Data Structures and Algorithms
(CS221)

List

- **Attendance?**

  - Active Attendance
  - **Dead Bodies.**
  - **Active Minds**
  - Mobiles in hands -> Mark as absent
  - 80% mandatory

List

The **list** *data type* is a collection type that stores ordered, non-unique elements; that is, it allows duplicate element values.

Array Based List

Linked List

List: Traverse through the list

Visit each element of the list

e.g., print each element of the list on the screen

List data	2	4	6	8	10	12	14	16

```
for (int i = 0; i < N; i++)
cout<<a[i];
```

Complexity : O(*N*)

List: Search for an element in the list

Search for a specific element in the list

e.g., search and return the index of 10, if found in the list

List data	2	4	6	8	10	12	14	16

```
SearchElement = 10;
 for (int i = 0; i < N; i++){
          if (a[i] = = SearchElement){
               index = i;
               return index;
           }
 }
index = -1;
cout<< "Element not found";
return index;
```

Complexity : O(*N*)

List: Read/Update an Element

- **Read/update an element in the list**
  - at the beginning,
  - at the end
  - anywhere
  - Example
    - Read/print *i*th element (*i*=3)
    - update *j*th element (*j*=4)

Read *i*th element

i

2	4	6	8	10	12	14	16

$i = 3;\ cout << a[i];$ → **8**

List: Read/Update an Element

Update *i*th element

				j			
2	4	6	8	10	12	14	16

2	4	6	8	15	12	14	16

$j = 4;\ a[j] = 15;$

Complexity : O(1)

List: Add Element

Add at the beginning:

Step 1 : Move all the elements towards right, creating space for one more element in the beginning

Step 2 : Add the new element at the beginning

Step 3 : Increment the size of the array by one

```
 N = N + 1;
    for (int i = N-1; i > 0; i--
)
          a[i] = a[i-1];
      a[0] = NewElement; // (=
1)
```

Complexity : O(*N*)

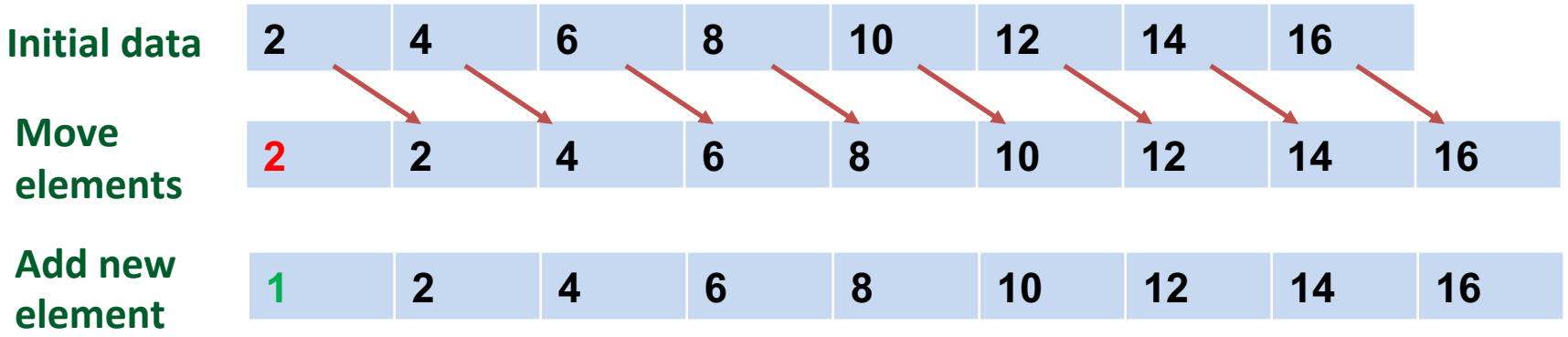
List: Add Element

Add at the beginning:

Step 1 : Move all the elements towards right, creating space for one more element in the beginning

Step 2 : Add the new element at the beginning

Step 3 : Increment the size of the array by one



List: Add Element

Add at the end:

Step 1 : Add the new element at end

Step 2 : Increment the size of the array by one

$N = N + 1;$
$a[N\text{-}1] = NewElement$; // (= 17)

Complexity : O(*1*)

List: Add Element

Add anywhere in the list (e.g., at the *j*th location)

Step 1 : Move all the elements, starting from the index *j*, towards right,

Step 2 : Add the new element at the index *j*

Step 3 : Increment the size of the array by one

```
N = N +1;
    for (int i = N-1; i > j; i-
-)
            a[i] = a[i-1];
            a[j] = NewElement ;
```

Complexity : O(*N*)

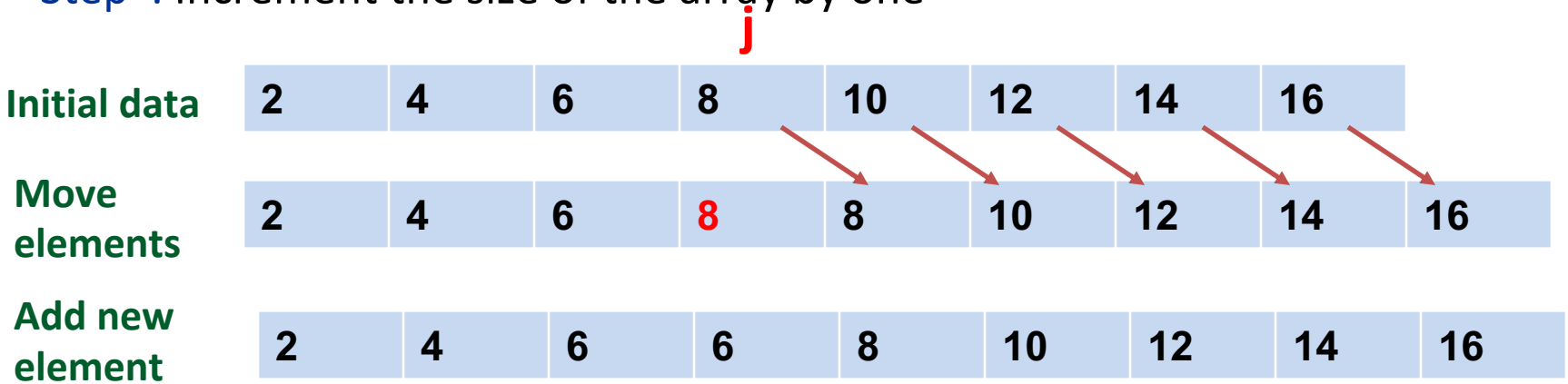
List: Add Element

Add anywhere in the list (e.g., at the jth location)

Step 1 : Move all the elements, starting from the index *j*, towards right,

Step 2 : Add the new element at the index *j*

Step : Increment the size of the array by one



List: Delete Element

Delete the beginning element:

Step 1 : Move all the elements towards left, overriding the first element
Step 2 : Decrement the size of the array by one

```
for (int i = 0; i < N-1; i++)
            a[i] = a[i +1];
            N = N - 1;
```

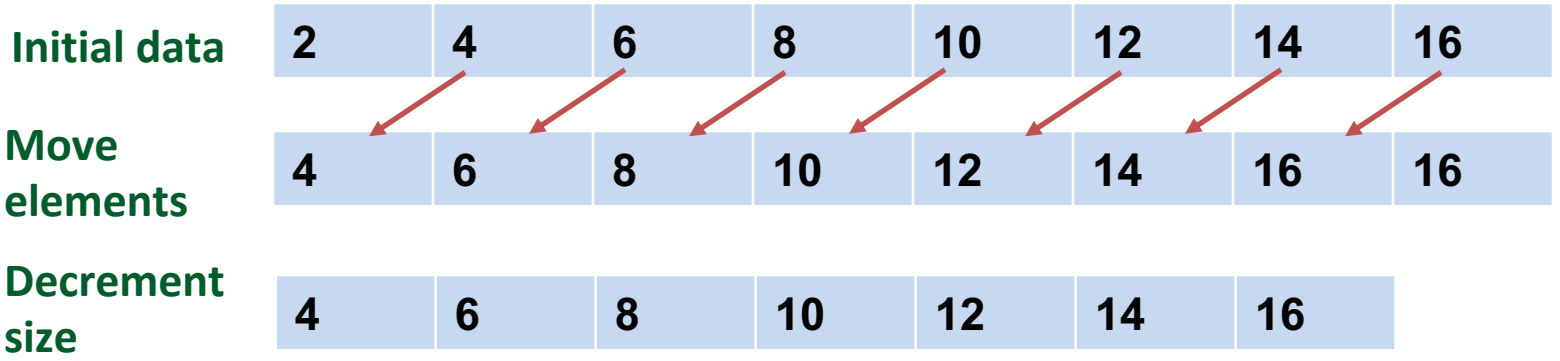
Complexity : O(*N*)

List: Delete Element

Delete the beginning element:

Step 1 : Move all the elements towards left, overriding the first element

Step 2 : Decrement the size of the array by one




```
for (int i = 0; i < N-1; i++)

a[i] = a[i +1];
N = N - 1;
```

Complexity : O(N)

List: Delete Element

Add at the end:

Step 1 : Delete the element from the end

Step 2 : Decrement the size of the array by one

Initial data

2	4	6	8	10	12	14	16

Delete element and decrement size

2	4	6	8	10	12	14

Complexity : O(1)

$N = N - 1;$

List: Delete Element

Delete from anywhere in the list (e.g., at the jth location)

Step 1 : Move all the elements, after the index *j*, towards left
Step 2 : Decrement the size of the array by one

```
for (int i = j; i < N-1; i++)
        a[i] = a[i+1];
        N = N -1;
```

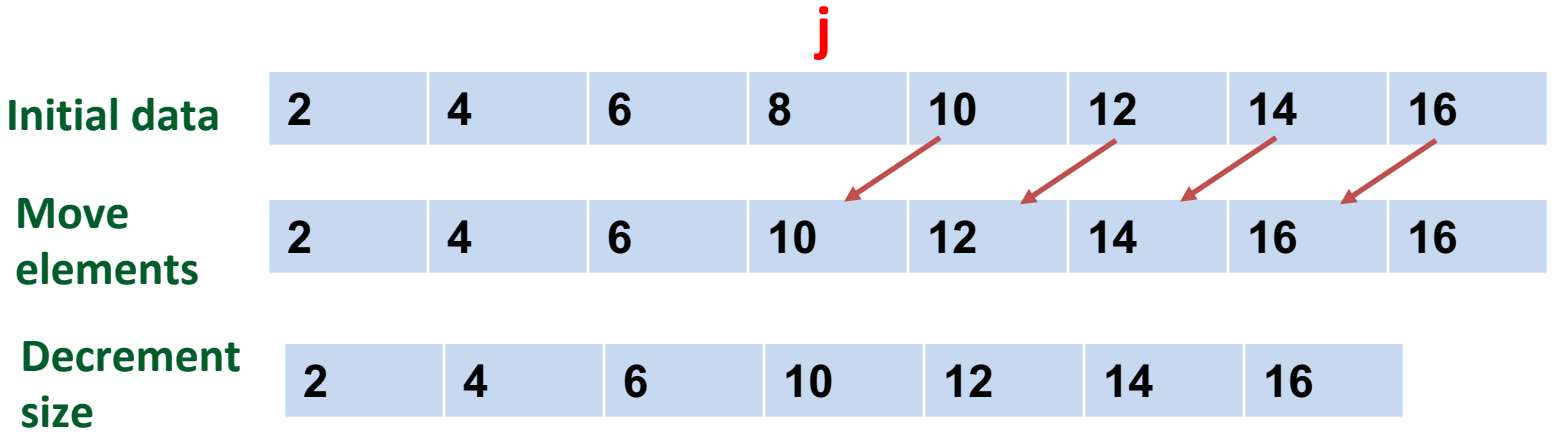
Complexity : O(*N*)

List: Delete Element

Delete from anywhere in the list (e.g., at the jth location)

Step 1 : Move all the elements, after the index *j*, towards left

Step 2 : Decrement the size of the array by one




```
for (int i = j; i < N-1; i++)
        a[i] = a[i+1];
    N = N -1;
```

Complexity : O(N)

List: Search and Update a Specific Element

```
void searchAndUpdate(int a[], int N, int target, int newValue)
{
    for (int i = 0; i < N; i++) {
        if (a[i] == target) {
            a[i] = newValue;
            cout << "Updated element " << target << " to " <<
newValue << " at index " << i << endl;
            return;
        }
    }
    cout << "Element " << target << " not found!" << endl;
}
```

Complexity : O(*N*)

List: Insert After a Specific Element

Steps

1. **Search** for the element (AfterElement).
2. **Shift elements right** from j+1.
3. **Insert new element** at j+1.
4. **Increment N**

Complexity : O(*N*)

List: Insert After a Specific Element

```
void insertAfter(int a[], int N, int
AfterElement, int newElement) {
    int j = -1;

for (int i = 0; i < N; i++) {
        if (a[i] == AfterElement) {
            j = i;
            break;
        }
    }

    if (j == -1) {
        cout << "Element " <<
AfterElement << " not found!" << endl;
        return;
    }
```

```
for (int i = N; i > j + 1; i--)
        a[i] = a[i - 1];

    a[j + 1] = newElement;
    N++;

     cout << "Inserted " << newElement
 << " after " << AfterElement << endl;
}
```

List: Insert Before a Specific Element

Steps

1. **Search** for the element (BeforeElement).
2. **Shift elements right** from j.
3. **Insert new element** at j.
4. **Increment N**

List: Insert Before a Specific Element

```
void insertBefore(int a[], int &N, int
BeforeElement, int newElement) {
    int j = -1;


    for (int i = 0; i < N; i++) {
        if (a[i] == BeforeElement) {
            j = i;
            break;
        }
    }

    if (j == -1) {
        cout << "Element " <<
BeforeElement << " not found!" << endl;
        return;
    }
```

```
    for (int i = N; i > j; i--)
        a[i] = a[i - 1];

    a[j] = newElement;
    N++;
    cout << "Inserted " << newElement << "
before " << BeforeElement << endl;
}
```

List implemented as an dynamic array

The static declaration of an array requires the size of the array to be known in advance

What if the actual size of the list exceeds its expected size?

Solution?

- Dynamic array
- Linked List

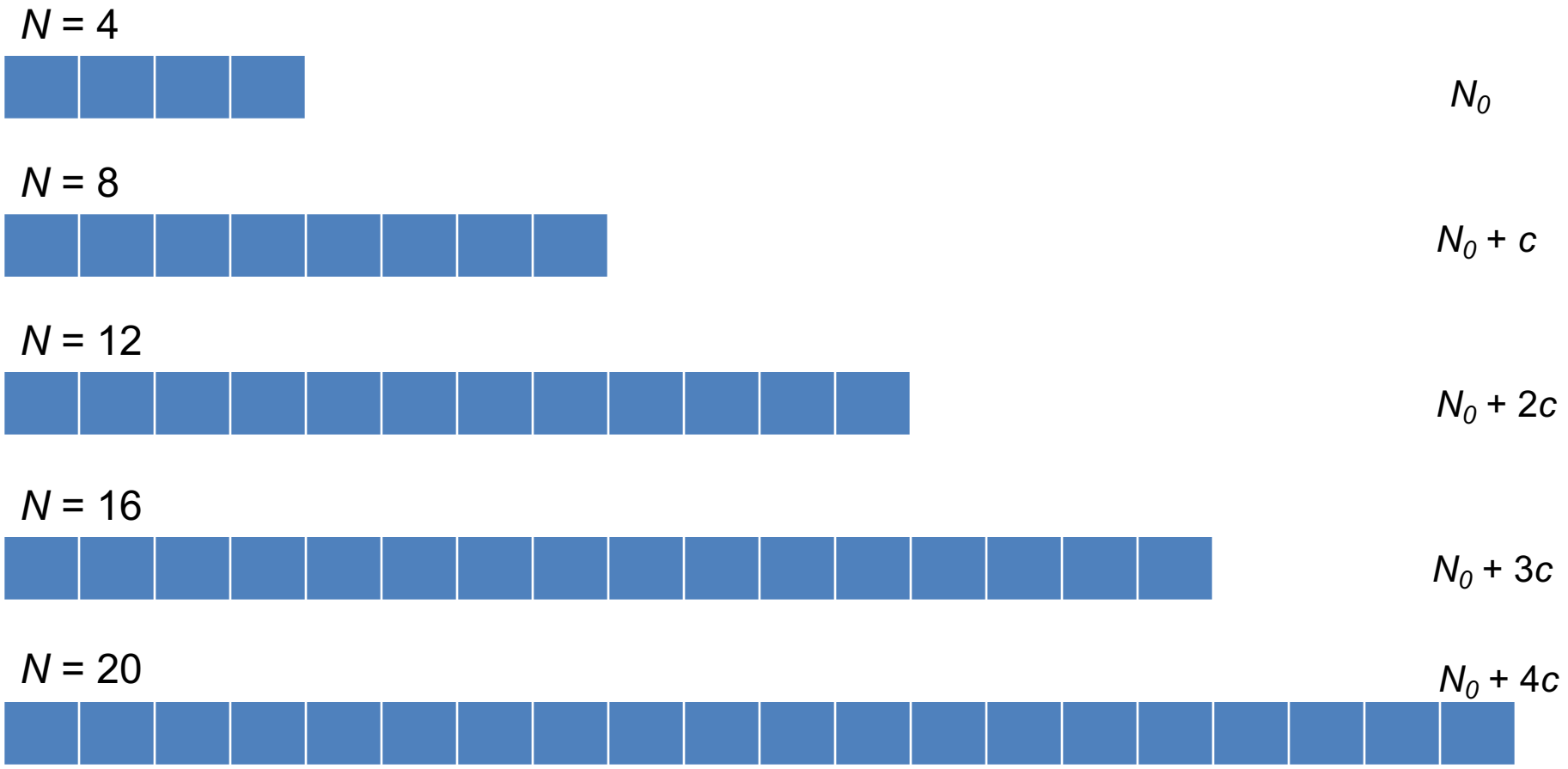
Array growth strategy

Increase the size by a constant (tight strategy)
$N = N + c$

Double the size of the array
$N = 2*N$

Increase the size by a constant

- Increment the size by a constant number c, each time the size needs to be re-adjusted
- Copy the contents of the previous list into the new list



Add Element Increase the size by a constant

Add at the end:

Step 1 : Increment the size of the array by a constant
Step 2 : Copy the old list into the new list
Step 3 : Rename the new list as the original list
Step 4 : Add the new element at end

```
if index == N  {  // if the current index exceeds the size of the list
    N = N + c;      // increase the size of the list by c
    int *tempA =  new int [N];  // allocate memory for the new list
    for (int i = 0; i < N - c; i++)
        tempA[i] = A[i];  // copy the old list into the new list
    delete [] A;
    A = tempA;              // rename the new list as the original list
    }
    A[index]= NewElement;
    Index ++;
```

Time Complexity??

Double The Size of the Array

- Increment the size twofold each time the size needs to be re-adjusted
- Copy the contents of the previous list into the new list

$N = 4$

N_0

$N = 8$

$2N_0$

$N = 16$

$4N_0$

.
.
$N_0 \times 2^k$

Dynamic List: Add Element
Double The Size of the Array

Add at the end:

Step 1 : Double the size of the array
Step 2 : Copy the old list into the new list
Step 3 : Rename the new list as the original list
Step 4 : Add the new element at end

```
if index == N  {  // if the current index exceeds the size of the list
    N = N * 2;       // double the size of the list
    int *tempA =  new int [N];  // allocate memory for the new list
    for (int i = 0; i < N / 2; i++)
        tempA[i] = A[i];  // copy the old list into the new list
    A = tempA;            // rename the new list as the original list
    }
    A[index]= NewElement;
    Index ++;
```

Time Complexity??

Questions?

zahmaad.github.io