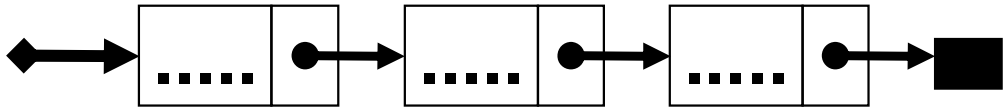
DANH SÁCH ĐƠN

❖TỔ CHỨC

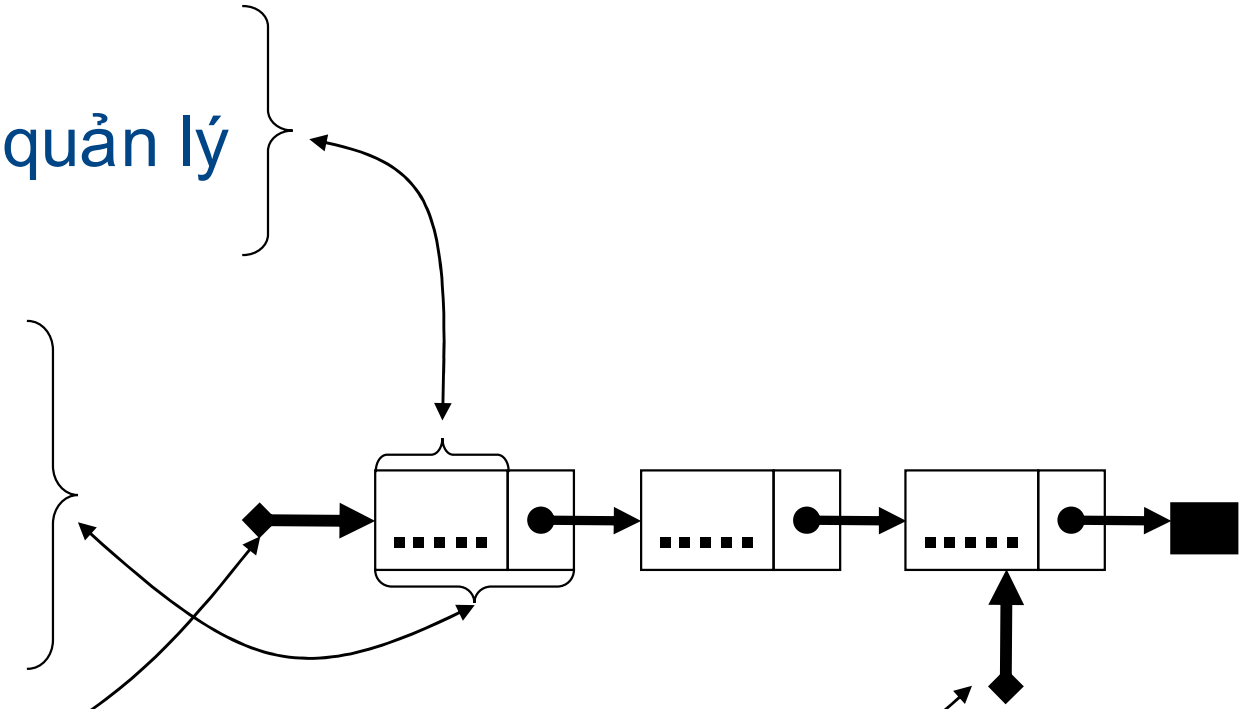


- Mỗi phần tử chứa liên kết đến phần tử đứng liền sau nó
- Mỗi phần tử là một cấu trúc gồm hai thành phần:
- Thành phần dữ liệu: chứa thông tin cần quản lý
- Thành phần liên kết: chứa địa chỉ của phần tử liền sau nó hoặc giá trị NULL nếu là phần tử cuối danh sách

DANH SÁCH ĐƠN

❖TỔ CHỨC

```
struct TenDulieu {
    ... // Thông tin cần quản lý
};
struct Node {
    TenDulieu info;
    Node * pNext;
};
struct TenDS {
    Node *pHead, *pTail;
};
```



DANH SÁCH ĐƠN

❖TỔ CHỨC

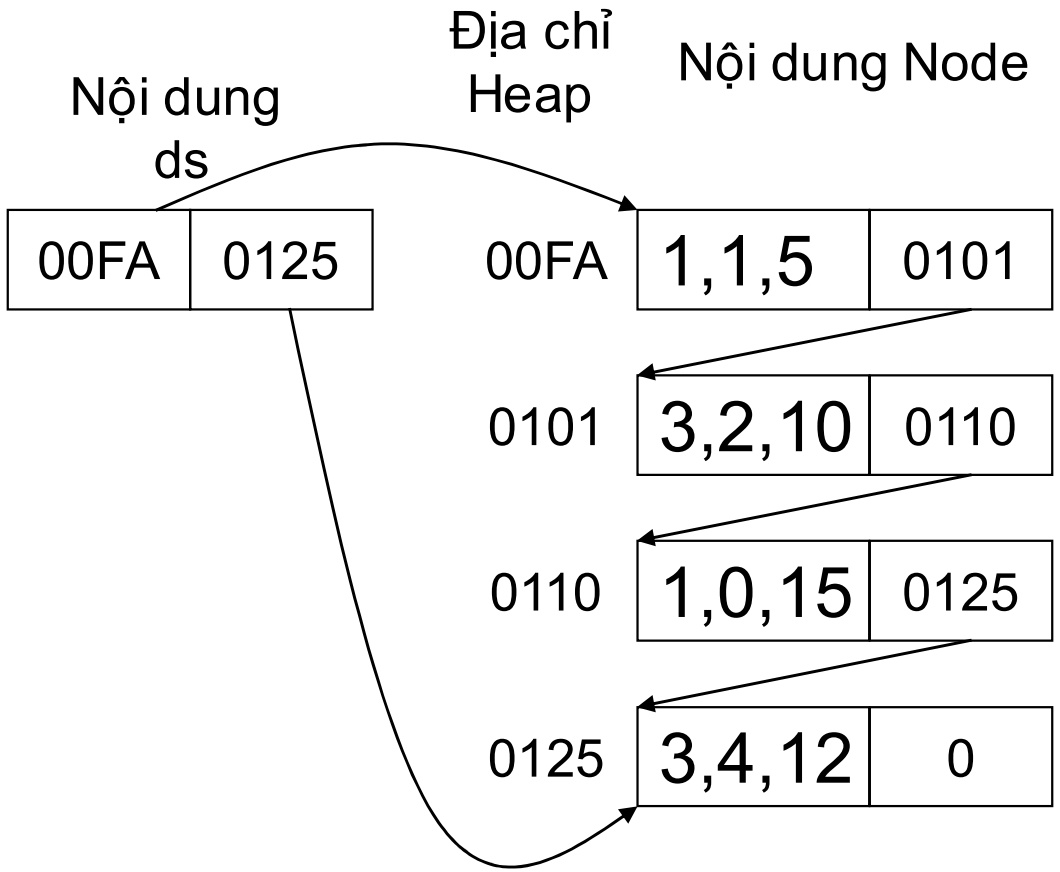
Ví dụ: Tổ chức dữ liệu cho một danh sách các hình tròn.

```
struct  HinhTron{
    double x, y, r;
};
struct NodeHinhTron {
    HinhTron info;
    NodeHinhTron *pNext;
};
```

DANH SÁCH ĐƠN

```
struct  DSHinhTron{
    NodeHinhTron *pHead,
    *pTail;
};
```

Giả sử có biến cấp phát tĩnh ds có kiểu DSHinhTron lưu trữ danh sách 4 hình tròn. Hình ảnh của ds như sau:



DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- Tạo danh sách đơn rỗng
- Tạo một nút có trường info bằng x
- Thêm phần tử vào danh sách
- Duyệt danh sách
- Hủy phần tử trong danh sách
- Hủy danh sách
- Sắp xếp danh sách

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Tạo danh sách đơn rỗng**

  Danh sách rỗng có pHead và pTail trỏ đến NULL

```
void CreateList(TenDS &p) {
   p.pHead = NULL; p.pTail = NULL;
}
```

Ví dụ

```
void CreateDSHinhTron(DSHinhTron &p) {
   p.pHead = NULL; p.pTail = NULL;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Tạo một nút có trường info bằng x**

  Tạo nút bằng cách cấp phát động một biến có kiểu Node, sau đó gán giá trị x cho trường info. Lúc này, nút vừa tạo chưa thuộc danh sách nên mặc định pNext mang giá trị NULL.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Tạo một nút có trường info bằng x**

```
Node* CreateNode(TenDuLieu x) {
    Node *p = new Node; // cấp phát vùng nhớ
    if (p != NULL) { // kiểm tra kết quả cấp phát
        p->info = x;
        p->pNext = NULL;
    }
    return p;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Tạo một nút có trường info bằng x**

Ví dụ

```
NodeHinhTron* CreateDSHinhTron(HinhTron x) {
    NodeHinhTron *p = new NodeHinhTron;
    if (p != NULL) {
        p->info = x;
        p->pNext = NULL;
    }
    return p;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**

  Xét việc thêm phần tử vào danh sách theo các trường hợp sau:

  - Thêm phần tử vào đầu danh sách
  - Thêm phần tử vào cuối danh sách
  - Thêm phần tử vào ngay sau phần tử q trong danh sách.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào đầu danh sách*

Thuật toán:

- Đầu vào: Danh sách l, phần tử p.
- Đầu ra: Danh sách l.

B1) Nếu ds rỗng: l.pHead ← p, l.pTail ← p

Ngược lại: p->pNext ← l.pHead, l.pHead ← p

B2) Kết thúc

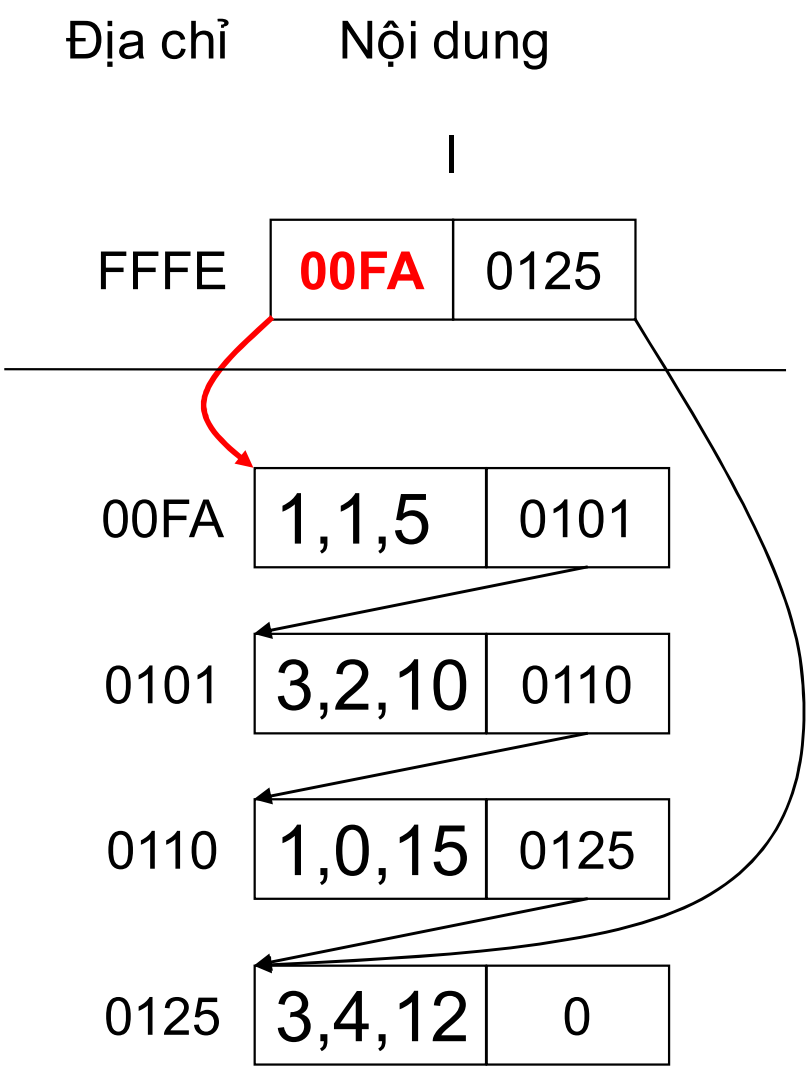
DANH SÁCH ĐƠN

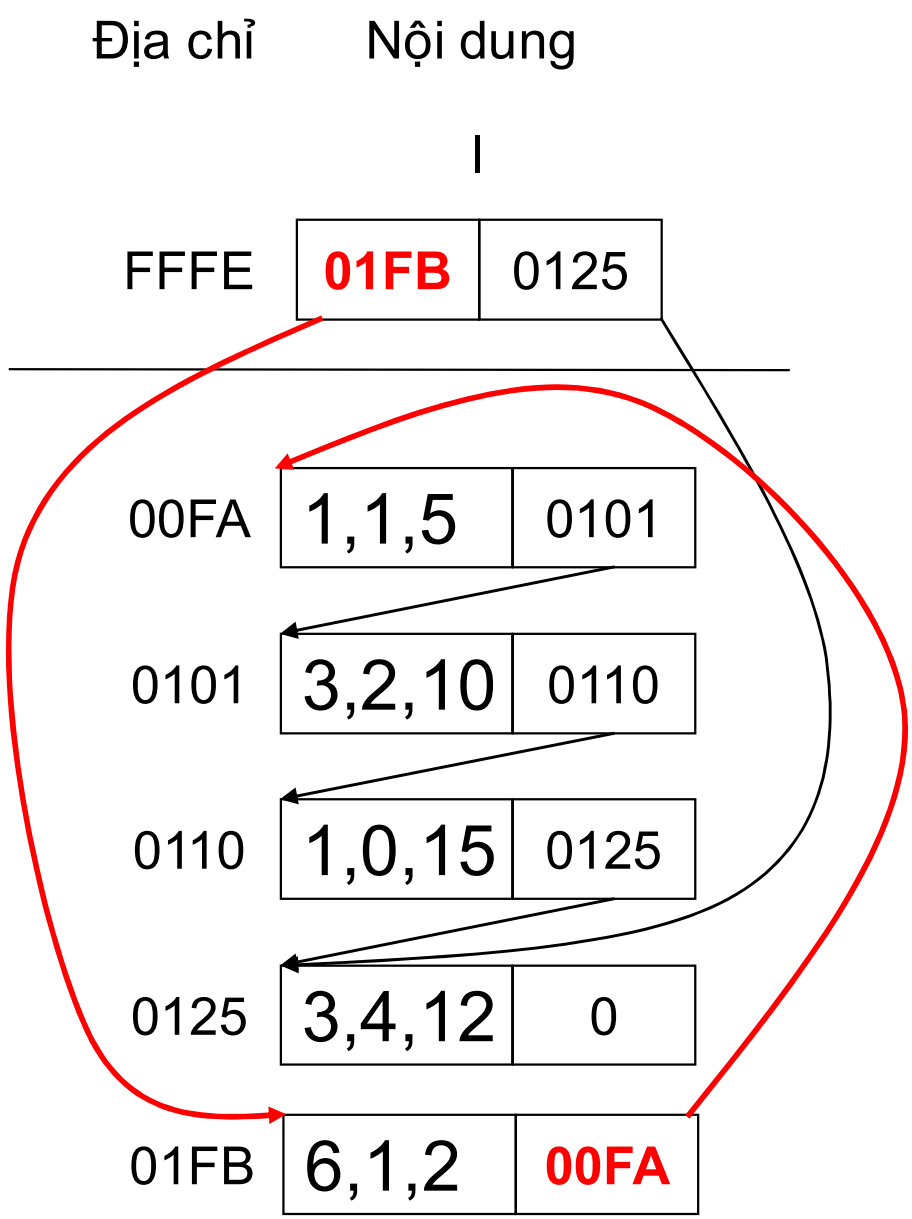
❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào đầu danh sách*

```
void AddFirst(TenDS &l, Node *p) {
    if (l.pHead == NULL) {
        l.pHead = p; l.pTail = p;
    }
    else {
        p->pNext = l.pHead; l.pHead = p;
    }
}
```

DANH SÁCH ĐƠN





DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào cuối danh sách*

Thuật toán:

- Đầu vào: Danh sách l, phần tử p.
- Đầu ra: Danh sách l.

B1) Nếu ds rỗng: l.pHead ← p, l.pTail ← p

Ngược lại: l.pTail->pNext ← p, l.Tail ← p

B2) Kết thúc

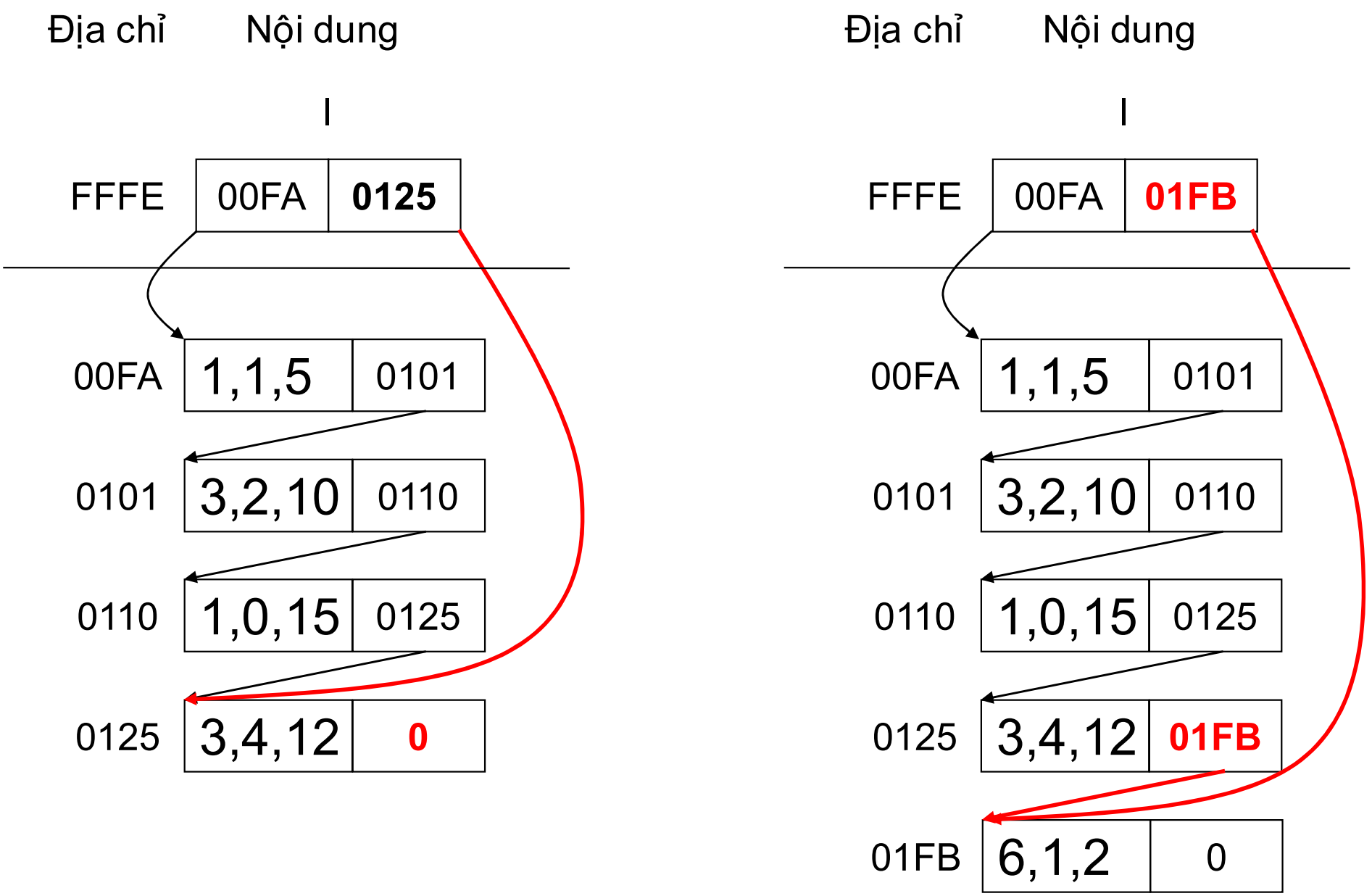
DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào cuối danh sách*

```
void AddLast(TenDS &l, Node *p) {
    if (l.pHead == NULL) {
        l.pHead = p; l.pTail = p;
    }
    else {
        l.pTail->pNext = p;l.pTail = p;
    }
}
```

DANH SÁCH ĐƠN



DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào sau phần tử q trong danh sách*

Thuật toán:

- Đầu vào: Danh sách l, phần tử p, phần tử q.

- Đầu ra: Danh sách l.

B1) Nếu q != NULL thì p->pNext ← q->pNext, q->pNext←p, ngược lại qua B3.

B2) Nếu l.pTail = q thì l.pTail ← p, qua B4.

B3) Thêm p vào đầu danh sách l.

B4) Kết thúc.

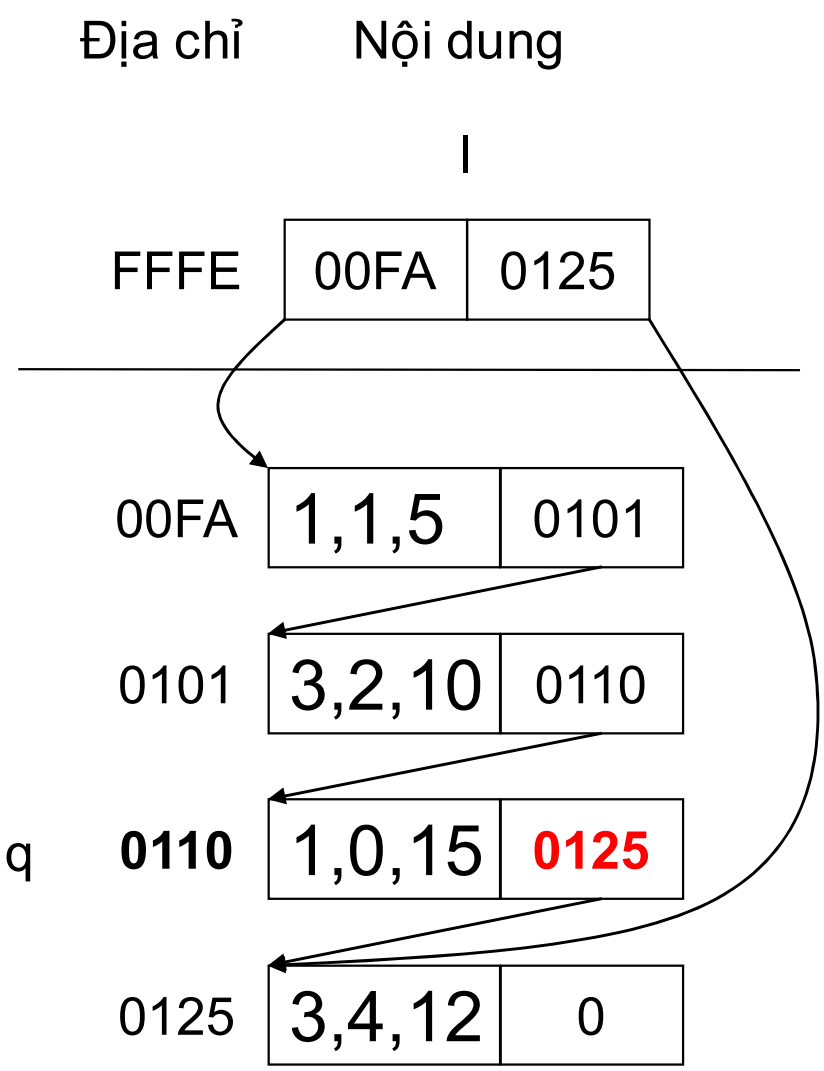
DANH SÁCH ĐƠN

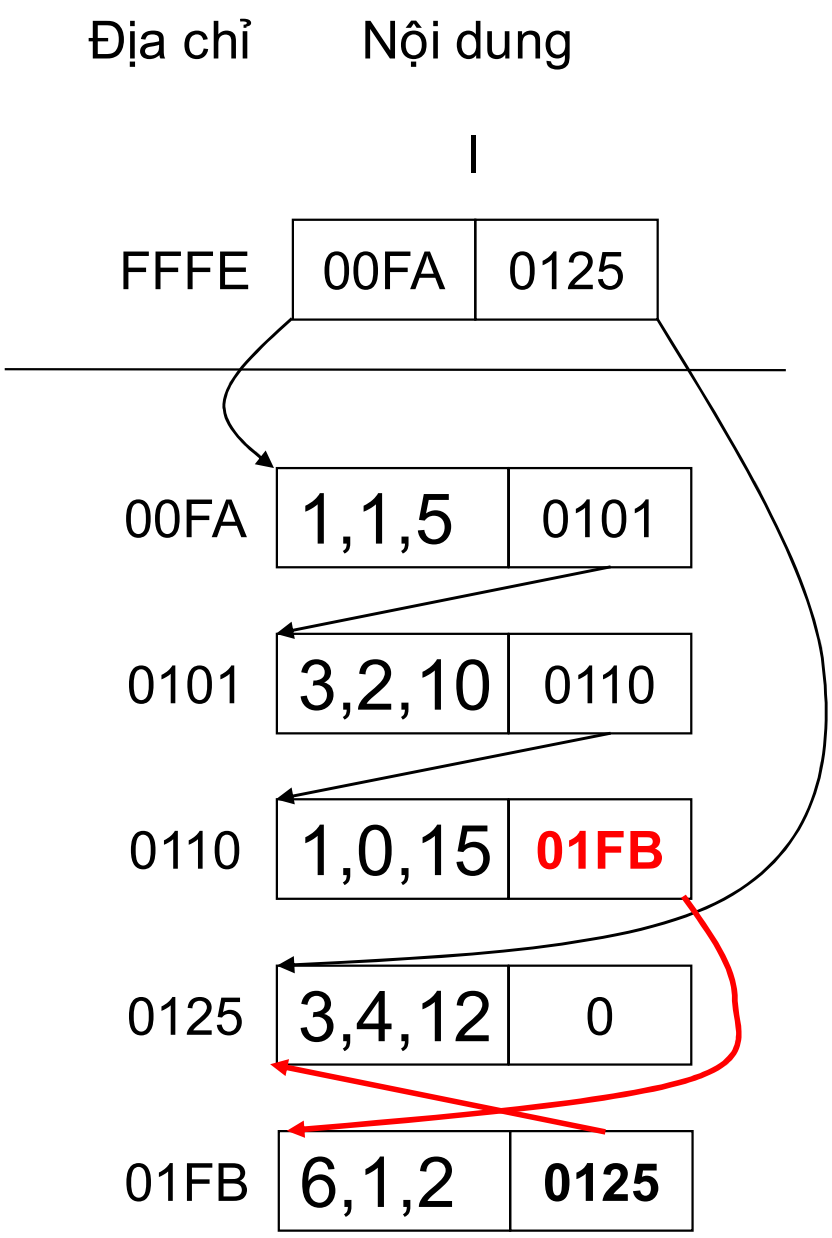
❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào sau phần tử q trong danh sách*

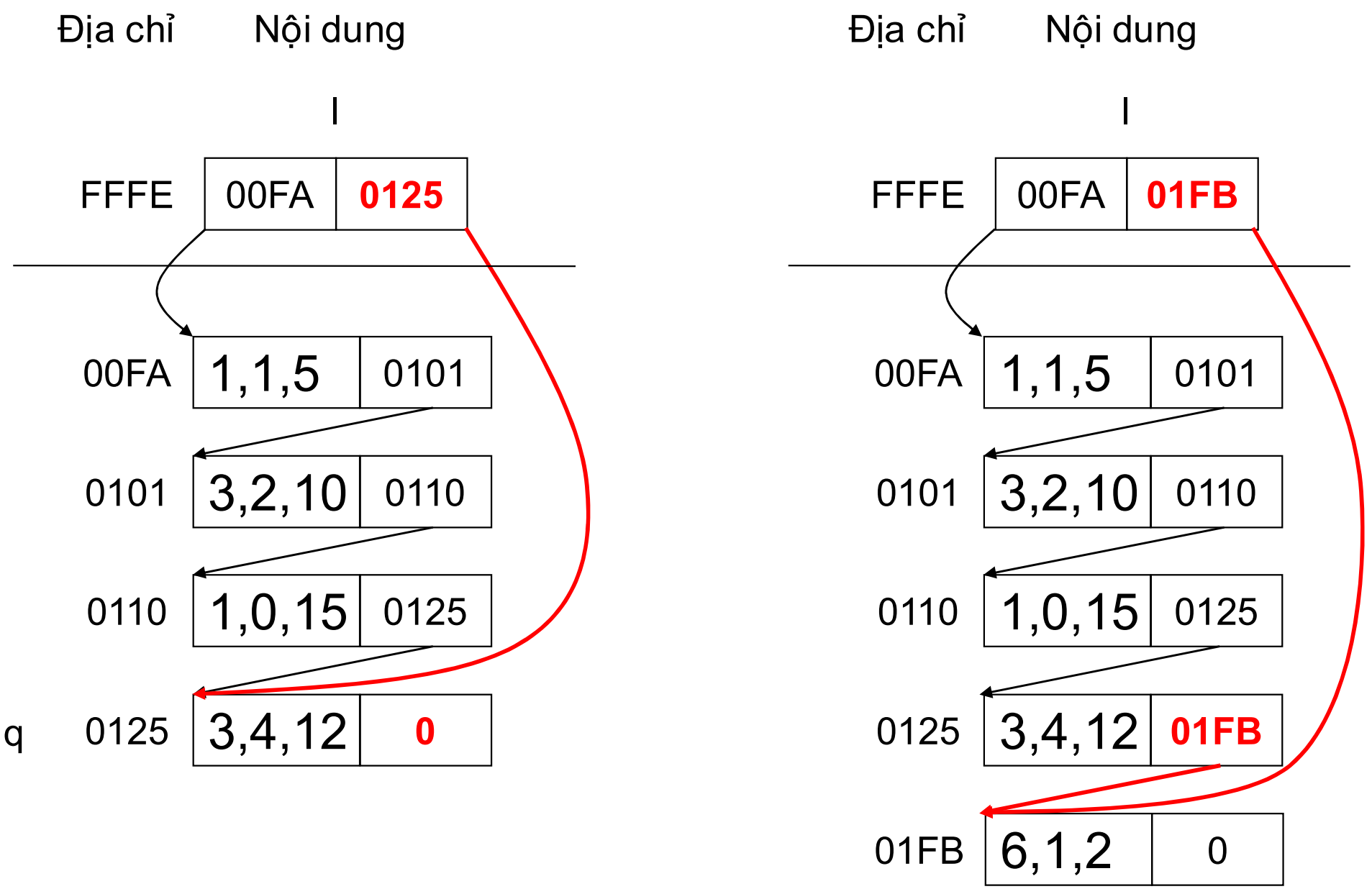
```
void AddAfter(TenDS &l, Node *p, Node *q) {
    if (q != NULL) {
        p->pNext = q->pNext; q->pNext = p;
        if (l.pTail == q) l.pTail = p;
    }
    else
        AddFirst(l, p);
}
```

DANH SÁCH ĐƠN





DANH SÁCH ĐƠN



DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Duyệt danh sách**

  Được thực hiện tuần tự từ phần tử đầu danh sách đến phần tử cuối danh sách. Duyệt danh sách nhằm mục đích đếm số phần tử, tìm phần tử thỏa điều kiện.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Duyệt danh sách**

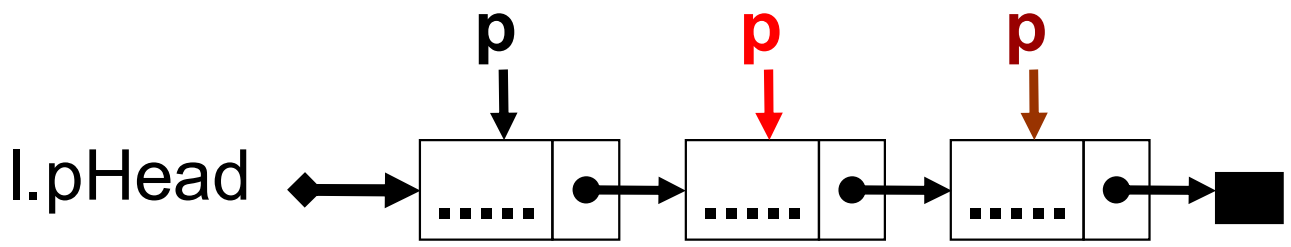
Nguyên tắc: Để duyệt danh sách l

B1) p ← l.pHead

B2) Nếu p = NULL qua B4

B3) Xử lý cho phần tử p, p ← p->pNext, qua B2.

B4) Kết thúc.



DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Duyệt danh sách: Tìm phần tử có trường info bằng x**

Thuật toán:

- Đầu vào: Danh sách l, giá trị x

- Đầu ra: phần tử p có giá trị x

B1) p ← l.pHead
B2) Nếu p = NULL qua B4
B3) Nếu p->info = x, qua B4,
    Ngược lại p ← p->pNext qua B2.
B4) Kết quả tìm là p.
B5) Kết thúc.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Duyệt danh sách: Tìm phần tử có trường info bằng x**

```
int Equal(TenDuLieu x, TenDuLieu y); // hàm so sánh
Node * Search(TenDS l, TenDuLieu x) {
    Node *p = l.pHead;
    while ((p != NULL) && (!Equal(p->info, x))
        p = p->pNext;
    return p;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

Bài tập 1: Viết chương trình cho phép nhập một danh sách hình tròn trong không gian 2 chiều cho tới khi nhập bán kính bằng 0. In ra màn hình các hình tròn có diện tích bằng s nhập từ bàn phím.

DANH SÁCH ĐƠN

```
struct Circle {
    double x, y, r;
};
struct CircleNode {
    Circle info;
    CircleNode *pNext;
};
struct CircleList {
    CircleNode *pHead, *pTail;
};
```

DANH SÁCH ĐƠN

```
void CreateList(CircleList &l) {
    l.pHead = NULL; l.pTail = NULL;
}
CircleNode* CreateNode(Circle x) {
    CircleNode *p = new CircleNode;
    if (p != NULL) {
        p->info = x;
        p->pNext = NULL;
    }
    return p;
}
```

DANH SÁCH ĐƠN

```
void AddLast(CircleList &l, CircleNode *p) {
    if (l.pHead == NULL) {
        l.pHead = p; l.pTail = p;
    } else {
        l.pTail->pNext = p; l.pTail = p;
    }
}
```

DANH SÁCH ĐƠN

```
int Compare(Circle x, double s) {
    double stmp = x.r * x.r * 3.14;
    if (stmp == s) return 0;
    if (stmp < s) return -1;
    return 1;
}
void Print(Circle x) {
    cout << '(' << x.x << ", " << x.y << "), " << x.r << ' ';
}
```

DANH SÁCH ĐƠN

```
void Browse(CircleList &l, double s) {
    CircleNode *p = l.pHead;
    while (p != NULL) {
        if (Compare(p->info, s) == 0)
            Print(p->info);
        p = p->pNext;
    }
}
```

DANH SÁCH ĐƠN

```
void InputList(CircleList &l) {
    CircleNode *p;
    Circle x;
    cin >> x.x >> x.y >> x.r;
    while (x.r > 0) {
        p = CreateNode(x);
        if (p == NULL)
            return;
        AddLast(l, p);
        cin >> x.x >> x.y >> x.r;
    }
}
```

DANH SÁCH ĐƠN

```
int main(int argc, char **argv) {
    CircleList list;
    double s;
    CreateList(list);
    InputList(list);
    cin >> s;
    Browse(list, s);
    return EXIT_SUCCESS;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

Bài tập 2: Với đề bài như Bài tập 1, viết hàm FilterList dùng để lọc các hình tròn có tọa độ tâm nằm trong góc phần tư thứ nhất của mặt phẳng tọa độ. Cho nguyên mẫu của hàm FilterList như sau:

void FilterList(CircleList in, CircleList &out);

DANH SÁCH ĐƠN

```
int Check(Circle x) {
    if ((x.x >= 0) && (x.y >= 0))
        return 1;
    return 0;
}
```

DANH SÁCH ĐƠN

```
void FilterList(CircleList in, CircleList &out) {
    CircleNode *p = in.pHead, *pTmp;
    CreateList(out);
    while (p != NULL) {
        if (Check(p->info)) {
            pTmp = CreateNode(p->info);
            if (pTmp == NULL) return;
            AddLast(out, pTmp);
        }
        p = p->pNext;
    }
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:** Xét các trường hợp sau:
  - Hủy phần tử đầu danh sách
  - Hủy phần tử ngay sau phần tử q trong danh sách
  - Hủy phần tử có khóa x

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử đầu danh sách*
- Đầu vào: Danh sách l
- Đầu ra: Danh sách l, dữ liệu x, kết quả thực hiện r

B1) Nếu l.pHead = NULL thì r ← 0 qua B4

B2) p ← l.pHead, l.pHead ← p->pNext, x ← p->info, r ← 1, giải phóng p.

B3) Nếu l.pHead = NULL thì l.pTail ← NULL

B4) Trả về r.

B5) Kết thúc.

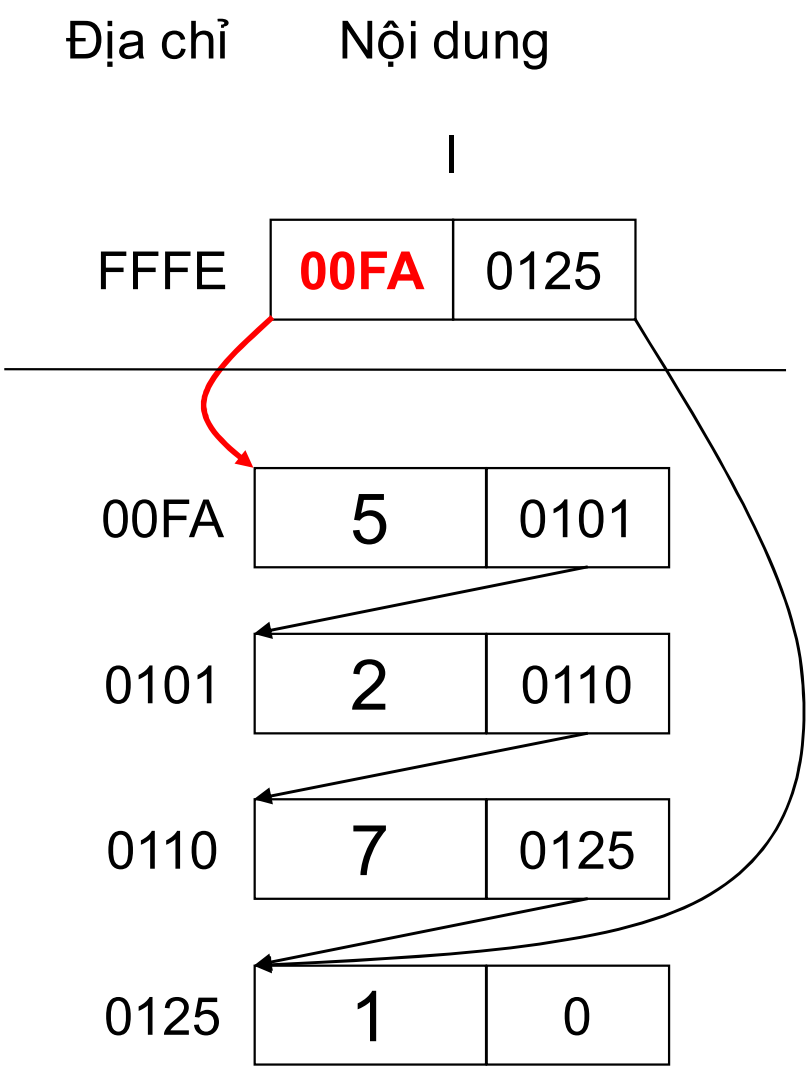
DANH SÁCH ĐƠN

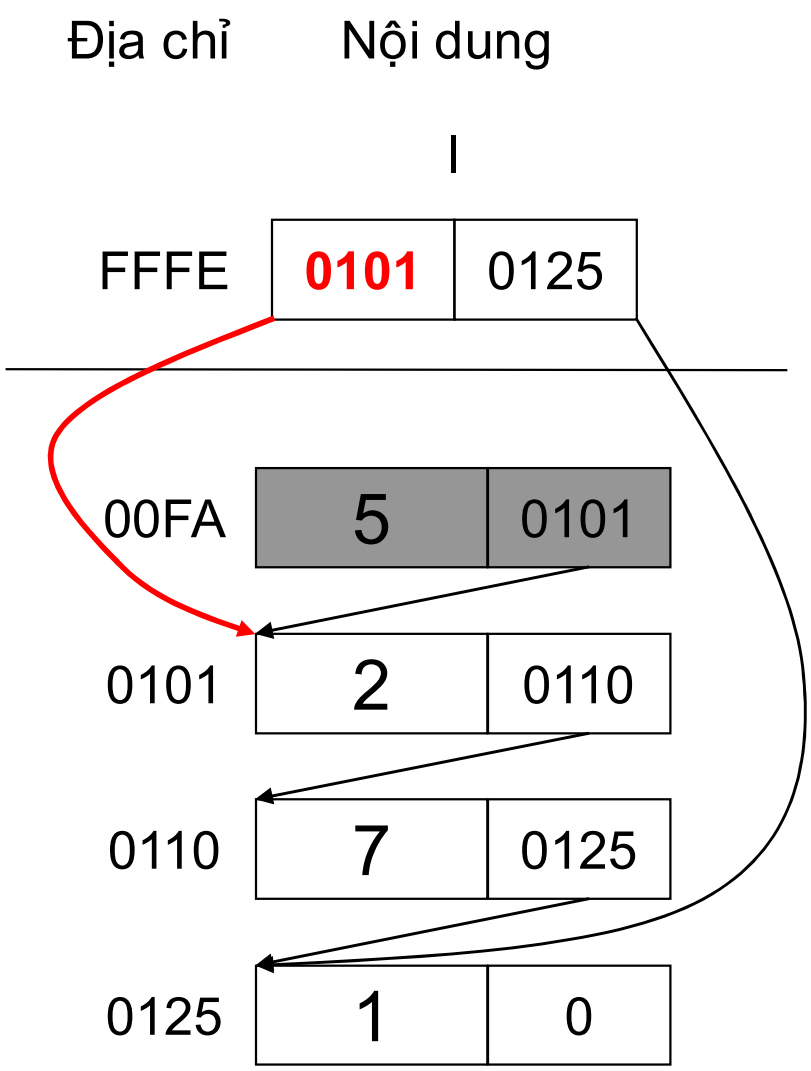
❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử đầu danh sách*

```
int RemoveFirst(TenDS &l, TenDulieu &x) {
    Node *p = l.pHead;  int r = 0;
    if (l.pHead != NULL) {
        x = p->info; l.pHead = p->pNext; delete p; r = 1;
        if (l.pHead == NULL) l.pTail = NULL;
    }
    return r;
}
```

DANH SÁCH ĐƠN





DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử ngay sau phần tử q trong danh sách*
- Đầu vào: Danh sách l, phần tử q
- Đầu ra: Danh sách l, dữ liệu x, kết quả thực hiện r

B1) Nếu q = NULL thì r ← 0, qua B5

B2) p ← q->pNext, nếu p = NULL thì r ← 0, qua B5

B3) Nếu p = l.pTail thì l.pTail ← q

B4) x ← p->info, q->pNext ← p->pNext, r←1, giải phóng p

B5) Trả về r.

B6) Kết thúc.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử ngay sau phần tử q trong danh sách*

```
int RemoveAfter(TenDS &l, Node *q, TenDulieu &x) {
    Node *p;  int r = 0;
    if (q != NULL) {   p = q->pNext;
        if (p != NULL) {   if (l.pTail == p) l.pTail = q;
            x = p->info; q->pNext = p->pNext; delete p; r = 1; }
    }
    return r;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử có khóa x*
- Đầu vào: Danh sách l, khóa x
- Đầu ra: Danh sách l, kết quả thực hiện r

B1) Tìm phần tử p có khóa x và q là phần tử trước p.

B2) Nếu p = NULL thì r ← 0, qua B4

B3) Nếu q = NULL thì r ← kết quả hủy phần tử đầu của l;
ngược lại thì r ← kết quả hủy phần tử ngay sau q.

B4) Trả về r.

B5) Kết thúc.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử có khóa x*

```
int Remove(TenDS &l, TenDulieu x) {
    Node *p = l.pHead, *q = NULL; int r = 0;
    while ((p != NULL) && (!Equal(p->info, x)))
    { q = p; p = p->pNext; }
    if (p != NULL)
        if (q == NULL) r = RemoveFirst(l,x);
        else r = RemoveAfter(l, q, x);
    return r; }
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy danh sách:**

- Đầu vào: Danh sách l.

- Đầu ra: Danh sách l rỗng.

B1) Nếu l.pHead = NULL qua B3

B2) p ← l.pHead, l.pHead ← p->pNext, giải phóng p, qua B1.

B3) l.pTail = NULL.

B4) Kết thúc.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Hủy danh sách:**

```
void RemoveList(TenDS &l) {
    Node *p;
    while (l.pHead != NULL)  {
        p = l.pHead; l.pHead = p->pNext; delete p;
    }
    l.pTail = NULL;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

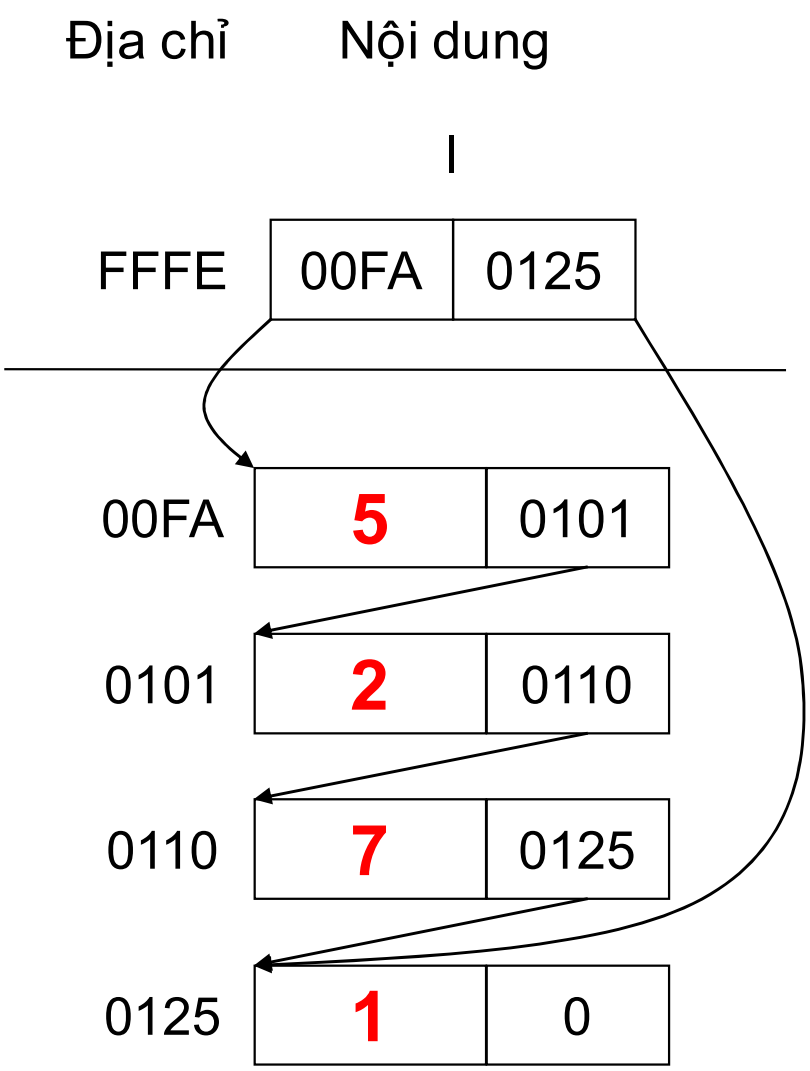
- **Sắp xếp danh sách:** Danh sách có thể được sắp xếp theo hai cách
- Hoán đổi thành phần info của các phần tử trong danh sách
- Thiết lập lại liên kết giữa các phần tử trong danh sách

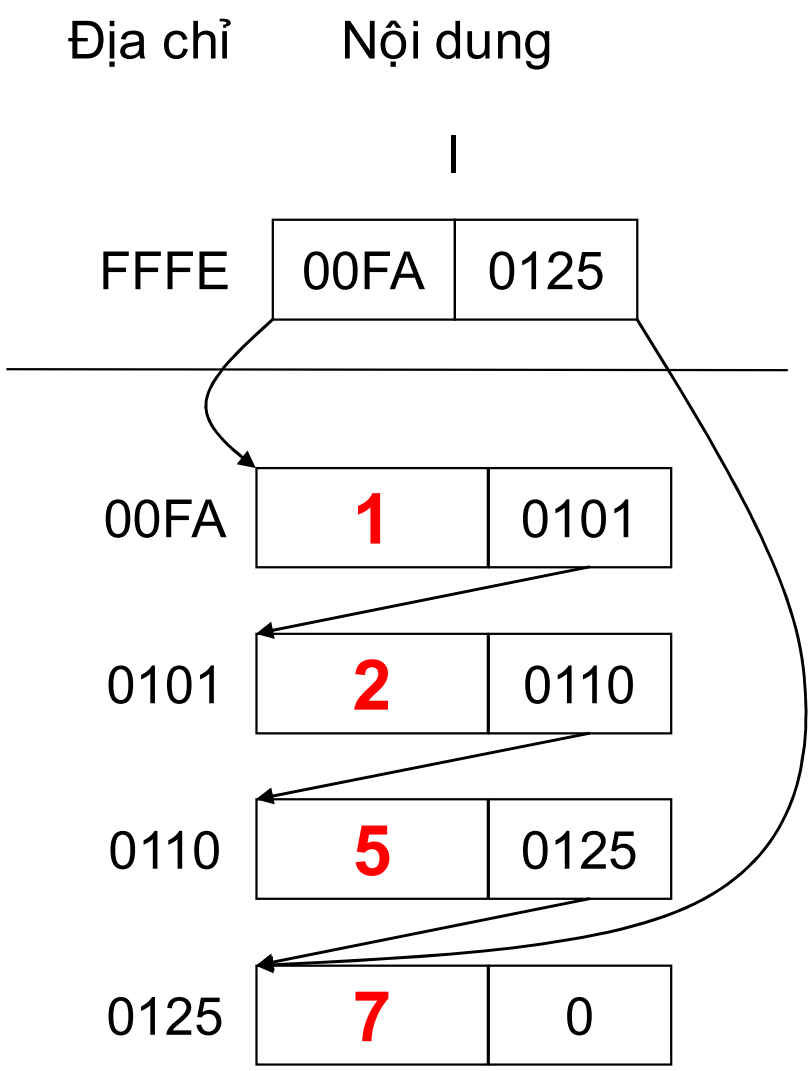
DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- Hoán đổi thành phần info của các phần tử trong danh sách:
  - Cài đặt đơn giản, tương tự sắp xếp mảng
  - Khi kích thước của info lớn, chi phí cho việc hoán đổi rất lớn dẫn đến thời gian sắp xếp chậm.

DANH SÁCH ĐƠN





DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- Hoán đổi thành phần info của các phần tử trong danh sách:

Ví dụ: Sắp xếp danh sách hình tròn trong bài tập 1 theo thứ tự bán kính tăng dần theo giải thuật Selection Sort.

DANH SÁCH ĐƠN

```
int CompareRadius(Circle x, Circle y) {
    if (x.r == y.r) return 0;
    if (x. r > y.r) return 1;
    return -1;
}
void Swap(Circle &x, Circle &y) {
    Circle tmp = x;
    x = y; y = tmp;
}
```

DANH SÁCH ĐƠN

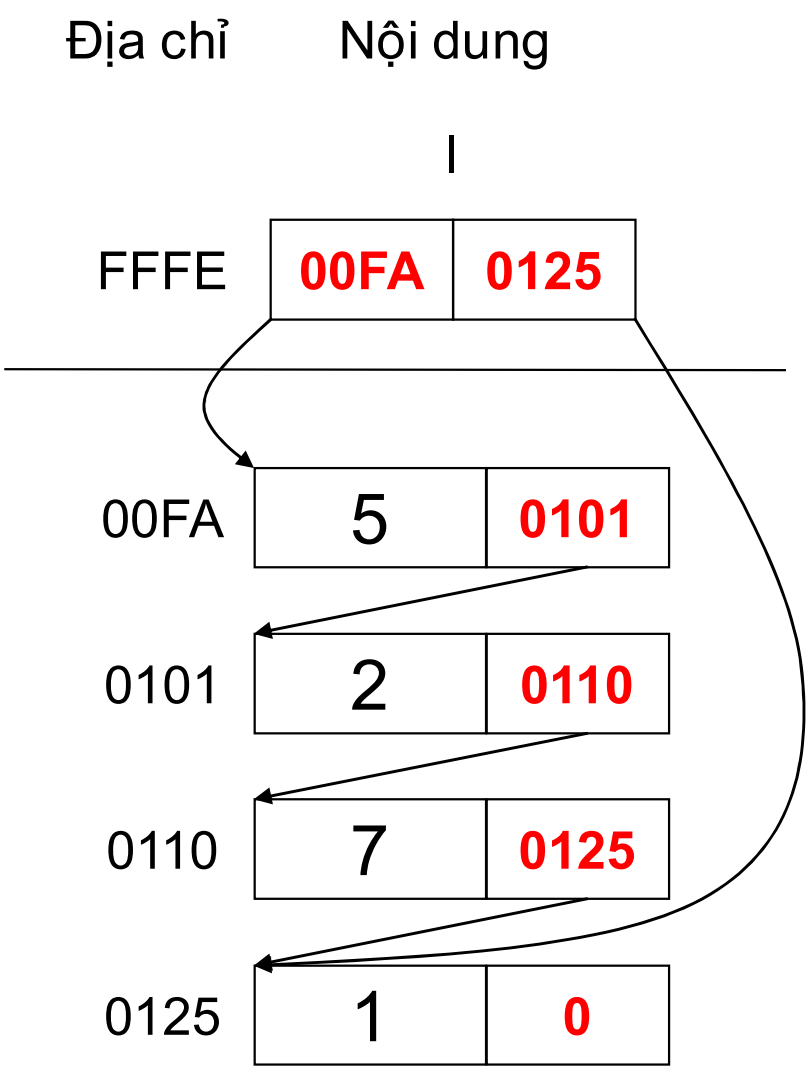
```
void SelectionSort(CircleList &l) {
    CircleNode *p, *q, *min;
    p = l.pHead;
    while (p != l.pTail) {
        min = p;     q = p->pNext;
        while (q != NULL) {
            if (CompareRadius(q->info, min->info) == -1)
                min = q;
            q=q->pNext;}
        Swap(min->info, p->info); p = p->pNext;
    }
}
```

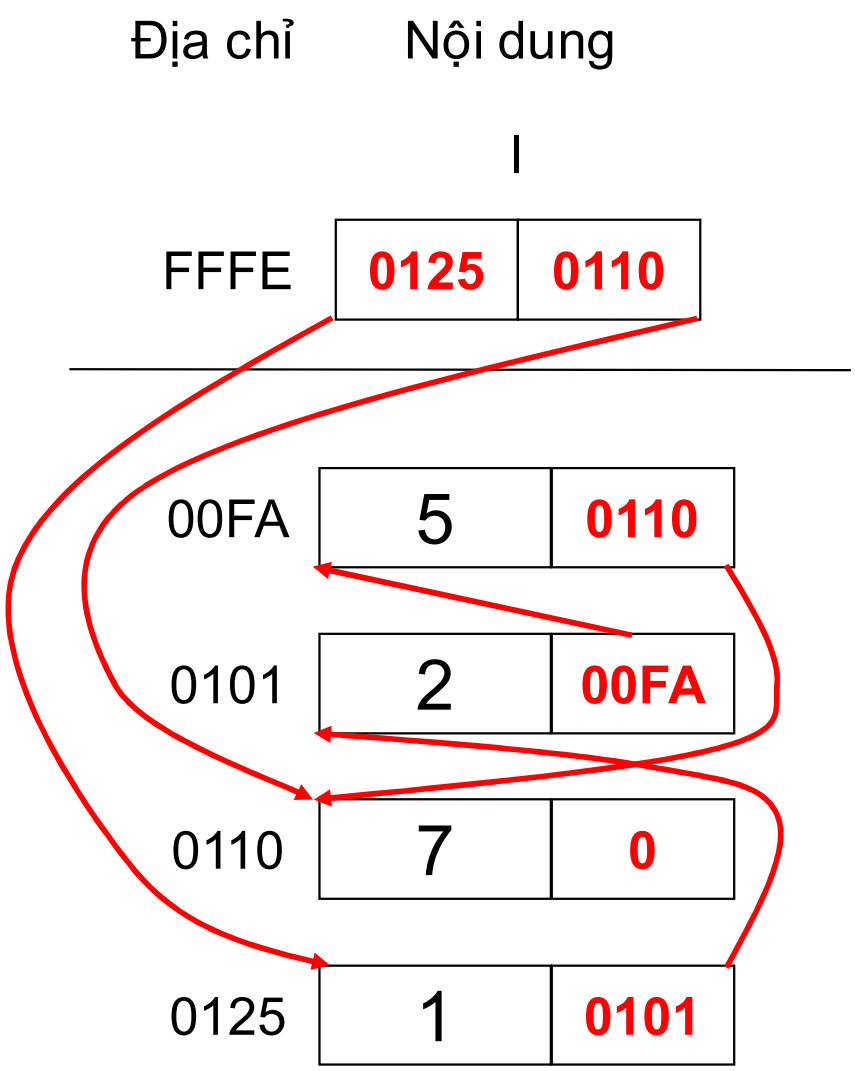
DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- Thiết lập lại liên kết giữa các phần tử trong danh sách:
  - Cài đặt phức tạp.
  - Chi phí hoán đổi liên kết cho một phần tử không chịu ảnh hưởng của trường info nên thời gian sắp xếp nhanh.

DANH SÁCH ĐƠN





DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- Thiết lập lại liên kết giữa các phần tử trong danh sách: rất thích hợp với các giải thuật sắp xếp
  - Quick Sort
  - Merge Sort
  - Radix Sort

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- *Quick Sort*
- Đầu vào: Danh sách L
- Đầu ra: Danh sách L đã có thứ tự

B1) Nếu L.pHead = L.pTail thì qua B8

B2) Chọn phần tử chốt X ←L.pHead,

L.pHead ← L.pHead->pNext

B3) Nếu L.pHead = NULL thì qua B6

B4) p ← L.pHead, L.pHead ← p->pNext, p->pNext ←NULL

DANH SÁCH ĐƠN

B5) Nếu p->info ≤ X->info thì thêm p vào danh sách L1; ngược lại thêm p vào danh sách L2

Qua B3

B6) Thực hiện Quick Sort cho L1 và L2

B7) Nối theo thứ tự L1, X, L2 thành L

B8) Kết thúc.

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**

- *Quick Sort*

```
int Compare(TenDulieu x, TenDulieu y);
// so sánh khóa: -1 nếu x < y, 0 nếu x = y, 1 nếu x > y
void QuickSort(TenDS &l) {
    Node *p, *X;
    TenDS l1, l2;
    if (l.pHead == l.pTail) return;
    CreateList(l1); CreateList(l2);
    X = l.pHead; l.pHead = X->pNext;
```

DANH SÁCH ĐƠN

```
while (l.pHead != NULL) {
    p = l.pHead; l.pHead = p->pNext; p->pNext = NULL;
    if (Compare(p->info, X->info) <= 0)
        AddLast(l1, p);
    else
        AddLast(l2, p);
}
l.pTail = NULL;
QuickSort(l1); QuickSort(l2);
```

DANH SÁCH ĐƠN

```
    if (l1.pHead != NULL) {
        l.pHead = l1.pHead; l1.pTail->pNext = X;
    }
    else
        l.pHead = X;
    X->pNext = l2.pHead;
    if (l2.pHead != NULL)
        l.pTail = l2.pTail;
    else
        l.pTail = X;
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- *Merge Sort*
- Đầu vào: Danh sách L
- Đầu ra: Danh sách L đã có thứ tự

B1) Nếu L.pHead = L.pTail thì qua B5

B2) Phân phối luân phiên từng run cho hai danh sách L1 và L2

B3) Thực hiện Merge Sort cho L1, L2

B4) Trộn L1, L2 thành L

B5) Kết thúc

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**

- *Merge Sort*

```
int Compare(TenDulieu x, TenDulieu y);
// so sánh khóa: -1 nếu x < y, 0 nếu x = y, 1 nếu x > y
void MergeSort(TenDS &l) {
    TenDS l1, l2;
    Node *p, *q;
    int n = 0;
    if (l.pHead == l.pTail) return;
    CreateList(l1); CreateList(l2);
```

DANH SÁCH ĐƠN

```
p = l.pHead; l.pHead = p->pNext; p->pNext = NULL;
AddLast(l1, p);
while (l.pHead != NULL) {
    q = l.pHead; l.pHead = q->pNext; q->pNext = NULL;
    if (Compare(p->info, q->info) <= 0) {
        if (n == 0) AddLast(l1, q); else AddLast(l2, q);
    } else {
        if (n == 0) { n = 1; AddLast(l2, q); }
        else { n = 0; AddLast(l1, q); }
    }
    p = q;
}
```

DANH SÁCH ĐƠN

```
l.pTail = NULL;
if (l2.pHead != NULL) { // l2 rỗng thì l1 đã có thứ tự
    MergeSort(l1); MergeSort(l2);
}
//Trộn danh sách
while ((l1.pHead != NULL) && (l2.pHead != NULL)) {
    if (Compare(l1.pHead->info, l2.pHead->info) <=0) {
        p = l1.pHead; l1.pHead = p->pNext;
        p->pNext = NULL;
    } else {
        p = l2.pHead; l2.pHead = p->pNext;
        p->pNext = NULL;
    }
    AddLast(l, p);
}
```

DANH SÁCH ĐƠN

```
    if (l1.pHead != NULL) {
        if (l.pHead == NULL) {
            l.pHead = l1.pHead; l.pTail = l1.pTail;
        } else {
            l.pTail->pNext = l1.pHead; l.pTail = l1.pTail;
        }
    }
    if (l2.pHead != NULL) {
        l.pTail->pNext = l2.pHead; l.pTail = l2.pTail;
    }
}
```

DANH SÁCH ĐƠN

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- *Radix Sort* (Sinh viên tự tìm hiểu)