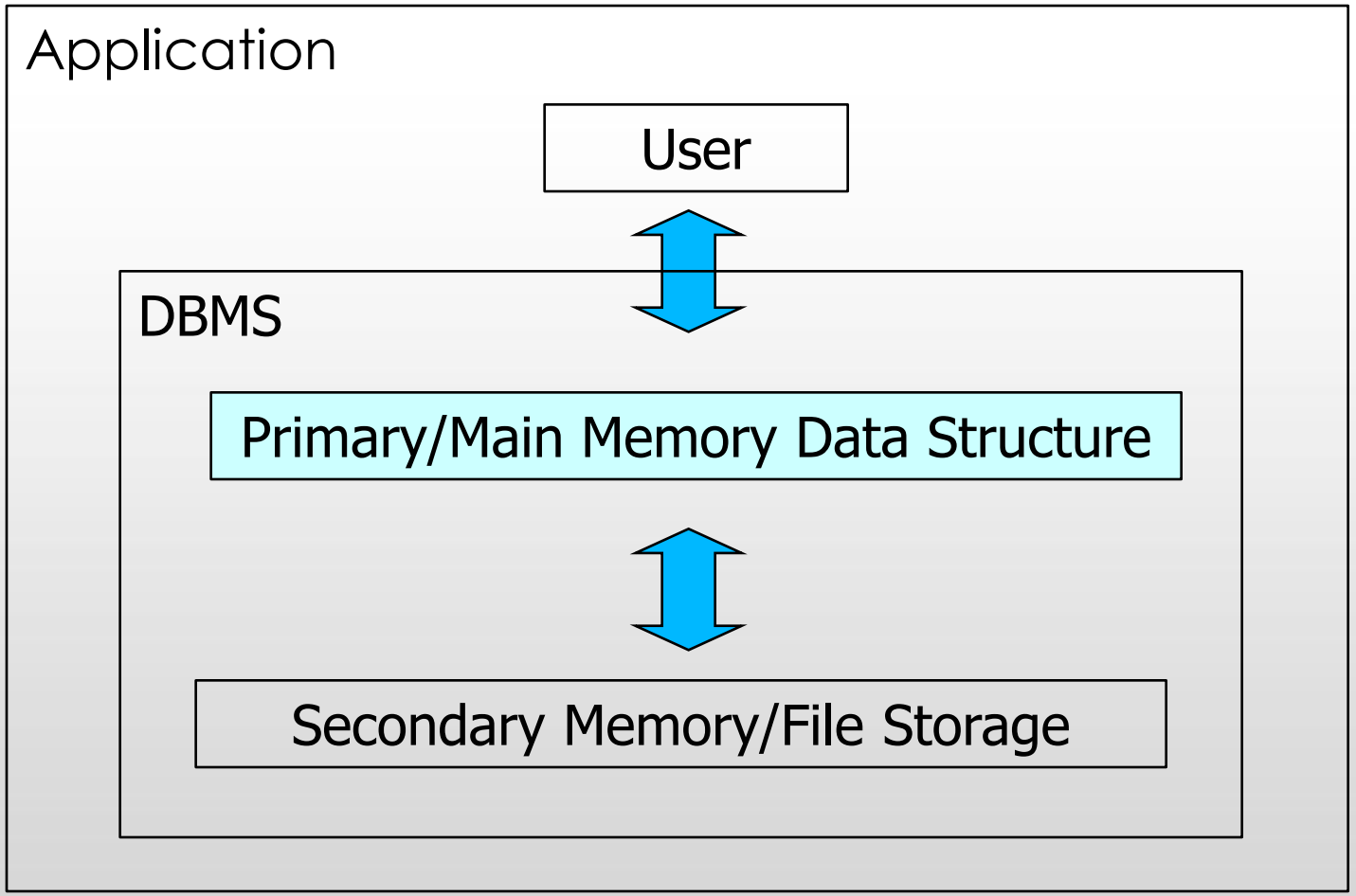
Data Structures

Overview

Overview


Application
User
DBMS
Primary/Main Memory Data Structure
Secondary Memory/File Storage

Definitions

- Data Type – set of values that a variable may assume (e.g. boolean, integer, double, float etc.)

Definitions

- Data Type – set of values that a variable may assume (e.g. boolean, integer, double, float etc.)

- Abstract Data Type (ADT) – a mathematical model, together with various operations defined on the model (e.g. Stack, Queue, BST, AVL, Binary Heap, etc.)

Definitions

- Data Structure – collection of variables, possibly of several different data types, connected in various ways (e.g. Array, Record, Singly-linked List, Doubly-linked List, Adjacency Matrix, Adjacency List etc.)

ADT vs. Data Structure

ADT

- Stack and Queue
- Graph

Data Structure

- Array or List Implementation
- Adjacency Matrix or List

1. The List ADT

1.1 Arrays

The List ADT

Definition:

- A sequence of zero or more elements of the same type.
- denoted by: $\mathbf{a_1, a_2, a_3, \ldots, a_n}$

  where n = size of the list and
  i = position of a_i

The List ADT

Operations:

- **insert** (add an element into the list)
- **delete** (remove an element from the list)
- **find** (returns the position of the first occurrence of an element)
- **find kth** (returns the element in the kth position)
- **next**, **previous** (takes as argument a position and returns the position of the successor and predecessor, respectively)
- **print list** (prints all the elements in the list)
- **make null** (delete all elements from the list)

Array Implementation

- Elements are contiguously stored in the main memory
- Fast access to an element
- Size is either fixed at compile time (static array) or at run-time (dynamic array)

```
int A1[5]  vs
int *A2 = (int *)malloc(5 * sizeof(int));
```

Array Implementation

- Indexing
  - first index: zero-based, one-based, n-based
  - bounds checking
  - associative arrays (use of non-integer indices)
  - multi-dimensional arrays

1.1 Arrays

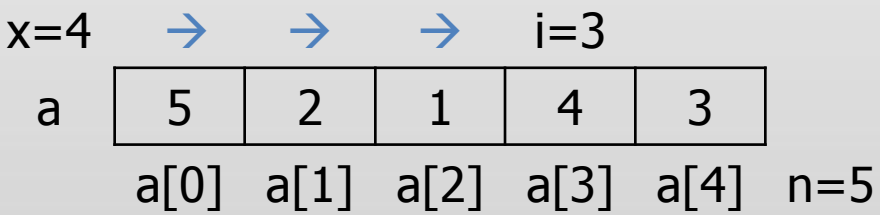
1.1.1 Searching

Linear Search

Linear Search - list is not necessarily sorted

```
int linear_search(int a[], int n, int x){
    int i;
    for(i=0;i<n;i++)
        if (a[i]==x) break; //found x
    return(i<n); //if x is found i<n else i=n
}
```

Case 1: x is in the list, i<n

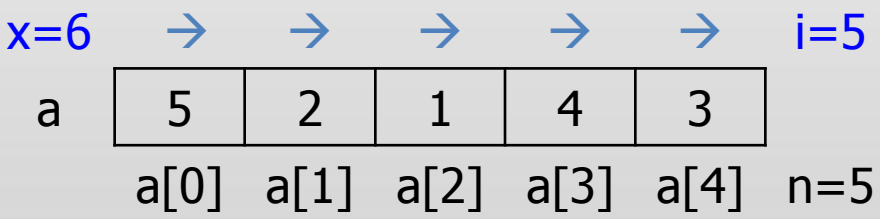


Linear Search

Linear Search - list is not necessarily sorted

```
int linear_search(int a[], int n, int x){
    int i;
    for(i=0;i<n;i++)
        if (a[i]==x) break; //found x
    return(i<n); //if x is found i<n else i=n
}
```

Case 2: x is not in the list, i=n



Binary Search

- Binary Search - list is sorted

```
int binary_search(int a[], int n, int x){
    int lower,upper, middle;
    lower = 0;
    upper = n-1;
    while(lower <= upper){
        middle = (lower + upper)/2;
        if (x > a[middle]) lower = middle+1;
        else if (x < a[middle] upper = middle - 1;
        else return(1);
    }
    return(0);
}
```

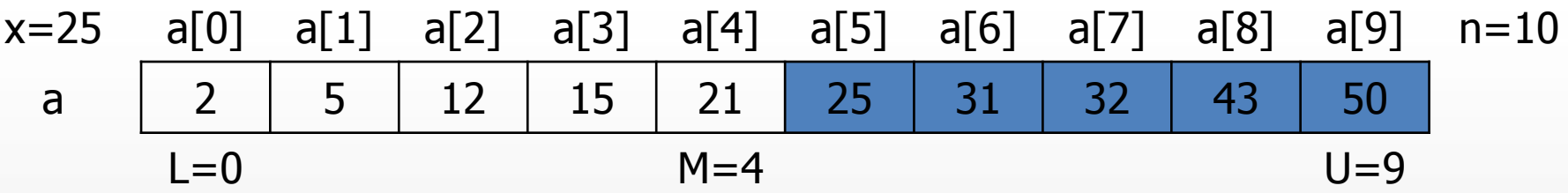
Binary Search

x=25

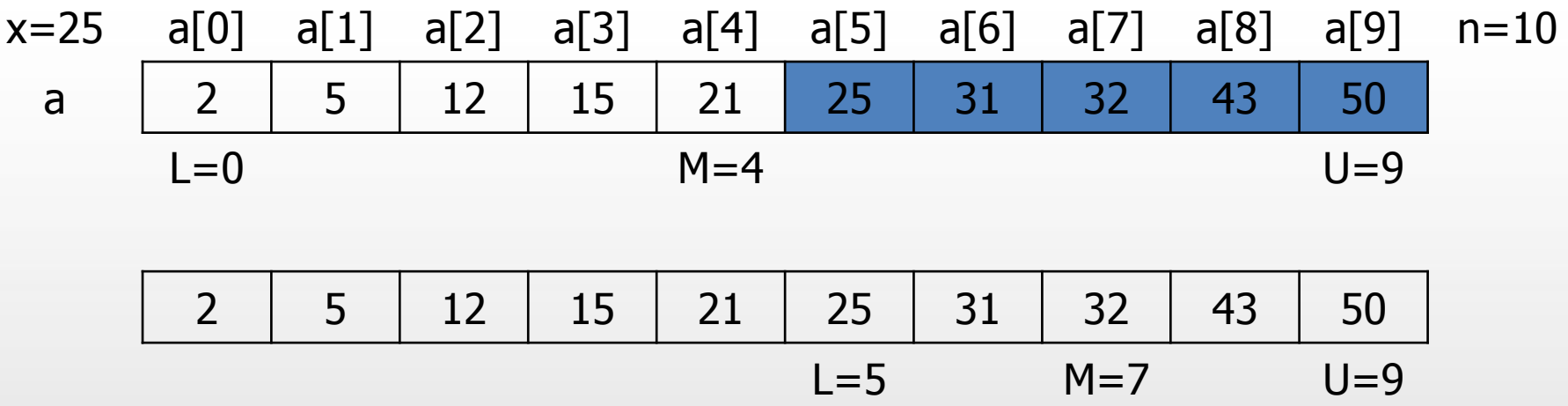
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
a	2	5	12	15	21	25	31	32	43	50
	L=0				M=4					U=9

n=10

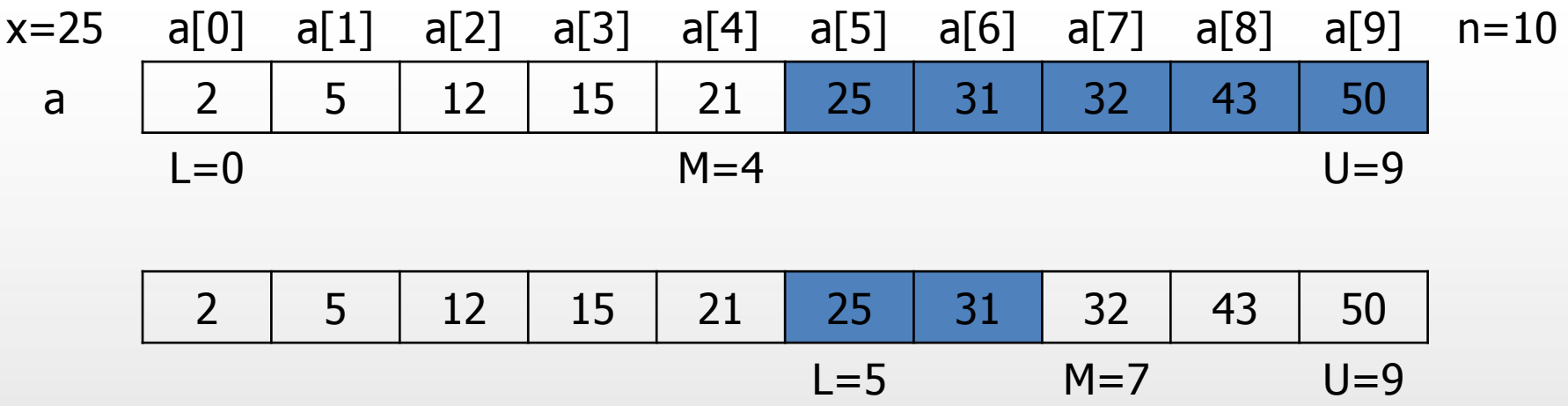
Binary Search



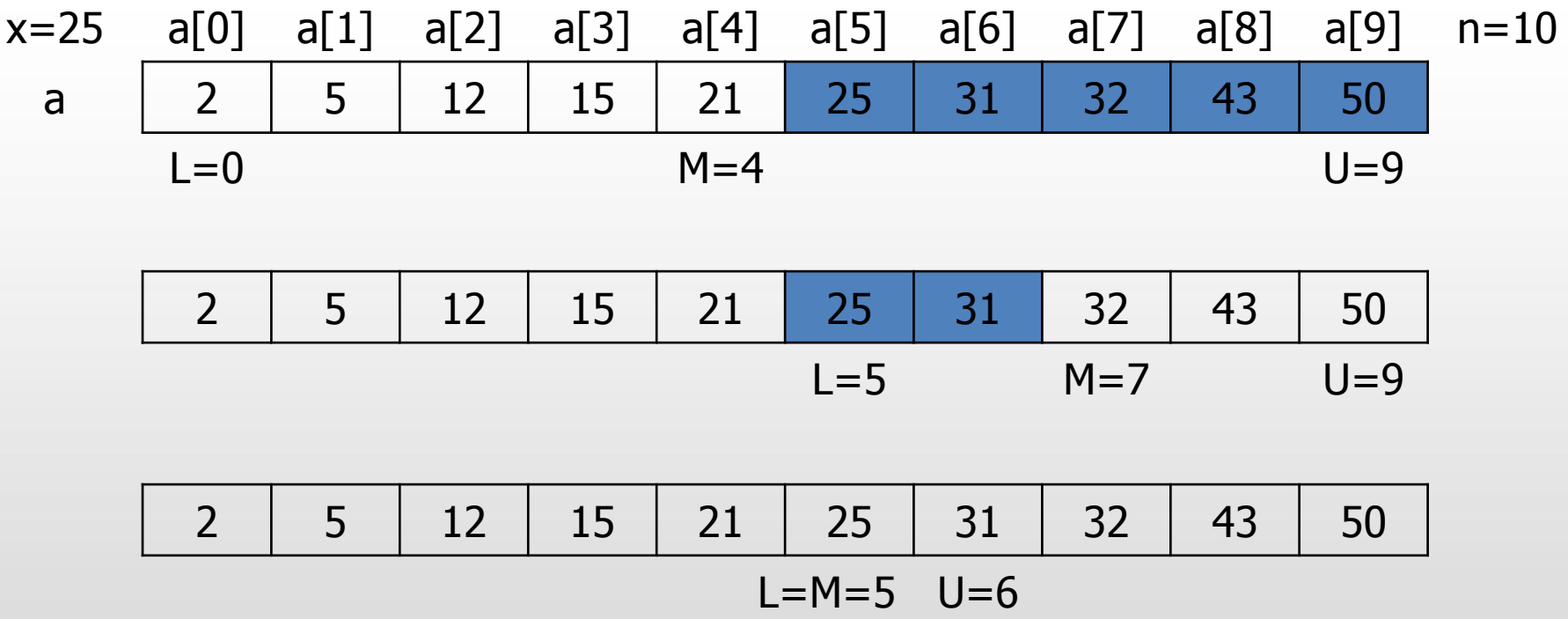
Binary Search



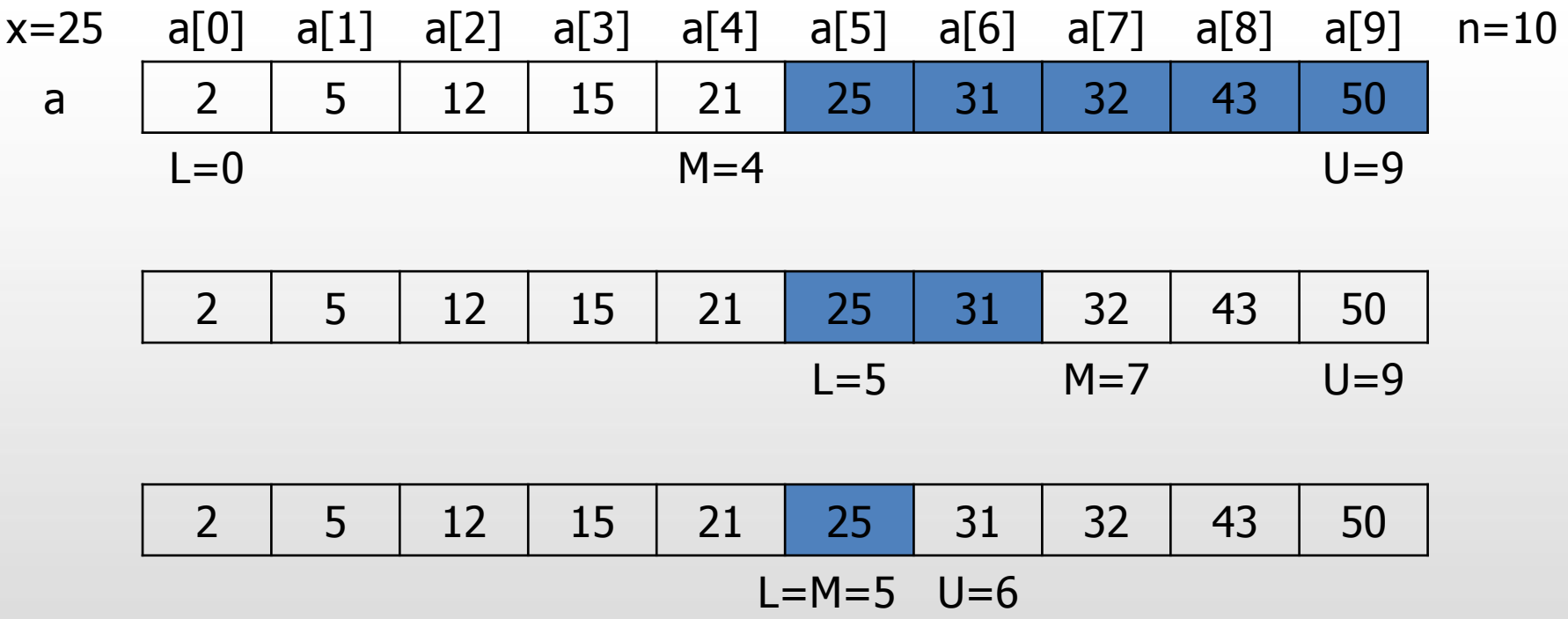
Binary Search



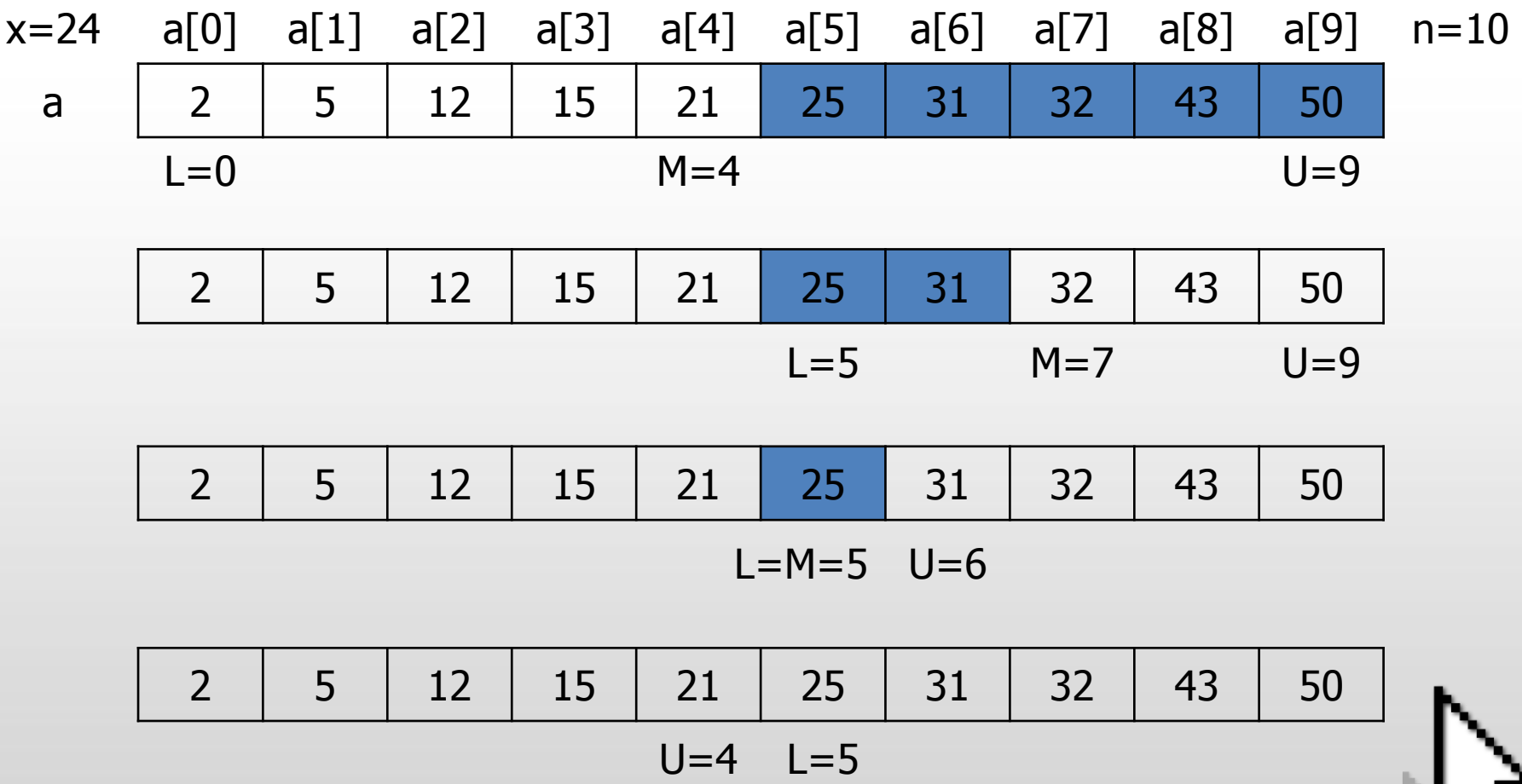
Binary Search



Binary Search



Binary Search



1.1 Arrays

1.1.2 Sorting

Bubble Sort

```
void bubble_sort(int a[], int n){
    int i,j;

    for(i=0;i<n-1;i++)
        for(j=1;j<n;j++)
            if (a[j]<a[j-1])
                swap(&a[j],&a[j-1]);
}
```

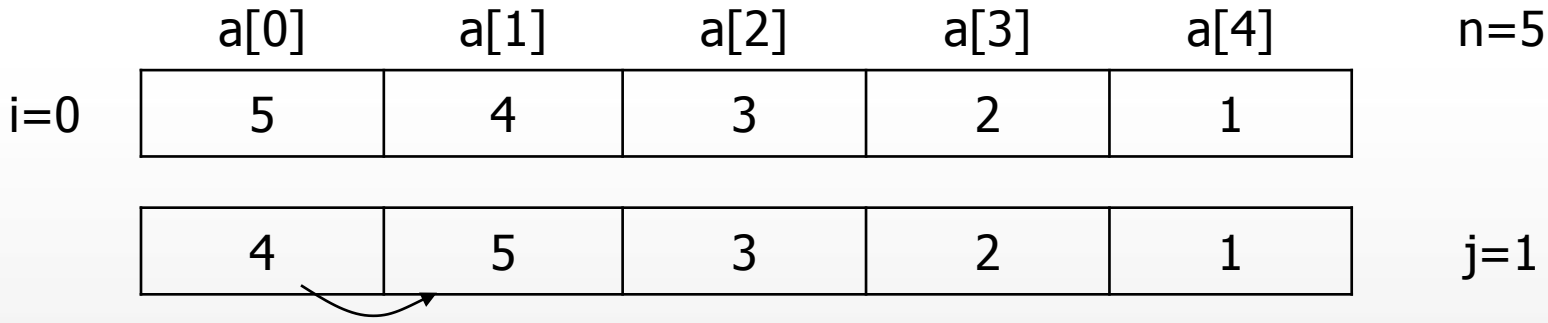
```
void bubble_sort(int a[], int n){
    int i,j;

    for(i=0;i<n-1;i++)
        for(j=1;j<n-i;j++)
            if (a[j]<a[j-1])
                swap(&a[j],&a[j-1]);
}
```

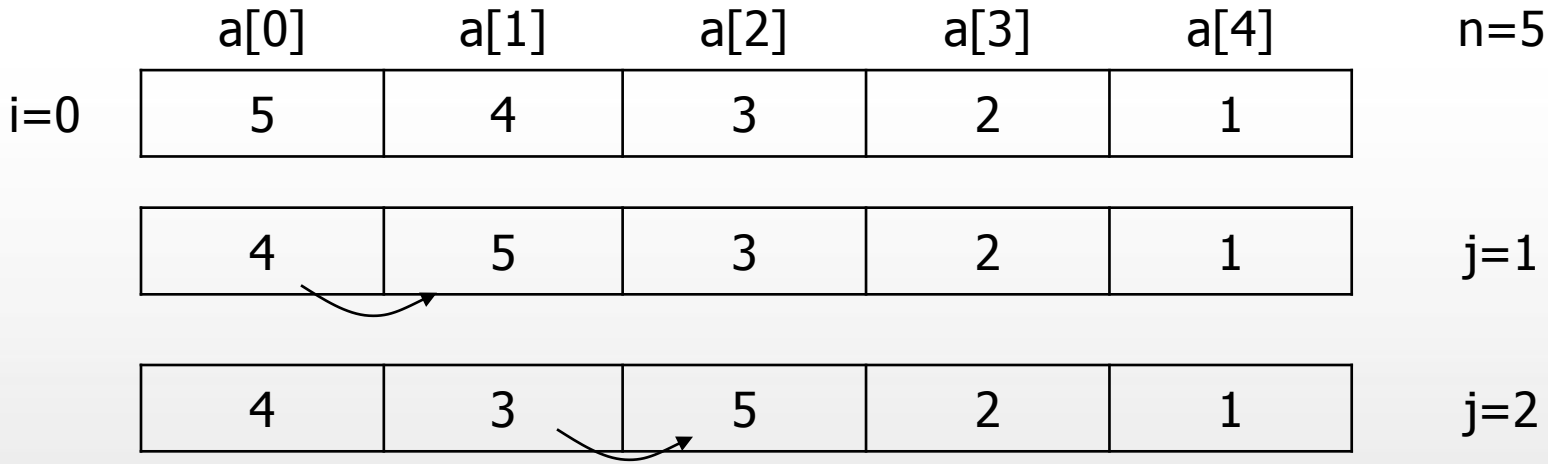
Bubble Sort (inversely sorted input)

	a[0]	a[1]	a[2]	a[3]	a[4]	n=5
i=0	5	4	3	2	1	

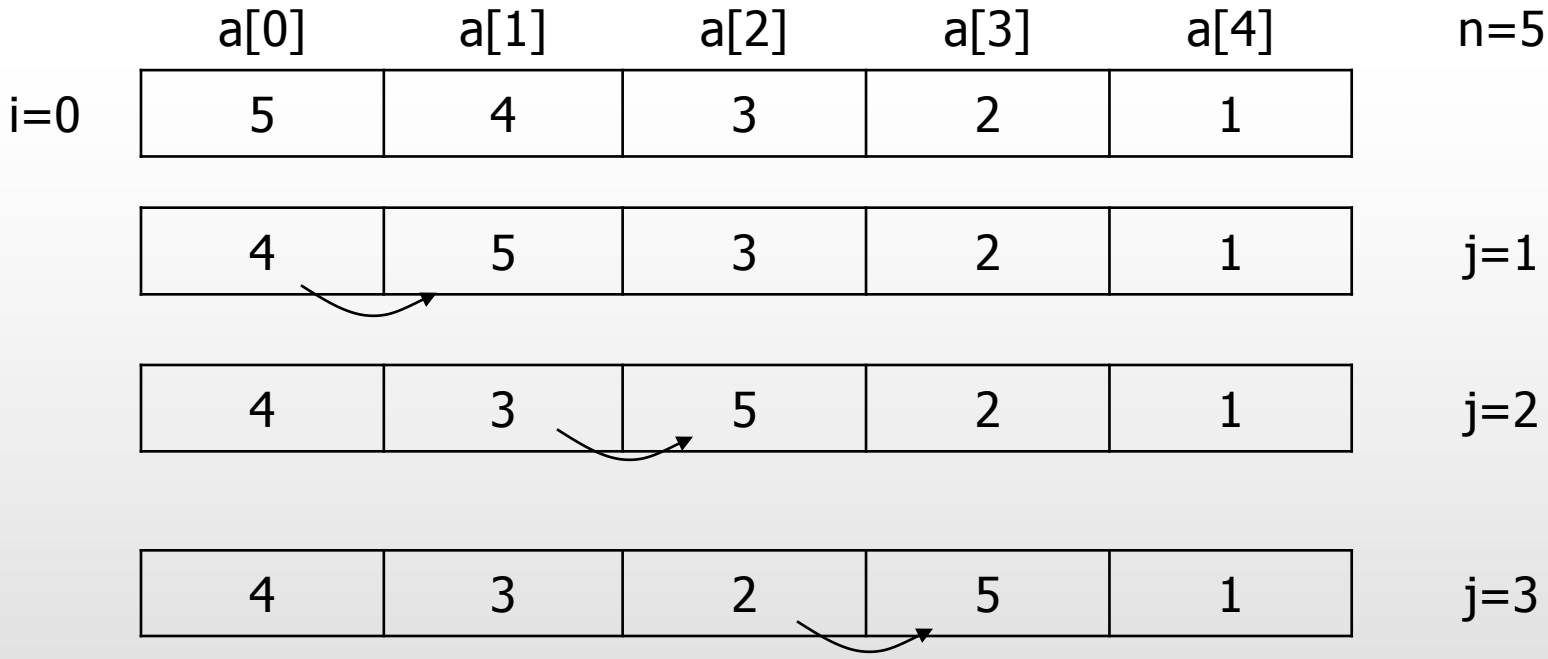
Bubble Sort (inversely sorted input)



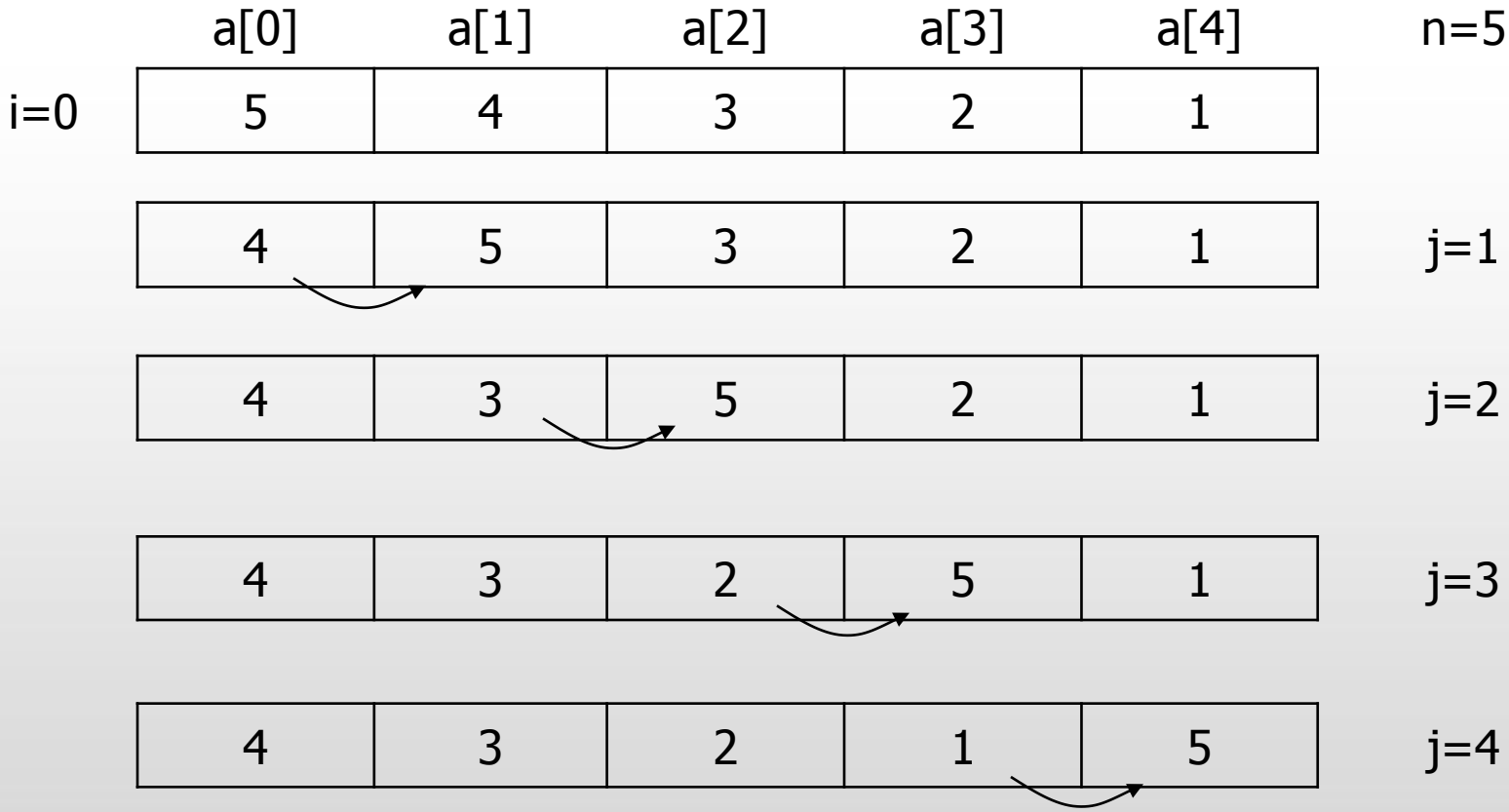
Bubble Sort (inversely sorted input)



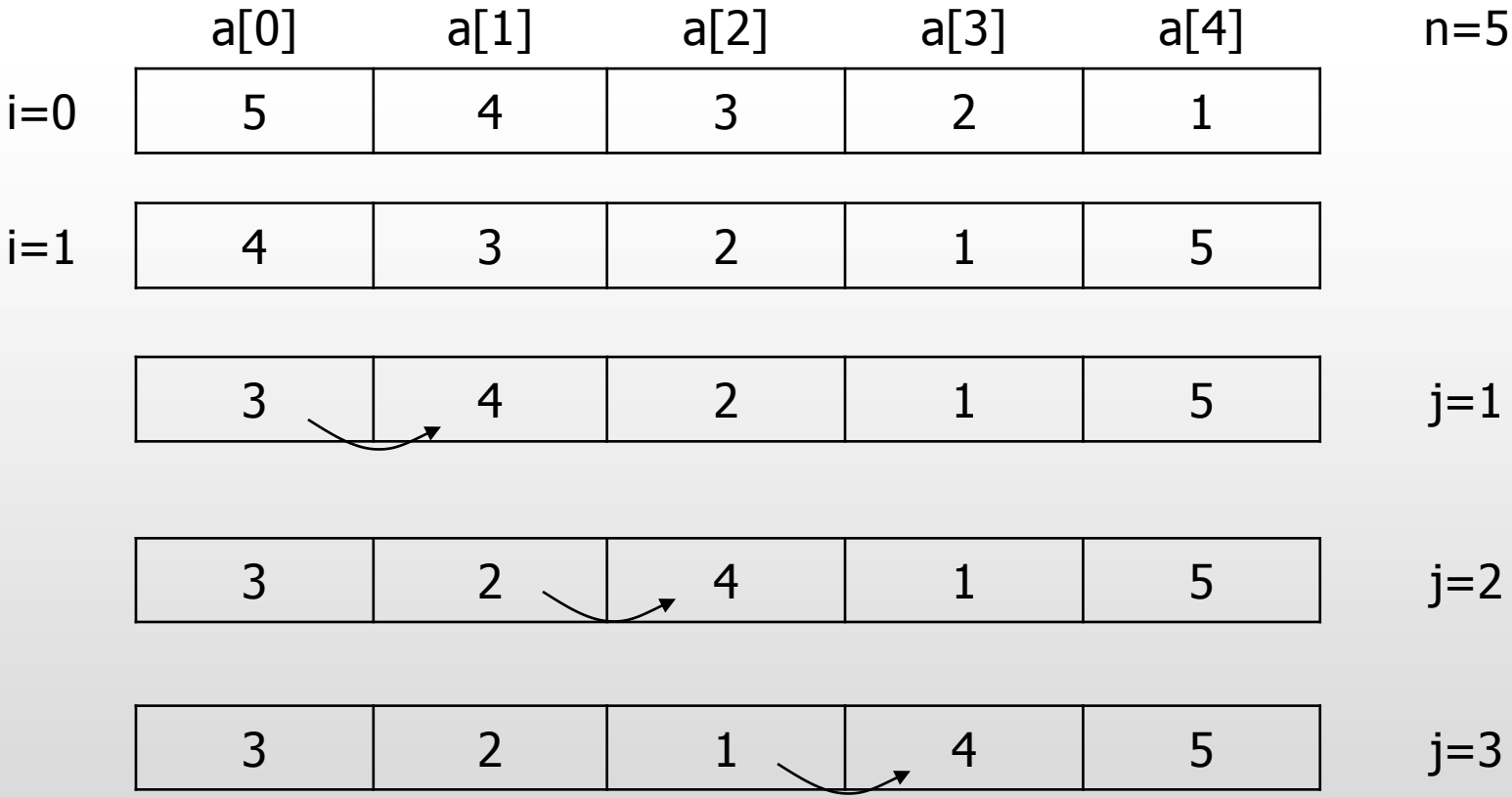
Bubble Sort (inversely sorted input)



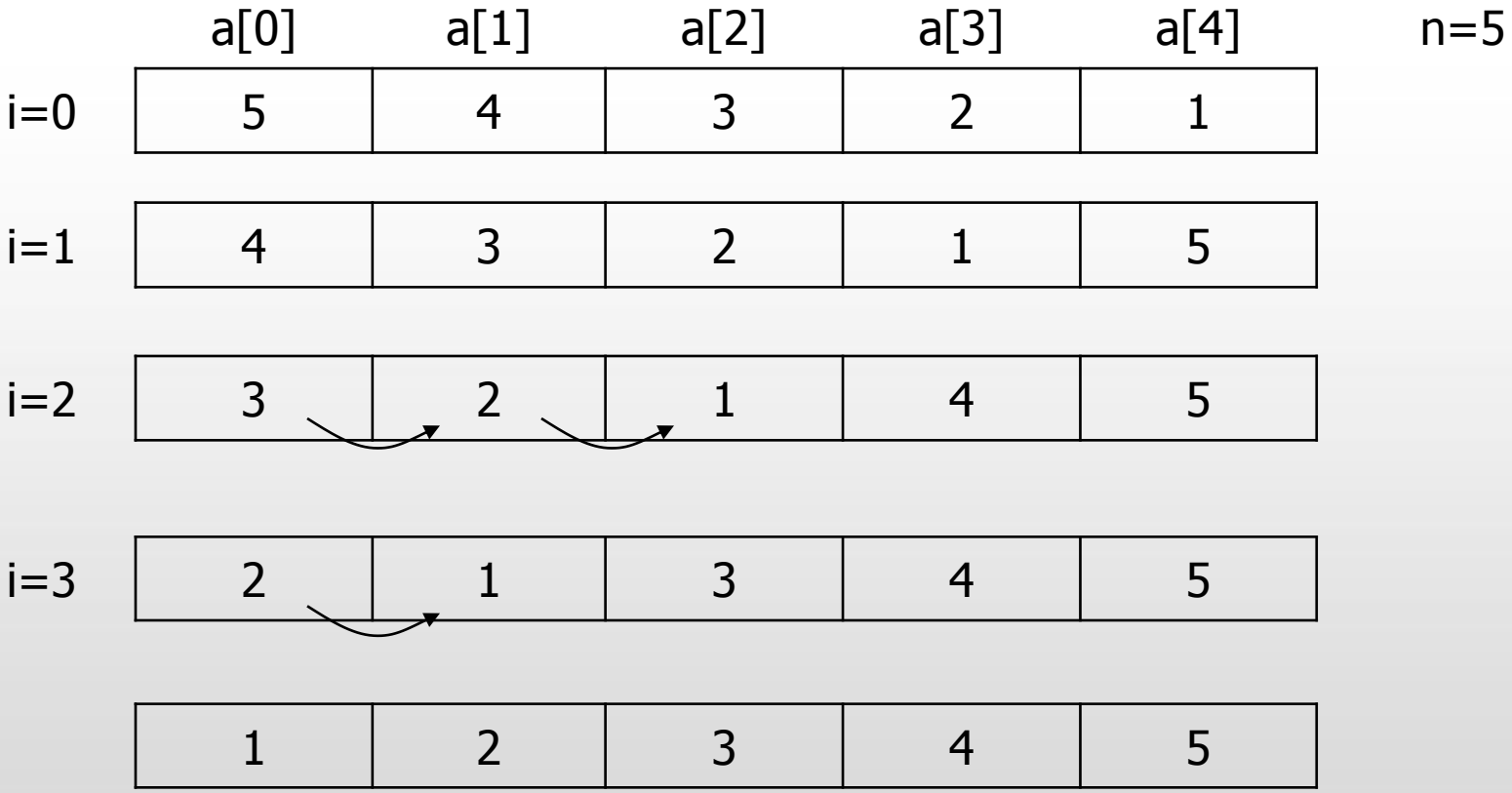
Bubble Sort (inversely sorted input)



Bubble Sort (inversely sorted input)



Bubble Sort (inversely sorted input)



Bubble Sort

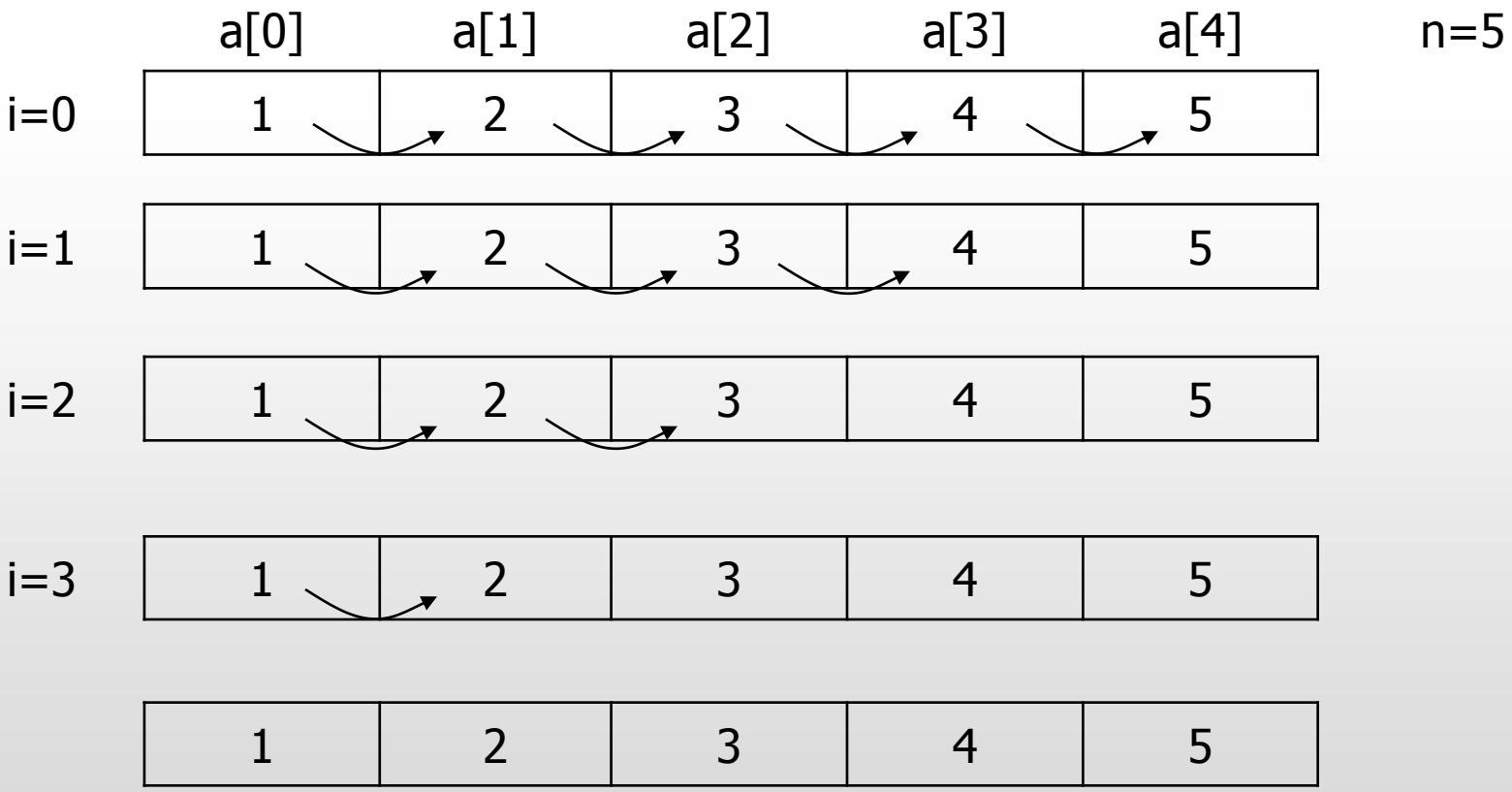
```
void bubble_sort(int a[], int n){
    int i,j;

    for(i=0;i<n-1;i++)
        for(j=1;j<n;j++)
            if (a[j]<a[j-1])
                swap(&a[j],&a[j-1]);
}
```

```
void bubble_sort(int a[], int n){
    int i,j;

    for(i=0;i<n-1;i++)
        for(j=1;j<n-i;j++)
            if (a[j]<a[j-1])
                swap(&a[j],&a[j-1]);
}
```

Bubble Sort (pre-sorted input)



Selection Sort

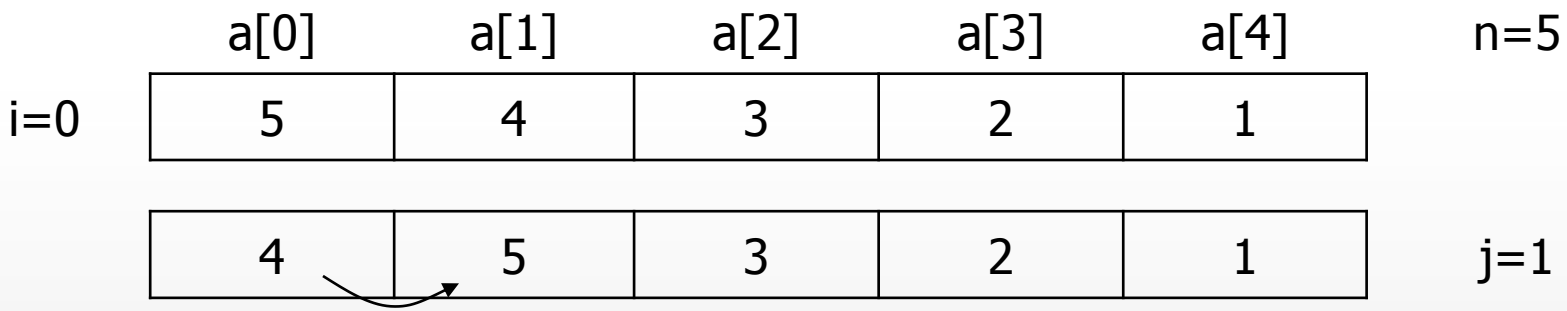
```
void selection_sort(int a[], int n){
    int i,j;

    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
            if (a[j]<a[i])
                swap(&a[i],&a[j]);
}
```

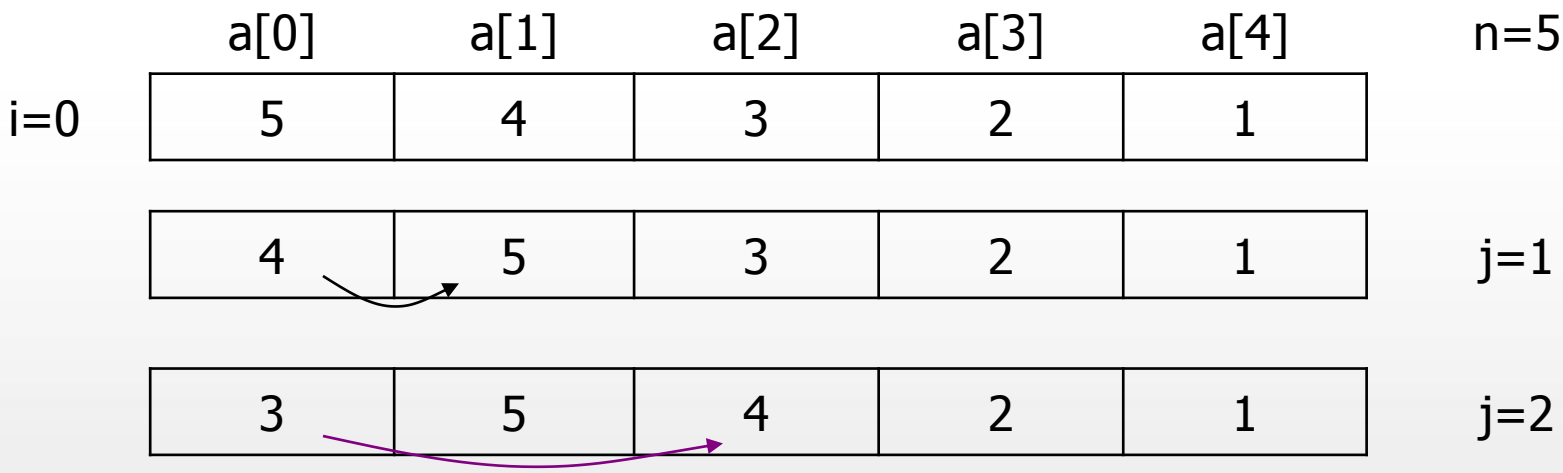
Selection Sort (inversely sorted input)

	a[0]	a[1]	a[2]	a[3]	a[4]	n=5
i=0	5	4	3	2	1	

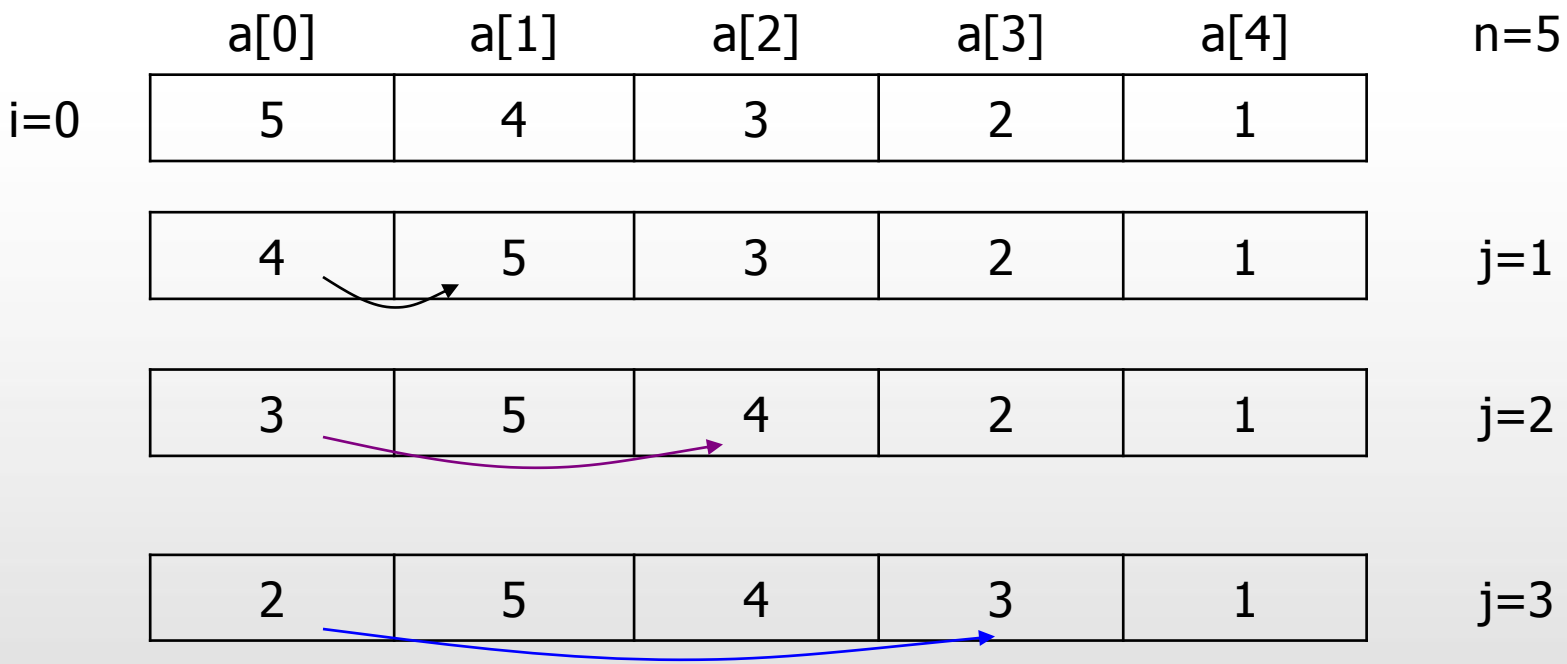
Selection Sort (inversely sorted input)



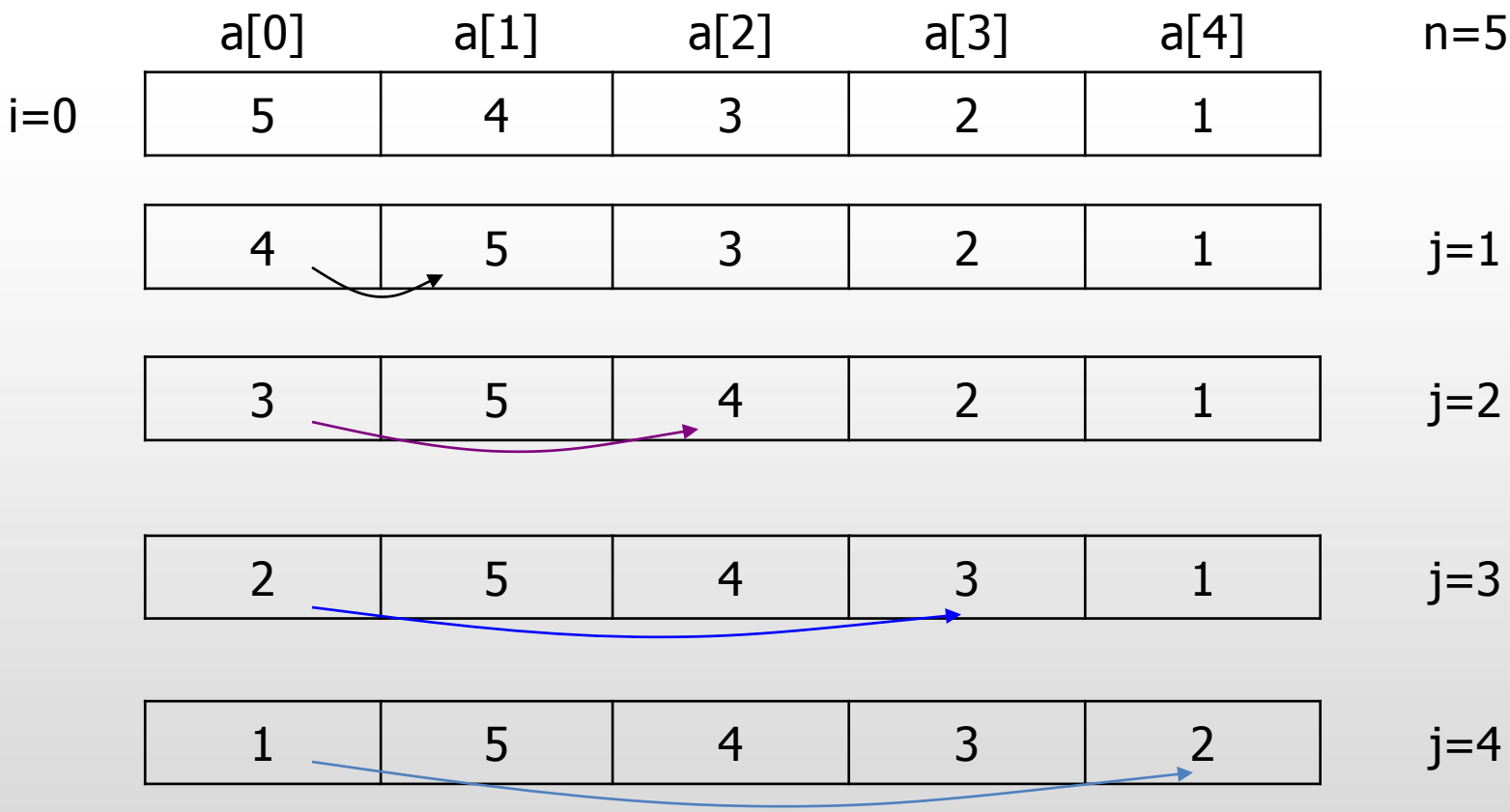
Selection Sort (inversely sorted input)



Selection Sort (inversely sorted input)

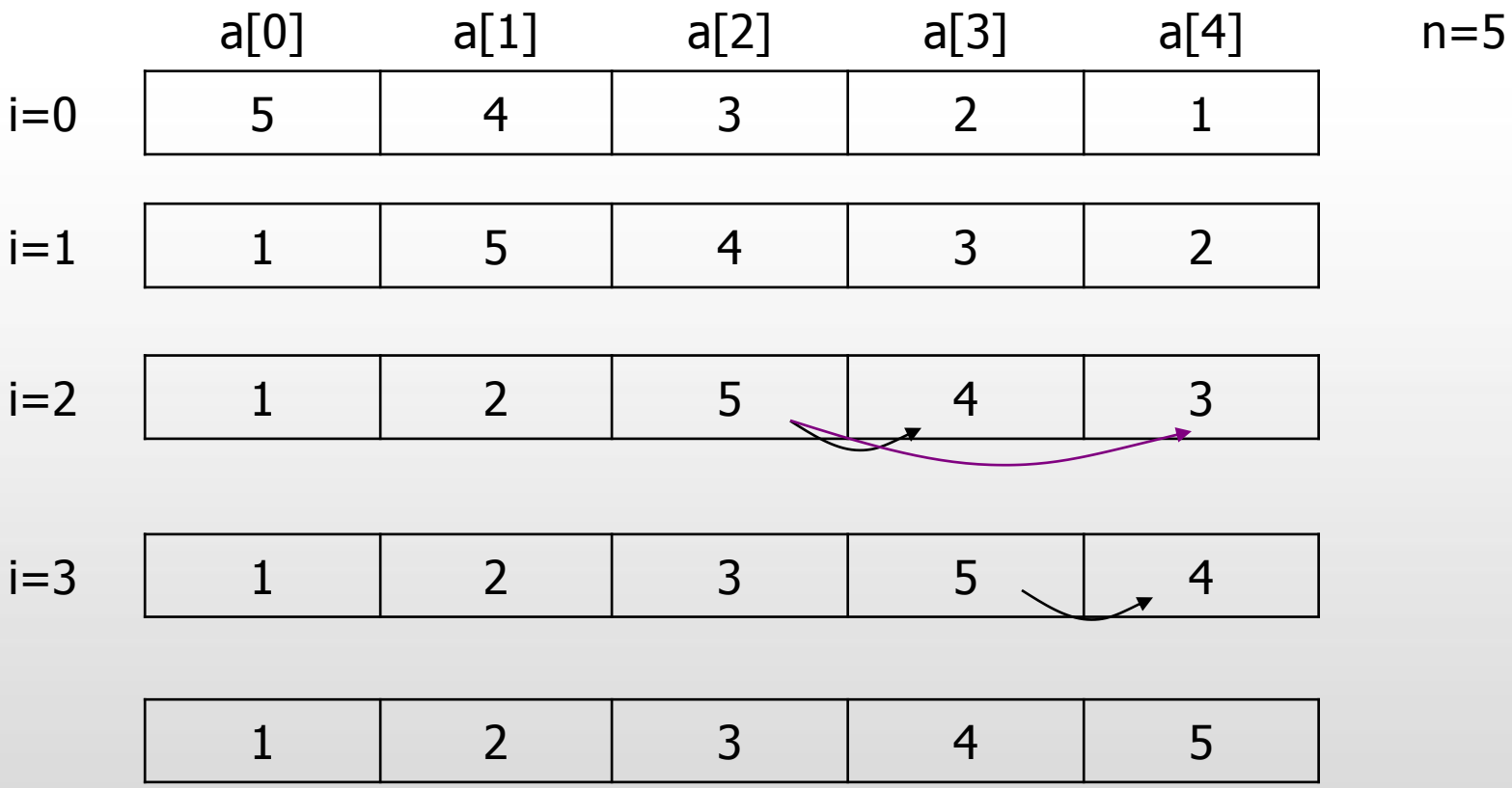


Selection Sort (inversely sorted input)

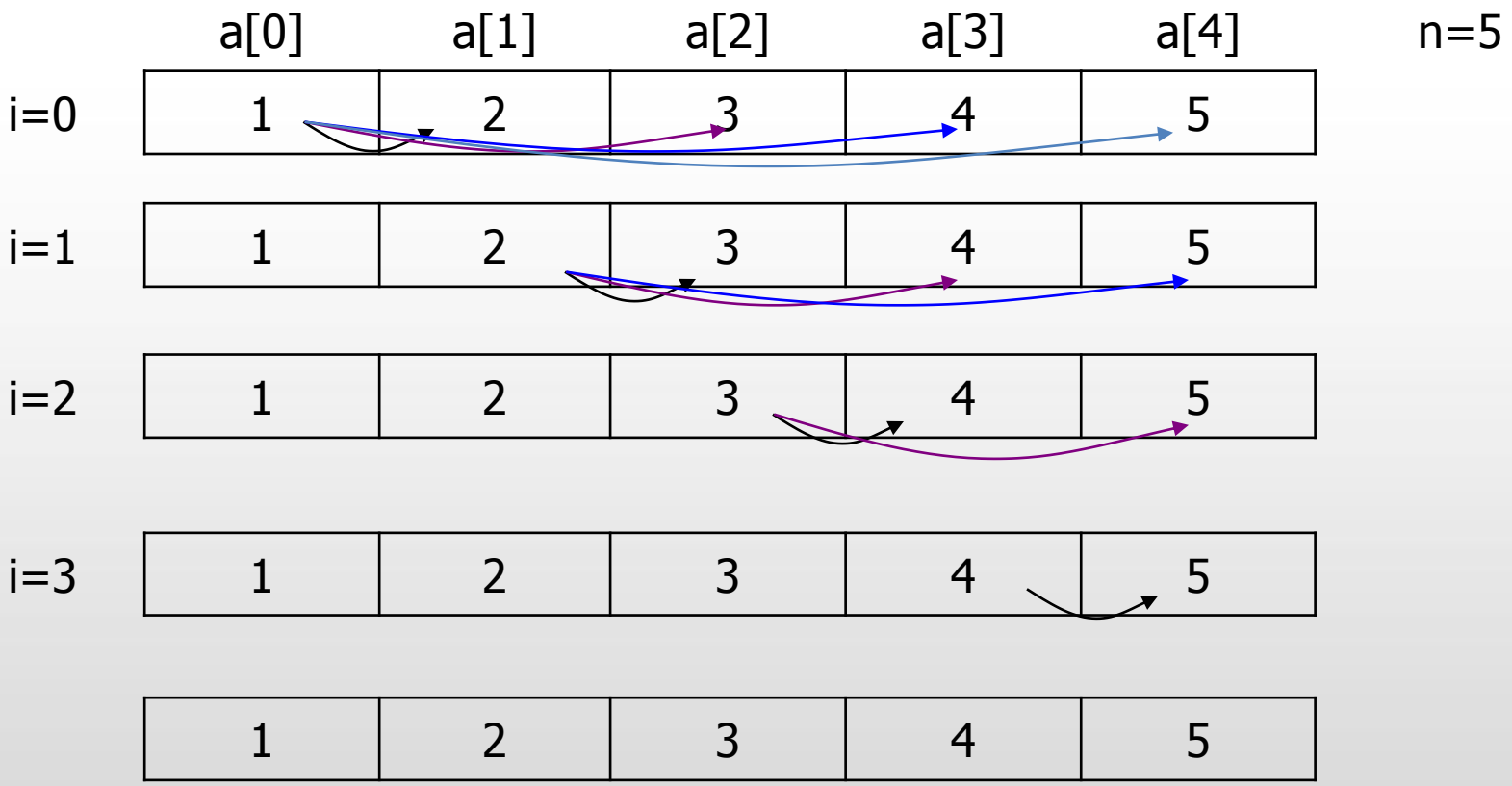


Selection Sort (inversely sorted input)

Selection Sort (inversely sorted input)



Selection Sort (pre-sorted input)

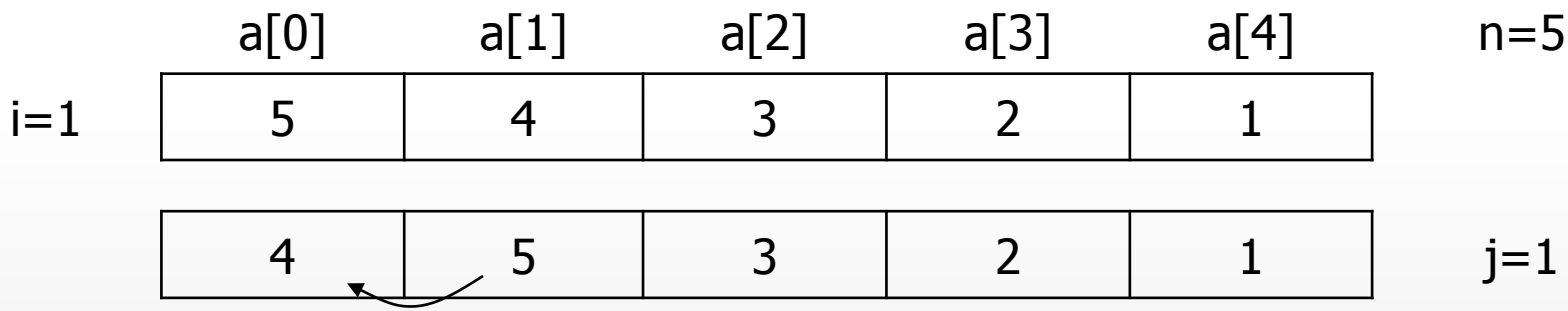


Insertion Sort

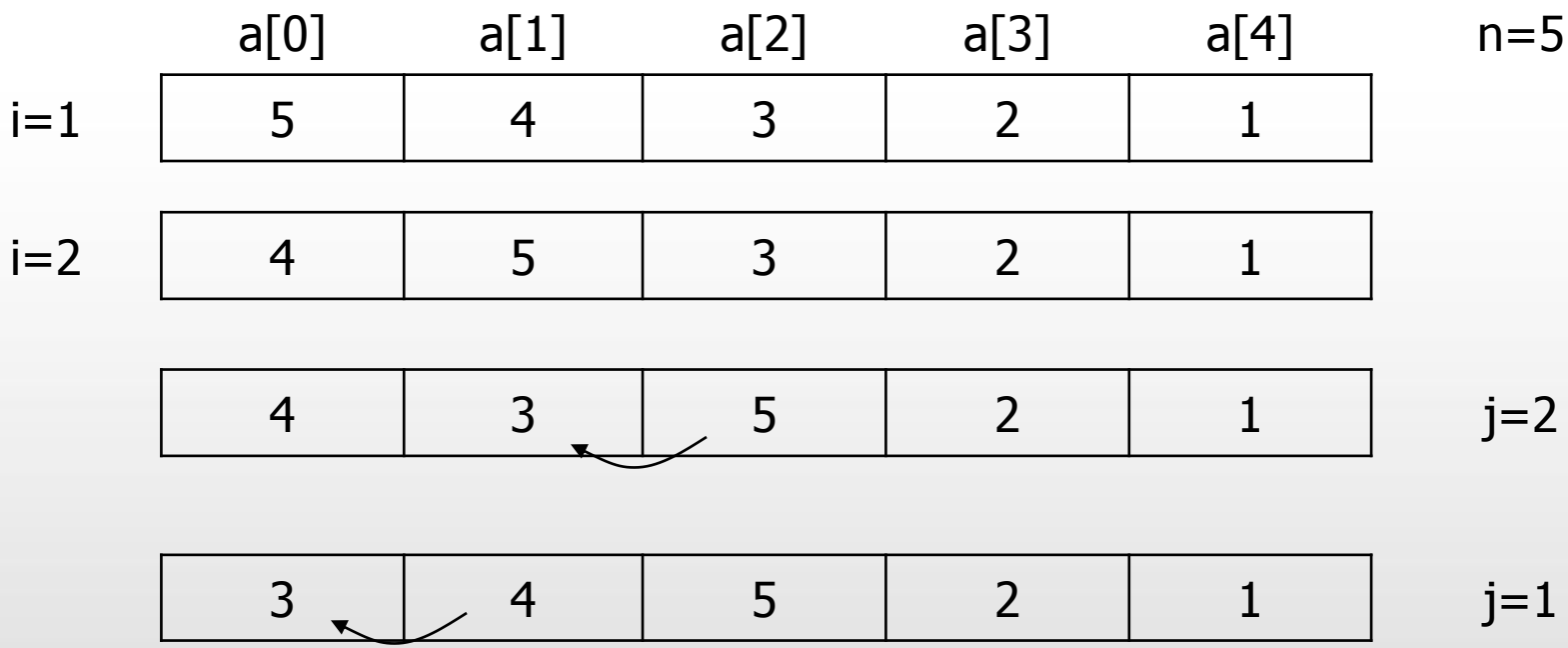
```
void insertion_sort(int a[], int n){
    int i,j;

    for(i=1;i<n;i++)
        for(j=i;j>0;j--)
            if (a[j]<a[j-1])
                swap(&a[j-1],&a[j]);
            else
                break;
}
```

Insertion Sort (inversely sorted input)



Insertion Sort (inversely sorted input)



Insertion Sort (inversely sorted input)

Insertion Sort (pre-sorted input)

Merge Sort

```
void merge_sort(int a[], int lower, int upper){
    int mid;

    if (upper-lower>0) {
        mid=(lower+upper)/2
        merge_sort(a, lower, mid);
        merge_sort(a, mid+1, upper);
        merge(a, lower, mid, upper);
    }
}
```

Merge Sort

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
8	1	5	3	7	2	6	4

Merge Sort

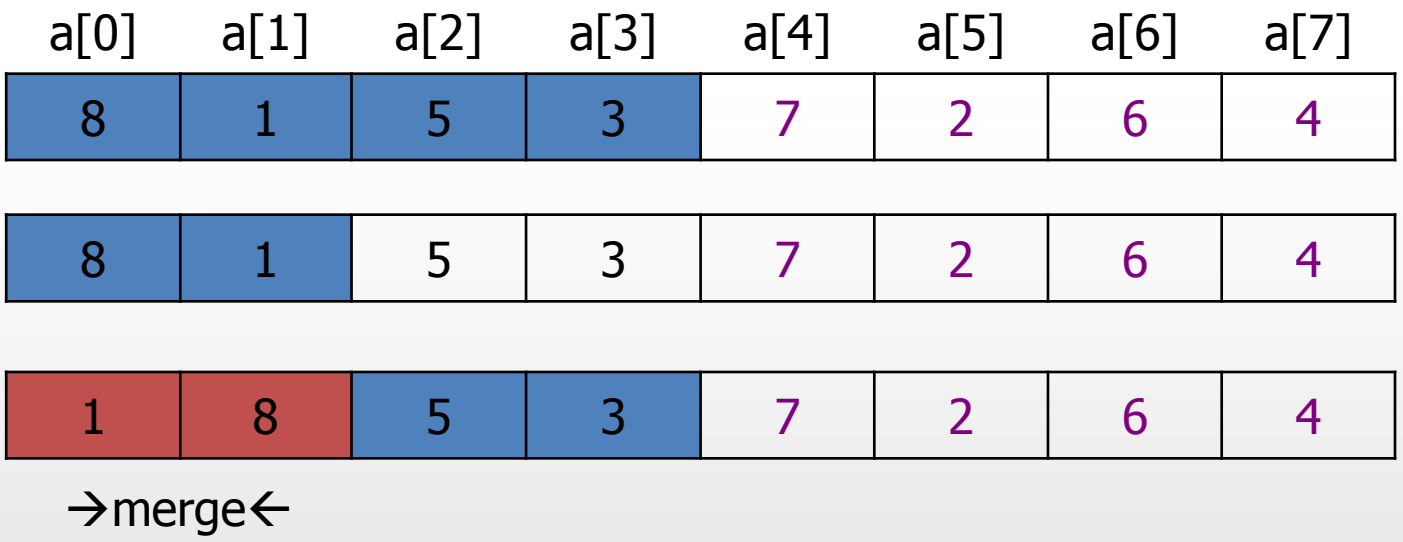
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
8	1	5	3	7	2	6	4

Merge Sort

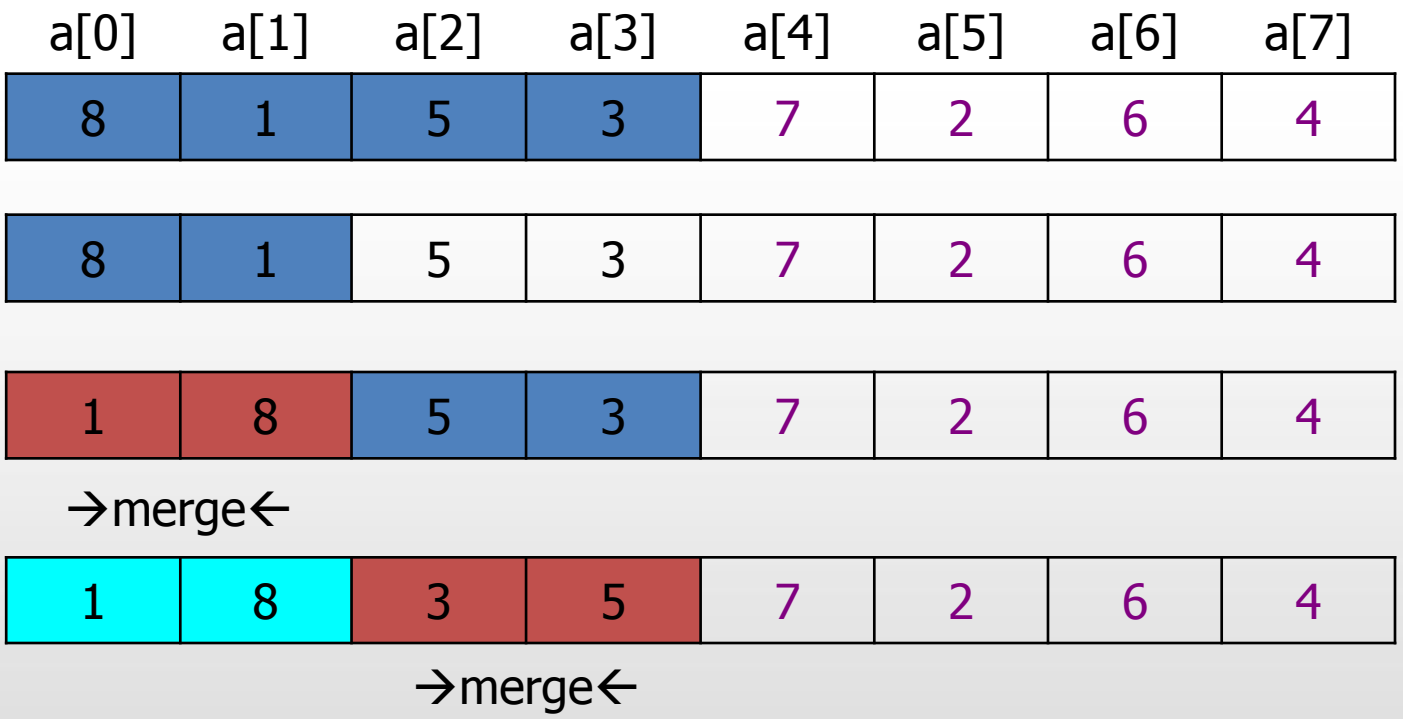
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
8	1	5	3	7	2	6	4

8	1	5	3	7	2	6	4

Merge Sort



Merge Sort


a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]
8 1 5 3 7 2 6 4
8 1 5 3 7 2 6 4
1 8 5 3 7 2 6 4
→merge←
1 8 3 5 7 2 6 4
→merge←

Merge Sort


a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]
8 1 5 3 7 2 6 4
8 1 5 3 7 2 6 4
1 8 5 3 7 2 6 4
→merge←
1 8 3 5 7 2 6 4
→merge←
1 3 5 8 7 2 6 4
→ merge ←

Merge Sort


a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]
1 3 5 8 7 2 6 4
1 3 5 8 2 7 6 4
→merge←
1 3 5 8 2 7 4 6
→merge←
1 3 5 8 2 4 6 7
→ merge ←
1 2 3 4 5 6 7 8
→ merge ←

Merge

```
void merge (int a[], int lower, int mid, int upper){
    int *temp,i,j,k;

    temp=(int *)malloc((upper-lower+1)*sizeof(int));

    for(i=0,j=lower,k=mid+1; j<=mid || k<=upper; i++)
        temp[i]=(j<=mid && (k>upper || a[j]<a[k]))?
                    a[j++]:a[k++];
    for(i=0,j=lower;j<=upper; i++, j++)
        a[j]=temp[i];

    free(temp);
}
```

Merge

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	
a	1	3	5	8	2	4	6	7	j++
temp	1								i=0
a	1	3	5	8	2	4	6	7	k++
temp	1	2							i=1
a	1	3	5	8	2	4	6	7	j++
temp	1	2	3						i=2
a	1	3	5	8	2	4	6	7	k++
temp	1	2	3	4					i=3

Merge

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	
a	1	3	5	8	2	4	6	7	j++
temp	1	2	3	4	5				i=4
a	1	3	5	8	2	4	6	7	k++
temp	1	2	3	4	5	6			i=5
a	1	3	5	8	2	4	6	7	k++
temp	1	2	3	4	5	6	7		i=6
a	1	3	5	8	2	4	6	7	j++
temp	1	2	3	4	5	6	7	8	i=7