First-Out*** (***LIFO***) list.



made by electron & free999

Queue

- Main Subroutine
  - Push
  - Pop
  - Front
  - empty



made by electron & free999

Queue

- Queue Usage in STL

```
/* stack example */
#include <iostream>
#include <queue>
using namespace std;

int main()
{
        queue<int> que;
        que.push(1);
        que.push(2);
        cout<<que.front();

        /* clear the stack */
        while(!que.empty()) que.pop();
}
```


made by electron & free999

vector

- Basic vector operation (#include <vector>)

```
vector<int>a;                       //declaration

a.push_back(1);                     //insertion
a.push_back(2);
a.push_back(3);
                                    //IO
for(int i=0; i<a.size(); i++)
        cout<<a[i]<<" ";            //1, 2, 3

vector<int>::iterator itr;
for(itr=in.begin(); itr!=in.end(); itr++)
        cout<<*itr<<" ";

a.clear();                          //clear the list
```

vector

- Sort a vector (#include <algorithm>)

```
vector<int>a;

a.push_back(3);
a.push_back(1);
a.push_back(5);                    //3, 1, 5

sort(a.begin(), a.end());          //1, 3, 5

Sort by decreasing order…
bool cmp (int a, int b)
{
        return a > b;
}
sort(a.begin(), a.end(), cmp);     //5, 3, 1
```

vector

- Multi-Dimension

```
vector<int>a[MAXN];
a[0].push_back(1), a[0].push_back(2);
a[1].push_back(5), a[1].push_back(4);
...
```

Ex: sort

```
sort(a[1].begin(), a[1].end());
```

Ex: clear

```
for(i=0; i<MAXN; i++)
    a[i].clear();
```

Ex: IO

```
for(i=0; i<a[4].size(); i++) cout<<a[4][i]<<" ";
```

vector

- Structure Based

```
struct STUDENT
{
    char name[10];
    int score;
};
vector<STUDENT>st;

STUDENT tmp;

//assign the data to tmp
Strcpy(tmp.name , "A");
tmp.score = 7;
...
st.push_back(tmp);
```

name	score
A	7
G	3
D	2
C	1
B	5
E	4
F	6

vector

- Sort

```
//by lexical order of name
bool cmp(STUDENT a, STUDENT b)
{
    return strcmp(a.name, b.name);
}
sort(st.begin(), st.end(), cmp);


struct STUDENT
{
    char name[10];
    int score;
    bool operator<(const STUDENT& t)const
    {  return strcmp(a.name, b.name)<0;  }
};
```

name	score
A	7
B	5
C	1
D	2
E	4
F	6
G	3

vector

- Sort

```
//by increasing order of score
bool cmp(STUDENT a, STUDENT b)
{
    return a.score < b.score;
}
sort(st.begin(), st.end(), cmp);

struct STUDENT
{
    char name[10];
    int score;
    bool operator<(const STUDENT& t)const
    {  return a.score < b.score;  }
};
```

name	score
C	1
D	2
G	3
E	4
B	5
F	6
A	7

vector

- 常用

#include <vector>

vector<int>v;

v.push_back()

v.size()

v.begin()

v.end()

v.clear()

v.erase()

v.erase(v.begin()+i)

Homework 2

- Total 43 Problems
- POJ (9)
  - 1068, 1083, 1503, 1664, 1804, 1936, 2159, 3094, 2388
- Uva (21)
  - 120, 156, 369, 374, 612, 673, 755, 10017, 10026, 10152, 10327, 10763, 10810, 10815, 10905, 11069, 11110, 11462, 11495, 10249, 105
- ZOJ2 (8)
  - p006, p007, p008, p009, p012, p015, p016, p017
- NCKU judge (5)
  - 7, 8, 10, 17, 18

made by electron & free999

Thank for Your Attention

made by electron & free999