	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

## 1. Thuật toán Bubble Sort

Thuật toán sắp xếp kiểu sủi bọt được thực hiện đơn giản bằng cách trao đổi hai phần tử liền kề nhau nếu chúng chưa được sắp xếp. Thuật toán được mô tả chi tiết trong Hình 3.3.

### 1.1 Biểu diễn thuật toán

**Thuật toán Bubble-Sort:**  
**Input:**

- Dãy các đối tượng (các số) :  $Arr[0], Arr[1], \dots, Arr[n-1]$ .
- Số lượng các đối tượng cần sắp xếp:  $n$ .

**Output:**

- Dãy các đối tượng đã được sắp xếp (các số) :  $Arr[0], Arr[1], \dots, Arr[n-1]$ .

**Formats:** Insertion-Sort( $Arr, n$ );

**Actions:**

```

for (i = 0; i < n; i++) { //lặp i=0, 1, 2,...,n.
    for (j=0; j<n-i-1; j++ ) { //lặp j =0, 1,..., n-i-1
        if (Arr[j] > Arr[j+1]) { //nếu Arr[j]>Arr[j+1] thì đổi chỗ
            temp = Arr[j];
            Arr[j] = Arr[j+1];
            Arr[j+1] = temp;
        }
    }
}
End.


```

**Hình 3.3.** Thuật toán Bubble Sort

### 1.2 Độ phức tạp thuật toán

Độ phức tạp thuật toán là  $O(N^2)$ , với  $N$  là số lượng phần tử. Bạn đọc tự tìm hiểu phương pháp ước lượng và chứng minh độ phức tạp thuật toán Bubble Sort trong các tài liệu liên quan.

### 1.3 Kiểm nghiệm thuật toán


	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

**Kiểm nghiệm thuật toán:**  $Arr[] = \{ 9, 7, 12, 8, 6, 5 \}$ ,  $n = 6$ .

Bước	Dãy số $Arr[] = ?$
$i = 0$	$Arr[] = \{ 7, 9, 8, 6, 5, 12 \}$
$i = 1$	$Arr[] = \{ 7, 8, 6, 5, 9, 12 \}$
$i = 2$	$Arr[] = \{ 7, 6, 5, 8, 9, 12 \}$
$i = 3$	$Arr[] = \{ 6, 5, 7, 8, 9, 12 \}$
$i = 4$	$Arr[] = \{ 5, 6, 7, 8, 9, 12 \}$
$i = 5$	$Arr[] = \{ 5, 6, 7, 8, 9, 12 \}$

#### 1.4 Cài đặt thuật toán

```
#include <iostream>
#include <iomanip>
using namespace std;
void swap(int *x, int *y){ //đổi chỗ hai số x và y
int temp = *x; *x = *y; *y = temp;
}
void bubbleSort(int arr[], int n){ //thuật toán bubble sort
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
        }
    }
}
void printArray(int arr[], int size){
    cout<<"\n Dãy được sắp:";
    for (int i=0; i < size; i++)
        cout<<arr[i]<<setw(3);
}
int main(){
    int arr[] = { 64, 34, 25, 12, 22, 11, 90};
```

	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

```

    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);  printArray(arr,
n);
}

```

## 2. Thuật toán Quick Sort


Thuật toán sắp xếp Quick-Sort được thực hiện theo mô hình chia để trị (Divide and Conquer). Thuật toán được thực hiện xung quanh một phần tử gọi là chốt (key). Mỗi cách lựa chọn vị trí phần tử chốt trong dãy sẽ cho ta một phiên bản khác nhau của thuật toán. Các phiên bản (version) của thuật toán Quicksort bao gồm:

- Luôn lựa chọn phần tử đầu tiên trong dãy làm chốt.
- Luôn lựa chọn phần tử cuối cùng trong dãy làm chốt.
- Luôn lựa chọn phần tử ở giữa dãy làm chốt.
- Lựa chọn phần tử ngẫu nhiên trong dãy làm chốt.

Mấu chốt của thuật toán Quick-Sort là làm thế nào ta xây dựng được một thủ tục phân đoạn (Partition). Thủ tục Partition có hai nhiệm vụ chính:

- Định vị chính xác vị trí của chốt trong dãy nếu được sắp xếp;
- Chia dãy ban đầu thành hai dãy con: dãy con ở phía trước phần tử chốt bao gồm các phần tử nhỏ hơn hoặc bằng chốt, dãy ở phía sau chốt có giá trị lớn hơn chốt.

Thuật toán Partition được mô tả chi tiết trong Hình 3.4 với khóa chốt là phần tử cuối cùng của dãy. Phiên bản Quick Sort tương ứng được mô tả chi tiết trong Hình 3.5.

	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

## 2.1 Biểu diễn thuật toán

### Thuật toán Partition:

#### Input :

- Dãy Arr[] bắt đầu tại vị trí l và kết thúc tại h.
- Cận dưới của dãy con: l
- Cận trên của dãy con: h

#### Output:

- Vị trí chính xác của Arr[h] nếu dãy Arr[] được sắp xếp.

**Formats:** Partition(Arr, l, h);

#### Actions:

```

x = Arr[h]; // chọn x là chốt của dãy Arr[l], Arr[l+1], ..., Arr[h]
i = (l - 1); // i là chỉ số của các phần tử nhỏ hơn chốt x.
for ( j = l; j <= h - 1; j++) { // duyệt các chỉ số j=l, l+1, ..., h-1.
    if (Arr[j] <= x) { // nếu Arr[j] nhỏ hơn hoặc bằng chốt x.
        i++; // tăng vị trí các phần tử nhỏ hơn chốt
        swap(&Arr[i], &Arr[j]); // đổi chỗ Arr[i] cho Arr[j].
    }
}
swap(&Arr[i + 1], &Arr[h]); // đổi chỗ Arr[i+1] cho Arr[h].
return (i + 1); // vị trí i+1 chính là vị trí chốt nếu dãy được sắp xếp

```

**End.**

**Hình 3.4.** Thuật toán Partition với chốt là vị trí cuối cùng của dãy

### Thuật toán Quick-Sort:

#### Input :

- Dãy Arr[] gồm n phần tử.
- Cận dưới của dãy: l.
- Cận trên của dãy : h

#### Output:

- Dãy Arr[] được sắp xếp.

**Formats:** Quick-Sort(Arr, l, h);

#### Actions:

```

if (l < h) { // Nếu cận dưới còn nhỏ hơn cận trên
    p = Partition(Arr, l, h); // thực hiện Partition() chốt h.
    Quick-Sort(Arr, l, p-1); // thực hiện Quick-Sort nửa bên trái.
    Quick-Sort(Arr, p+1, h); // thực hiện Quick-Sort nửa bên phải
}


```

**End.**

**Hình 3.5.** Thuật toán Quick-Sort với chốt là vị trí cuối cùng của dãy

## 2.2 Độ phức tạp thuật toán

Độ phức tạp thuật toán trong trường hợp xấu nhất là  $O(N^2)$ , trong trường hợp tốt nhất là

	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

$O(N \cdot \log(N))$ ), với  $N$  là số lượng phần tử. Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán Quick Sort trong các tài liệu liên quan.


### 2.3 Kiểm nghiệm thuật toán

**Kiểm nghiệm thuật toán Quick-Sort:** Quick-Sort(Arr, 0, 9);  
 Arr[] = {10, 27, 15, 29, 21, 11, 14, 18, 12, 17};  
 Cận dưới  $l=0$ , cận trên  $h = 9$ .

p = Partition(Arr,l,h)	Giá trị Arr[]=?
p=5:l=0, h=9	{10,15,11,14,12}, (17),{29,18, 21, 27}
P=2:l=0, h=4	{10,11},{(12)}, {14,15}, (17),{29,18, 21, 27}
P=1:l=0, h=1	{10,11},{(12)}, {14,15}, (17),{29,18, 21, 27}
P=4: l=3, h=4	{10,11},{(12)}, {14,15}, (17),{29,18, 21, 27}
P=8: l=6, h=9	{10,11},{(12)}, {14,15}, (17),{18,21},{(27)},{29}
P=7:l=6, h=7	{10,11},{(12)}, {14,15}, (17),{18,21},{(27)},{29}
Kết luận dãy được sắp Arr[] = { 10, 11, 12, 14, 15, 17, 18, 21, 27, 29}	

### 2.4. Cài đặt thuật toán

```
#include<iostream>
#include<iomanip>
using namespace std;
void swap(int* a, int* b){ //đổi chỗ a và b
    int t = *a; *a = *b; *b = t;
}
int partition (int arr[], int l, int h){ //thuật toán partition chốt h
    int x = arr[h]; // x chính là chốt
    int i = (l - 1); // i lấy vị trí nhỏ hơn l
    for (int j = l; j <= h- 1; j++) { //duyet từ l đến h-1
        // If current element is smaller than or equal to
        // pivot if (arr[j] <= x){ //nếu arr[j] bé hơn hoặc
        // bằng chốt
            i++; // tăng i lên một đơn vị
            swap(&arr[i], &arr[j]); // đổi chỗ arr[i] cho arr[j]
        }
    }
    swap(&arr[i + 1], &arr[h]); //đổi chỗ cho arr[i+1] và arr[h]
```

	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

```


        return (i + 1); //đây là vị trí của chốt
    }
    void quickSort(int arr[], int l, int
        h){ if (l < h){
            int p = partition(arr, l, h); // tìm vị trí của chốt
            quickSort(arr, l, p - 1); //trị nửa bên trái
            quickSort(arr, p + 1, h); //trị nửa bên phải
        }
    }
    void printArray(int arr[], int size){ //thủ tục in kết quả
        int i; cout<<"\n Dãy được
        sắp:"; for (i=0; i < size; i++)
            cout<<arr[i]<<setw(3);
    }
    int main(){ //chương trình chính
        int arr[] = {10, 27, 15, 29, 21, 11, 14, 18, 12, 17};
        int n =
        sizeof(arr)/sizeof(arr[0]);
        quickSort(arr, 0, n-1);
        printArray(arr, n);
    }

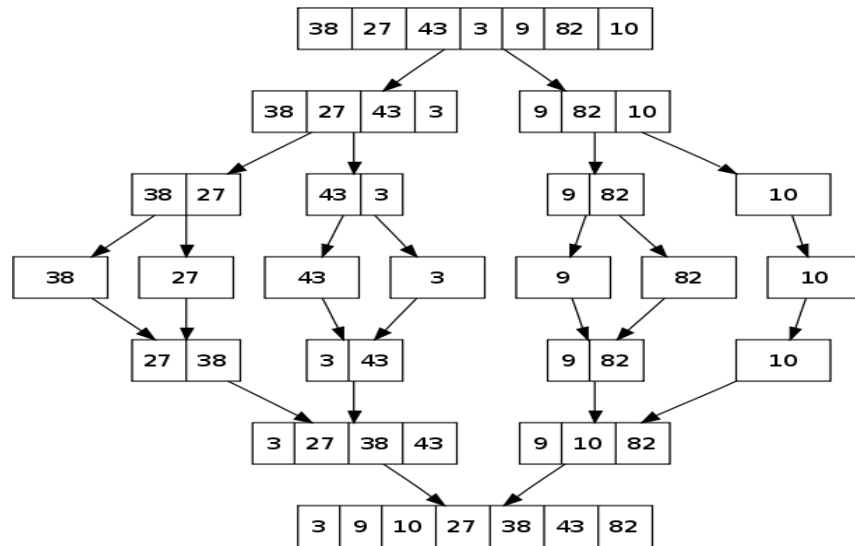
```

### 3. Thuật toán Merge Sort

Giống như Quick-Sort, Merge-Sort cũng được xây dựng theo mô hình chia để trị (Divide and Conquer). Thuật toán chia dãy cần sắp xếp thành hai nửa. Sau đó gọi đệ qui lại cho mỗi nửa và hợp nhất lại các đoạn đã được sắp xếp. Thuật toán được tiến hành theo 4 bước dưới đây:

- Tìm điểm giữa của dãy và chia dãy thành hai nửa.
- Thực hiện Merge-Sort cho nửa thứ nhất.
- Thực hiện Merge-Sort cho nửa thứ hai.
- Hợp nhất hai đoạn đã được sắp xếp.

	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1




Mấu chốt của thuật toán Merge-Sort là làm thế nào ta xây dựng được một thủ tục hợp nhất (Merge). Thủ tục Merge thực hiện hòa nhập hai dãy đã được sắp xếp để tạo thành một dãy cũng được sắp xếp. Bài toán có thể được phát biểu như sau:

**Bài toán hợp nhất Merge:** Cho hai nửa của một dãy  $Arr[1,..,m]$  và  $A[m+1,..,r]$  đã được sắp xếp. Nhiệm vụ của ta là hợp nhất hai nửa của dãy  $Arr[1,..,m]$  và  $Arr[m+1,..,r]$  để trở thành một dãy  $Arr[1, 2,..,r]$  cũng được sắp xếp.

Thuật toán Merge được mô tả chi tiết trong Hình 3.6. Thuật toán Merge Sort được mô tả chi tiết trong Hình 3.7.

**Biểu diễn thuật toán:**

	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

**Thuật toán Merge:**

**Input:** Arr[], l, m, r. Trong đó, Arr[l]..Arr[m] và Arr[l+1]..Arr[r] đã được sắp

**output:** Arr[] có Arr[l]..Arr[r] đã được sắp

Begin:

```

int i, j, k, n1 = m - l + 1; n2 = r - m;
int L[n1], R[n2]; //tạo lập hai mảng phụ L và R
for(i = 0; i < n1; i++) //lưu đoạn được sắp thứ nhất vào L
    L[i] = Arr[l + i];
for(j = 0; j < n2; j++) //lưu đoạn được sắp thứ nhất vào R
    R[j] = Arr[m + 1 + j];
// hợp nhất các mảng phụ và trả lại vào Arr[]
i = 0; j = 0; k = l;
while (i < n1 && j < n2){
    if (L[i] <= R[j]){ Arr[k] = L[i]; i++; }
    else { Arr[k] = R[j]; j++; }
    k++;
}
while (i < n1) {
    Arr[k] = L[i]; i++; k++;
}
while (j < n2) {
    arr[k] = R[j]; j++; k++;
}

```

End.

**Hình 3.6.** Thuật toán hợp nhất hai đoạn đã được sắp xếp.

**Thuật toán Merge-Sort:**

**Input :**

- Dãy số : Arr[];
- Cận dưới: l;
- Cận trên m;

**Output:**

- Dãy số Arr[] được sắp theo thứ tự tăng dần.

**Formats:** Merge-Sort(Arr, l, r);

**Actions:**

```

if ( l < r ) {
    m = (l + r - 1) / 2; //phép chia Arr[] thành hai nửa
    Merge-Sort(Arr, l, m); //trị nửa thứ nhất
    Merge-Sort(Arr, m+1, r); //trị nửa thứ hai
    Merge(Arr, l, m, r); //hợp nhất hai nửa đã sắp xếp
}

```


**End.**

**Hình 3.7.** Thuật toán Merge Sort

### 3.1 Độ phức tạp thuật toán

Độ phức tạp thuật toán là  $O(N \cdot \log(N))$  với N là số lượng phần tử. Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán Merge Sort trong các tài liệu liên quan.



	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

### 3.2 Kiểm nghiệm thuật toán


**Kiểm nghiệm thuật toán Merge-Sort:** Merge-Sort(Arr,0,7)

**Input** : Arr[] = {38, 27, 43, 3, 9, 82, 10}; n = 7;

Bước	Kết quả Arr[]=?
1	Arr[] = { 27, 38, 43, 3, 9, 82, 10}
2	Arr[] = { 27, 38, 3, 43, 9, 82, 10}
3	Arr[] = { 3, 27, 38, 43, 9, 82, 10}
4	Arr[] = {3, 27, 38, 43, 9, 82, 10}
5	Arr[] = { 3, 27, 38, 43, 9, 10, 82}
6	Arr[] = { 3, 9, 10, 27, 38, 43, 82}

### 3.3 Cài đặt thuật toán

```
#include<iostream>
#include<iomanip>
using namespace std;
void merge(int arr[], int l, int m, int r){//thuật toán hợp nhất hai đoạn đã sắp xếp
    int i, j, k;
    int n1 = m - l + 1; //số lượng phần tử đoạn 1
    int n2 = r - m; //số lượng phần tử đoạn 2
    int L[n1], R[n2]; //tạo hai mảng phụ để lưu hai đoạn được sắp
    for(i = 0; i < n1; i++) //lưu đoạn thứ nhất vào L[]
        L[i] = arr[l + i];
    for(j = 0; j < n2; j++) //lưu đoạn thứ hai vào R[]
        R[j] = arr[m + 1 + j];
    i = 0; j = 0; k = l; //bắt đầu hợp nhất
    while (i < n1 && j < n2) { //quá trình hợp nhất
        if (L[i] <= R[j]) {
            arr[k] = L[i]; i++;
        }
        else {
            arr[k] = R[j]; j++;
        } k++;
    } while (i < n1) { //lấy các phần tử còn lại trong L[] vào arr[]
        arr[k] = L[i];
        i++; k++;
    }
    while (j < n2) { //lấy các phần tử còn lại trong R[] vào arr[]
```

	VIETTEL AI RACE	TD054
	THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2	Lần ban hành: 1

```

        arr[k] = R[j];
        j++; k++;
    }
}

void mergeSort(int arr[], int l, int r) { //thuật toán Merge Sort
    if (l < r) { //nếu cận dưới còn bé hơn cận trên
        int m = l+(r-l)/2; //tìm vị trí ở giữa đoạn l, r
        mergeSort(arr, l, m); //trị nửa thứ nhất
        mergeSort(arr, m+1, r); //trị nửa thứ hai
        merge(arr, l, m, r); //hợp nhất hai đoạn đã được sắp
    }
}

void printArray(int Arr[], int size) { //in kết quả
    int i; cout<<"\n Dãy được sắp:";
    for (i=0; i < size; i++)
        cout<<Arr[i]<<setw(3);
}

int main() {
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
    int arr_size =
        sizeof(arr)/sizeof(arr[0]);
    mergeSort(arr, 0, arr_size - 1);
    printArray(arr, arr_size);
}


```

#### 4. Thuật toán Heap Sort

Thuật toán Heap-Sort được thực hiện dựa trên cấu trúc dữ liệu Heap. Nếu ta muốn sắp xếp theo thứ tự tăng dần ta sử dụng cấu trúc Max Heap, ngược lại ta sử dụng cấu trúc Min-Heap. Vì Heap là một cây nhị phân đầy đủ nên việc biểu diễn Heap một cách hiệu quả có thể thực hiện được bằng mảng. Nếu ta xem xét phần tử thứ  $i$  trong mảng thì phần tử  $2*i + 1$ ,  $2*i + 2$  tương ứng là node con trái và node con phải của  $i$ .

Tư tưởng của Heap Sort giống như Selection Sort, chọn phần tử lớn nhất trong dãy đặt vào vị trí cuối cùng, sau đó lặp lại quá trình này cho các phần tử còn lại. Tuy nhiên, điểm khác biệt ở đây là phần tử lớn nhất của Heap luôn là phần tử đầu tiên trên Heap và các phần tử node trái và phải bao giờ cũng nhỏ hơn nội dung node gốc.

Thuật toán được thực hiện thông qua ba bước chính như sau:

	<b>VIETTEL AI RACE</b>	TD054
	<b>THUẬT TOÁN SẮP XẾP KINH ĐIỂN - PHẦN 2</b>	Lần ban hành: 1

- Xây dựng Max Heap từ dữ liệu vào. Ví dụ với dãy  $A[] = \{9, 7, 12, 8, 6, 5\}$  thì Max Heap được xây dựng là  $A[] = \{12, 8, 9, 7, 6, 5\}$ .
- Bắt đầu tại vị trí đầu tiên là phần tử lớn nhất của dãy. Thay thế, phần tử này cho phần tử cuối cùng ta nhận được dãy  $A[] = \{5, 8, 9, 7, 6, 12\}$ .