Searching Techniques

Objectives

To learn and appreciate the following concepts

Searching Technique

- Linear Search
- Binary Search

Session outcome

- At the end of session student will be able to understand
  - Searching Techniques

Arrays – A recap

1D Array:

Syntax: **type array_name[size];**

- **Initialization:**

  type array-name [size]={list of values};

```
int arr[]={1,2,3,4,5};
```

- **Read:**

```
for(i=0;i<n;i++)
scanf("%d", &a[i]);
```

Write:

```
for(i=0;i<n;i++)
printf("%d", a[i]);
```

- **examples:**

```
int arr[5]={1,2};
```

arr is

1	2	0	0	0

```
int arr[]={1,2};
```

arr is

1	2

```
int arr[5]={0};
```

arr is

0	0	0	0	0

```
int arr[3]={1};
```

arr is

1	1	1

1

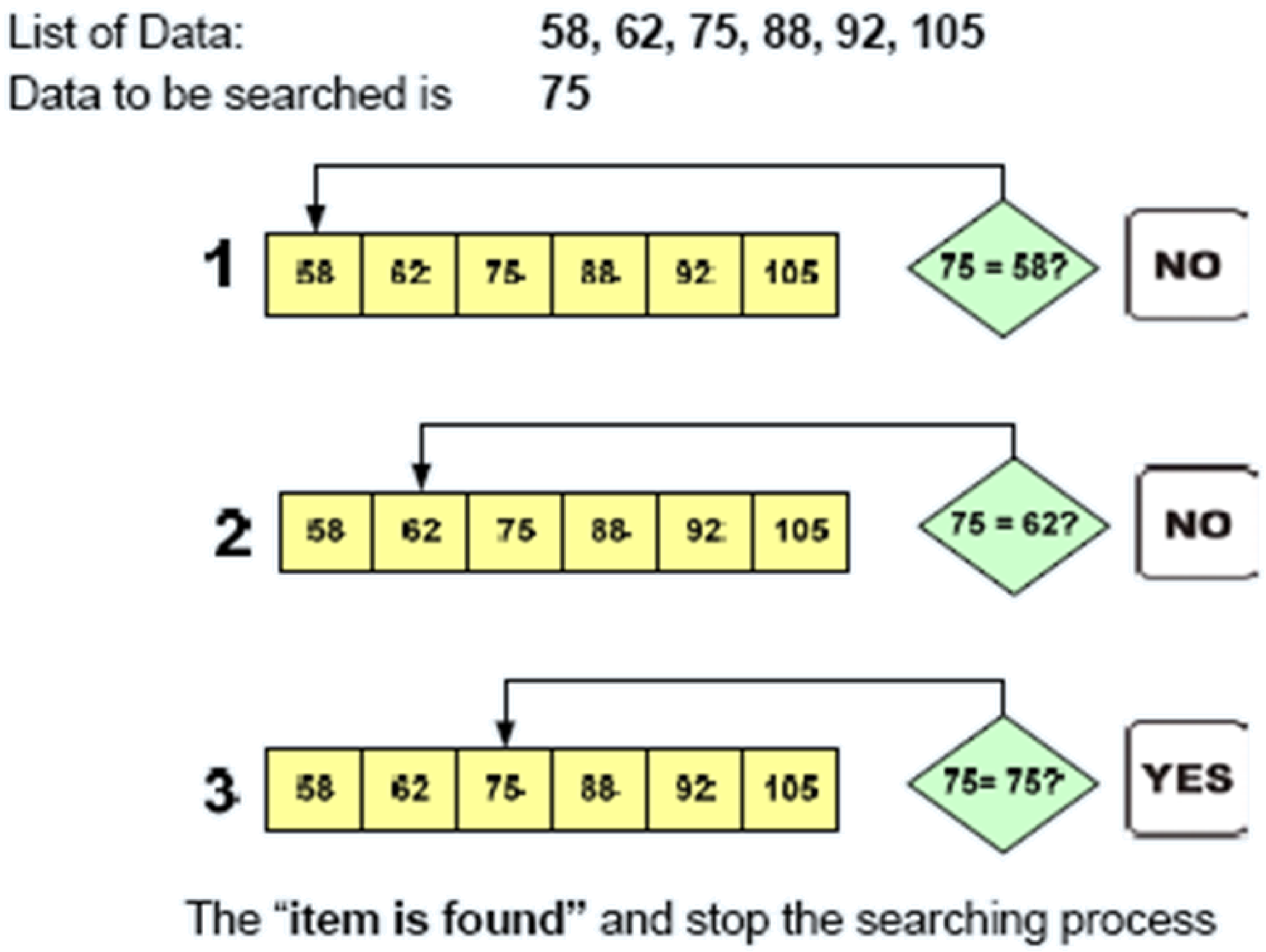
1	0	0

Linear Search or Sequential Search

Finding whether a data item is present in a set of items

- The Linear Search is applied on the set of items that are **not arranged** in any particular order
- In linear search , the searching process starts from the **first item**.
- The **searching is continued** till either the **item is found** or the **end of the list is reached** indicating that the item is not found.
- The items in the list are assumed to be **unique**.

Linear search- illustration 1



Linear search- illustration 2

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

search element **12**

Step 1:

search element (12) is compared with first element (65)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
	12							

Both are not matching. So move to next element

Step 2:

search element (12) is compared with next element (20)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
		12						

Both are not matching. So move to next element

Step 3:

search element (12) is compared with next element (10)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
			12					

Both are not matching. So move to next element

Step 4:

search element (12) is compared with next element (55)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
				12				

Both are not matching. So move to next element

Step 5:

search element (12) is compared with next element (32)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
					12			

Both are not matching. So move to next element

Step 6:

search element (12) is compared with next element (12)

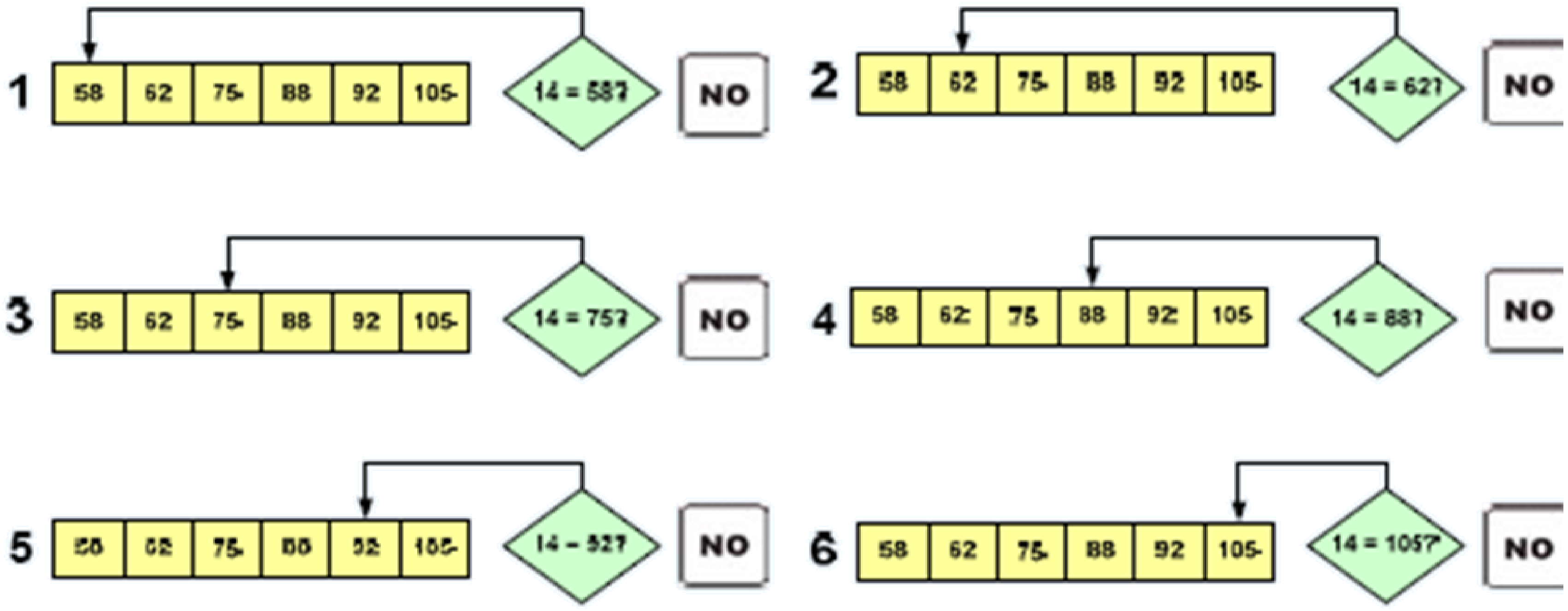
	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99
						12		

Both are matching. So we stop comparing and display element found at index 5.

Linear search- illustration 3

List of Data: 58, 62, 75, 88, 92, 105

Data to be searched is 14



Now the end of the list is reached. There are no more elements in the list.
So the item **14 is "not found"** in the list.

Linear search – procedure

```
int main(){
int found, arr[10],i , n, key, pos;
found=0; //setting flag
printf("enter no of numbers");
scanf("%d", &n);
printf("enter the numbers:\n");
// enter array data items
for(i=0;i<n;i++)
scanf("%d", &arr[i]);
printf("enter the search element");
// data to be searched
scanf("%d", &key);
```

```
/*search procedure*/
for(i=0; i<n; i++) {
if(arr[i]==key){    // comparison
        found=1;
        pos=i+1;
        break;
    }
}

if(found==1)
   printf("Element found in %d position.", pos);
else
   printf("Searched element NOT FOUND.");

return 0;
}
```

Binary Search

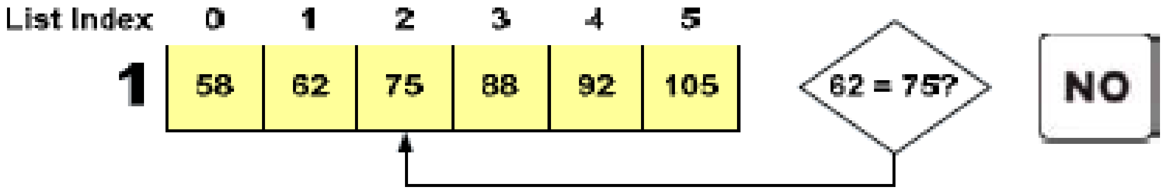
- A binary search is a searching technique that can be applied only to a **sorted list** of items
- This searching technique is similar to dictionary search.
- **Algorithm:**
  - **Step 1**: Set First = 0 and Last = Number of Items – 1
  - **Step 2**: Find the middle of the list as mid = (First + Last) /2. Take only the integer part, if the result is a real number.
  - **Step 3**: Compare the middle item with the searching item. If they are equal then "**Item is found**" and go to step 8.
  - **Step 4**: If the searching item is less than the middle item then the searching item comes before this middle element. So, set Last = mid -1 and there is no change in the value of First. Go to step 6.
  - **Step 5**: Since the above conditions are false the searching element should be greater than the middle element. So, set First = mid +1 and there is no change in the value of Last. Go to the next step.
  - **Step 6**: If First <= Last then go to step 2.
  - **Step 7**: Since end of the list is reached, the searching item is "**not found**" in the list
  - **Step 8**: End of the algorithm.

Binary Search – example-1

List of Data: **58, 62, 75, 88, 92, 105**

Data to be searched is **62**

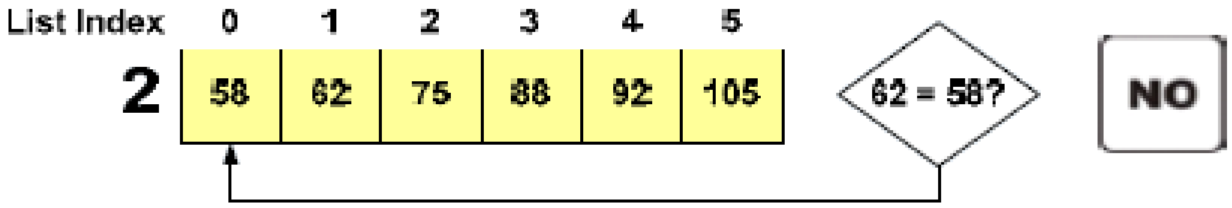
- <u>Step 1:</u> First = 0 and Last = 5
- <u>Step 2:</u> Step 2: Mid = (0 + 5) / 2. That is Mid = 2 (Only the integer part is taken)
- <u>Step 3:</u>



- <u>Step 4:</u> The searching item 62 is less than 75. So it should appear before 75. Now First = 0 and Last = mid-1 that is Last = 2-1. So Last = 1
- <u>Step 5:</u> Compute Mid = (0 + 1) / 2 that is Mid = 0 (Integer part)

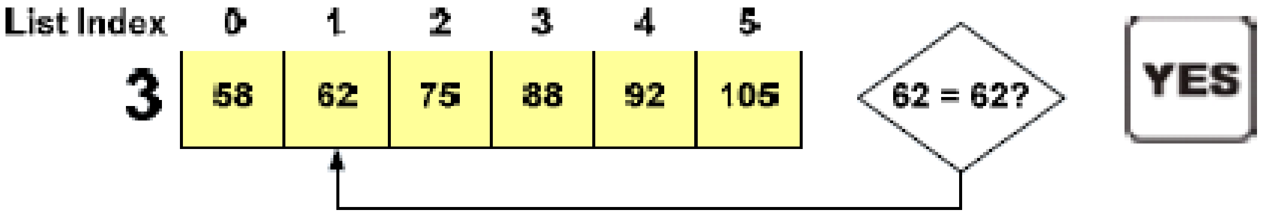
Binary Search – example-1

- Step 6: Compare 0^{th} item with 62. That is compare 58 and 62. Since they are not equal proceed to the next step



- Step 7: Since the searching item 62 is greater than 58, the searching item comes after 58. First = mid+1 that is First = 0+1. So, First = 1 and Last=1. Now, mid=(1+1)/2=1

- Step 8: Compare 62 with the item in position 1. That is also 62. So, the "**item is found**" and stop the searching process

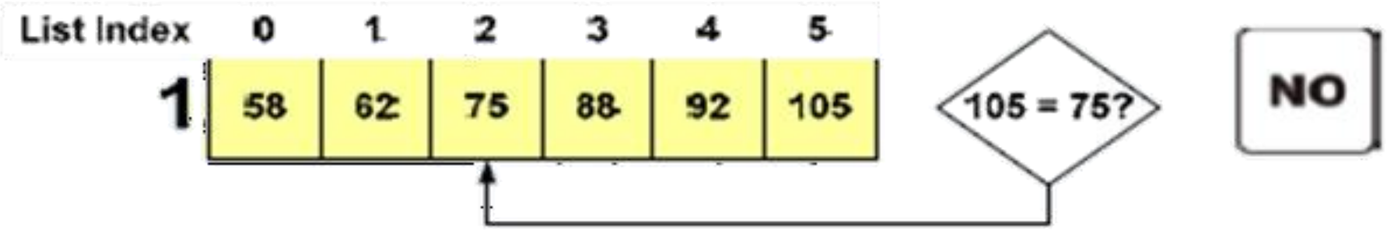


Binary Search – example-2

List of Data: **58, 62, 75, 88, 92, 105**

Data to be searched is **105**

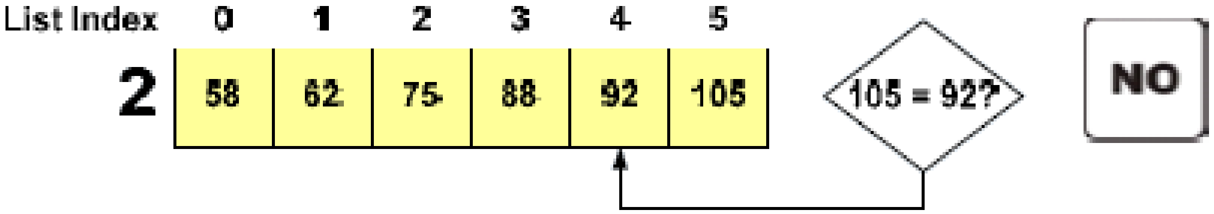
- Step 1: First = 0 and Last = 5
- Step 2: Step 2: Mid = (0 + 5) / 2. That is Mid = 2 (Only the integer part is taken)
- Step 3:



- Step 4: The searching item 105 is greater than 75. So it comes after 75.
  First = 2 +1=3. That is, First=3 and Last=5

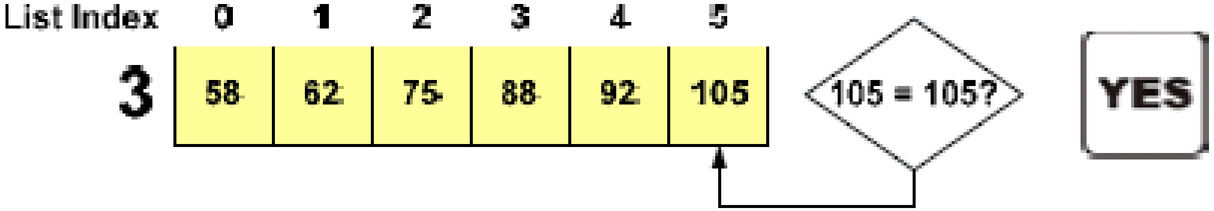
Binary Search – example-2

- <u>Step 5:</u> Compute Mid = (3+5) / 2 = 4



- <u>Step 6:</u> The searching item 105 is greater than 92. So the searching item 105 comes after 92. First= (4+1) = 5 and Last =5. So Mid=(5+5)/2 = 5

- <u>Step 7:</u> Compare Searching element 105 with the 5th element. Since they are equal "**Item is found**" and stop the searching process

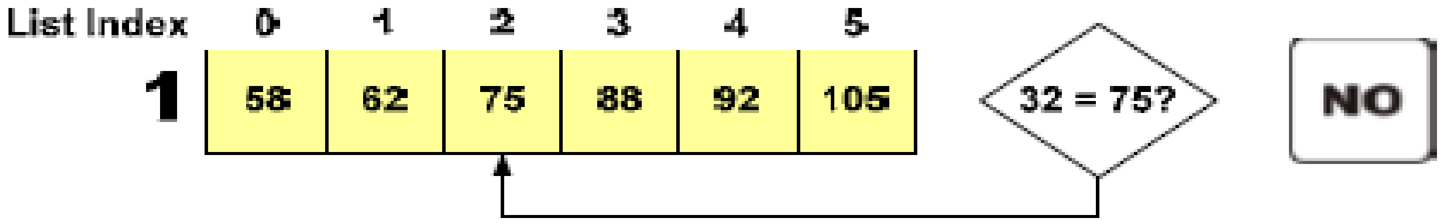


Binary Search – example-3

List of Data: **58, 62, 75, 88, 92, 105**

Data to be searched is **32**

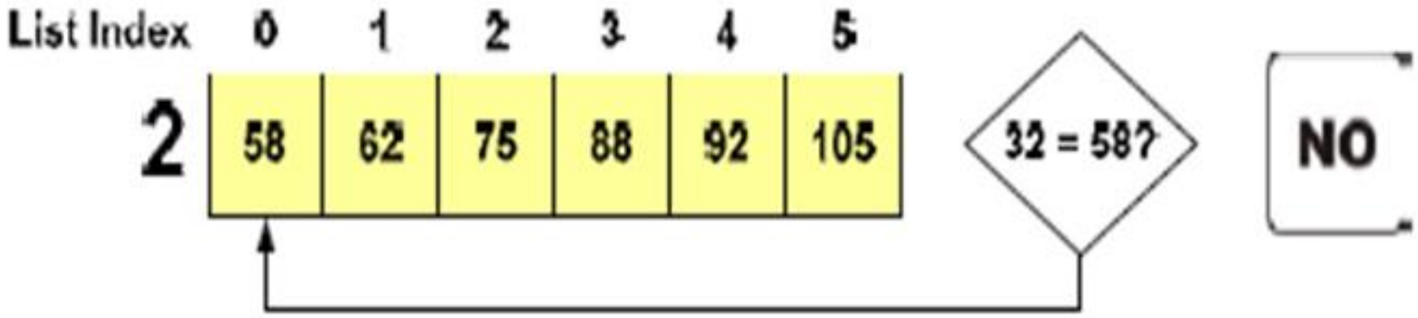
- Step 1: First = 0 and Last = 5
- Step 2: Mid = (0 + 5) / 2. That is Mid = 2 (Only the integer part is taken)
- Step 3: Compare the searching item 32 and 75. Since they are not equal proceed with the next step



- Step 4: The searching item 32 is less than 75. So First=0 and Last=1. Mid=(0+1)/2=0

Binary Search – example-3

- Step 5: Compare 32 and 58. Since they are not equal proceed with the next step



- Step 6: The searching item 32 is less than 58. So First = Mid-1 that is First =0-1= -1 and First = 0. Since First > Last, **"Item is not found"** and stop the searching process

Binary Search

- A binary search is a searching technique that can be applied only to a **sorted list** of items
- This searching technique is similar to dictionary search.

- **Algorithm:**
  - **Step 1**: Set First = 0 and Last = Number of Items – 1
  - **Step 2**: Find the middle of the list as mid = (First + Last) /2. Take only the integer part, if the result is a real number.
  - **Step 3**: Compare the middle item with the searching item. If they are equal then "**Item is found**" and go to step 8.
  - **Step 4**: If the searching item is less than the middle item then the searching item comes before this middle element. So, set Last = mid -1 and there is no change in the value of First. Go to step 6.
  - **Step 5**: Since the above conditions are false the searching element should be greater than the middle element. So, set First = mid +1 and there is no change in the value of Last. Go to the next step.
  - **Step 6**: If First <= Last then go to step 2.
  - **Step 7**: Since end of the list is reached, the searching item is "**not found**" in the list
  - **Step 8**: End of the algorithm.

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo														↑ hi

```
lo is the lower bound and hi is the upper bound
Calculate mid(index of variable with which 33 is to be compared.)
      mid = ( lo + hi )/2
      mid = (0 + 14 )/2
      mid = 7
```

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo							↑ mid							↑ hi

```
    mid = 7
Compare 33 with element at a[7], i.e. 53
It is NOT matching!
As 33 is less than 53, new hi = mid - 1
```

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo						↑ hi								

```
Again calculate mid
     mid = ( lo + hi )/2
     mid = (0 + 6 )/2
     mid = 3
```

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lo			mid			hi								

```
mid = 3
Compare 33 with element at a[3], i.e. 25
It is NOT matching!
As 33 is greater than 25, new low = mid + 1
```

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lo hi

Again calculate mid

```
mid = ( lo + hi )/2
mid = (4 + 6 )/2
mid = 5
```

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lo mid hi

```
mid = 5
Compare 33 with element at a[5], i.e. 43
It is NOT matching!
As 33 is smaller than 43, new hi = mid - 1
```

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lo
hi

```
Again calculate mid
     mid = ( lo + hi )/2
     mid = (4 + 4 )/2
     mid = 4
```

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

lo
hi
mid

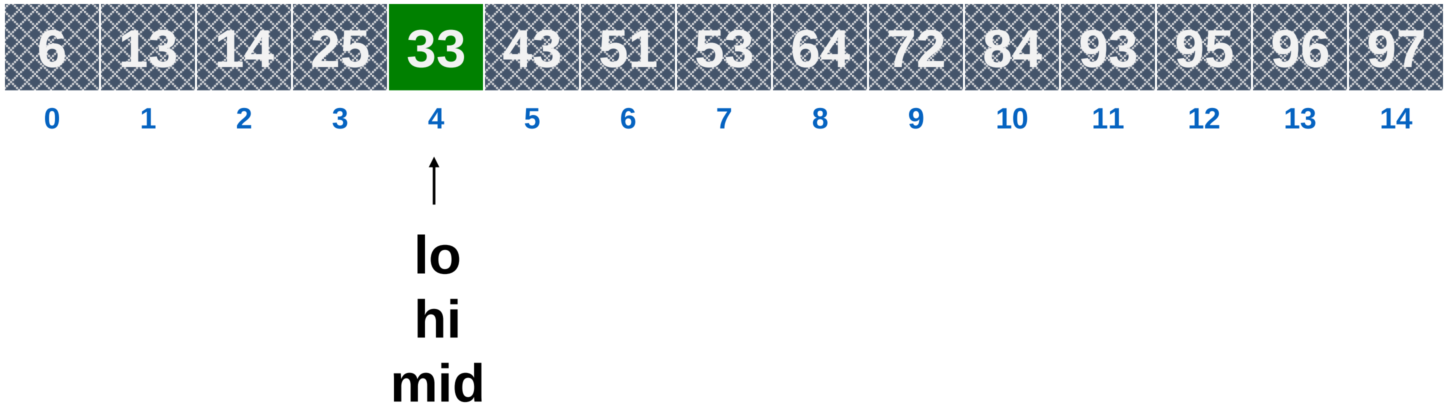
mid = 4

Compare 33 with element at a[4], i.e. 33

It is MATCHING!

Binary Search

- Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Ex. Binary search for 33.



`The element is found at a[4].`

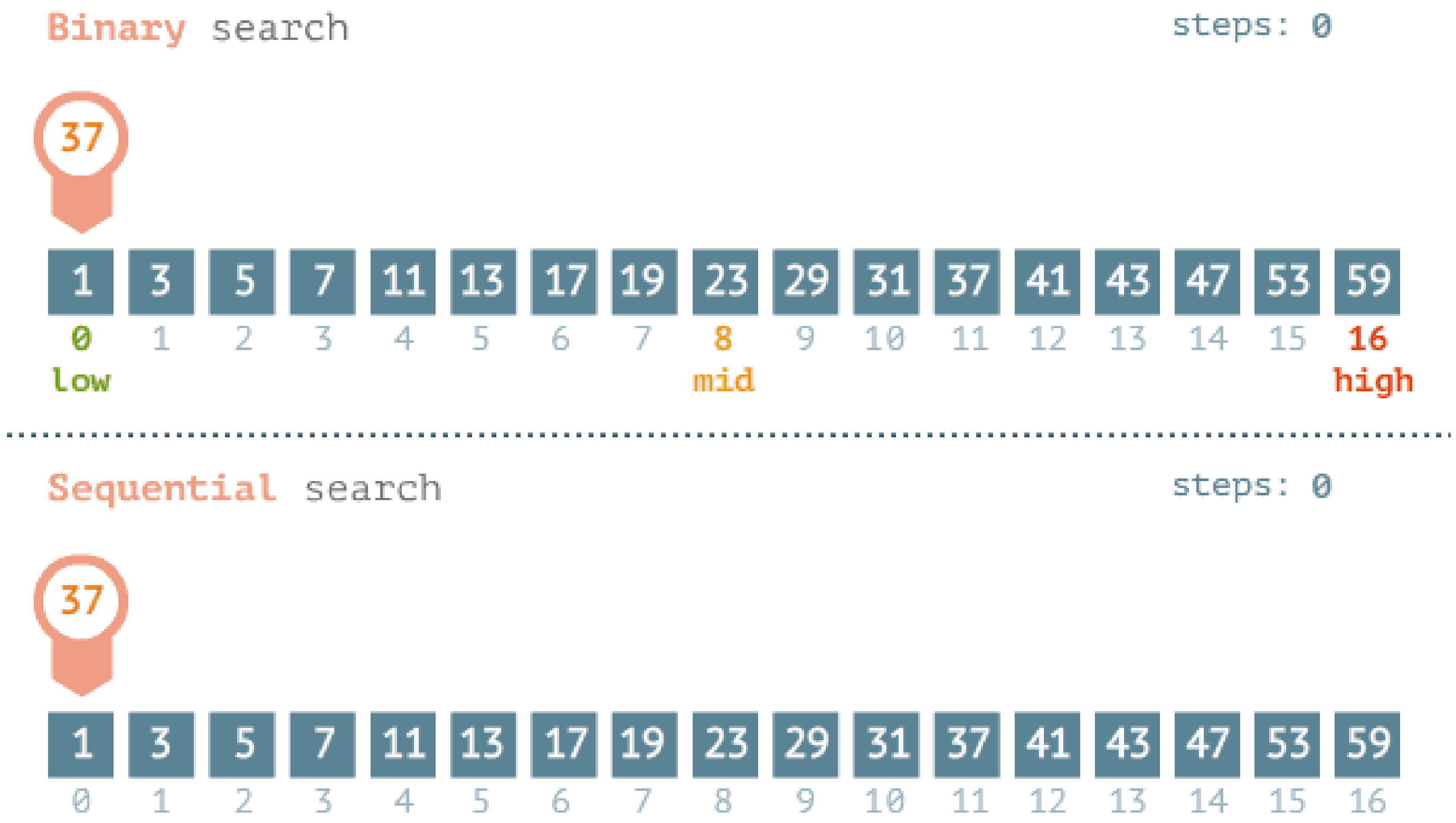
Binary search

Binary Search – procedure

```
/* Binary search on sorted array */
low=0;
high=N-1;
do {
mid= (low + high) / 2;
if ( key < array[mid] )
high = mid - 1;
else if ( key > array[mid])
low = mid + 1;
} while( key!=array[mid] && low <= high);
```

```
if( key == array[mid] )
  printf("SUCCESSFUL SEARCH.\n
  Element found at position %d", mid+1);
else
  printf("Search element NOT Found\n");
```

Linear *versus* Binary Search



Linear *versus* Binary Search

Linear Search	Binary Search
Can be applied on sorted and unsorted list of items	Can be applied only on sorted list of items
Searching time is more	Searching time is less

Summary

- **Searching Technique**
  - Linear Search
  - Binary Search

Sorting Techniques

Objectives

To learn and appreciate the following concepts

Sorting Technique

- **Bubble Sort**
- **Selection Sort**

Sorting

Arrangement of data elements in a particular order

→ **Bubble sorting**

Bubble Sort

- **Sorting takes an unordered collection and makes it an ordered one.**

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
5	**12**	**35**	**42**	**77**	**101**

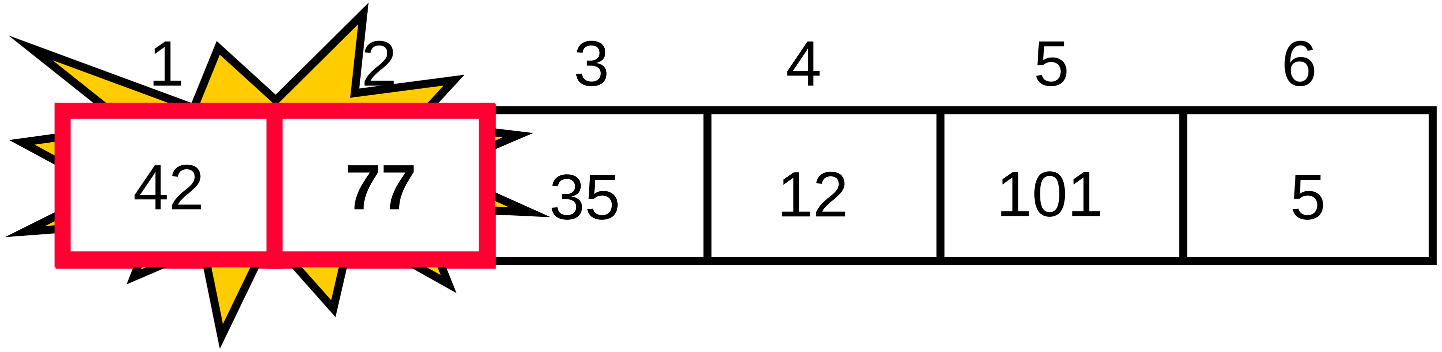
Bubble Sort- Illustration: "Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

1	2	3	4	5	6
77	42	35	12	101	5

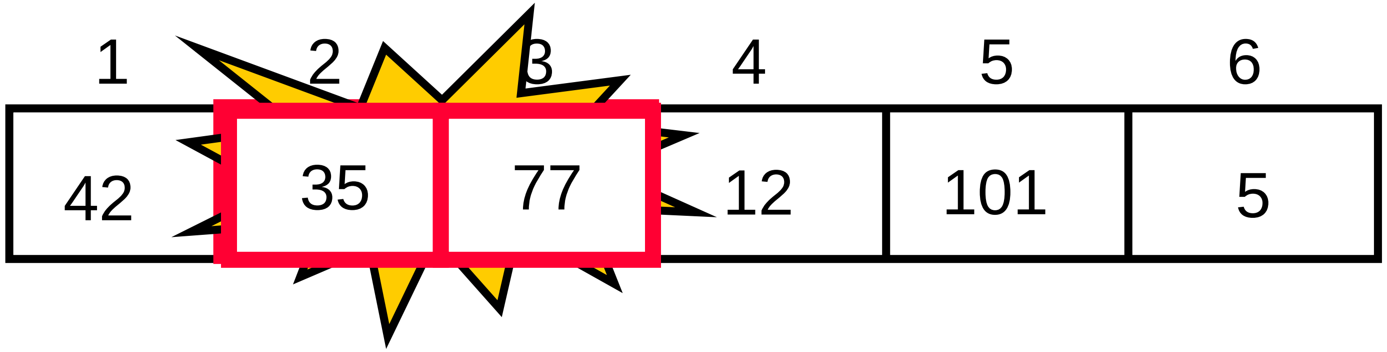
Bubble Sort- Illustration: "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



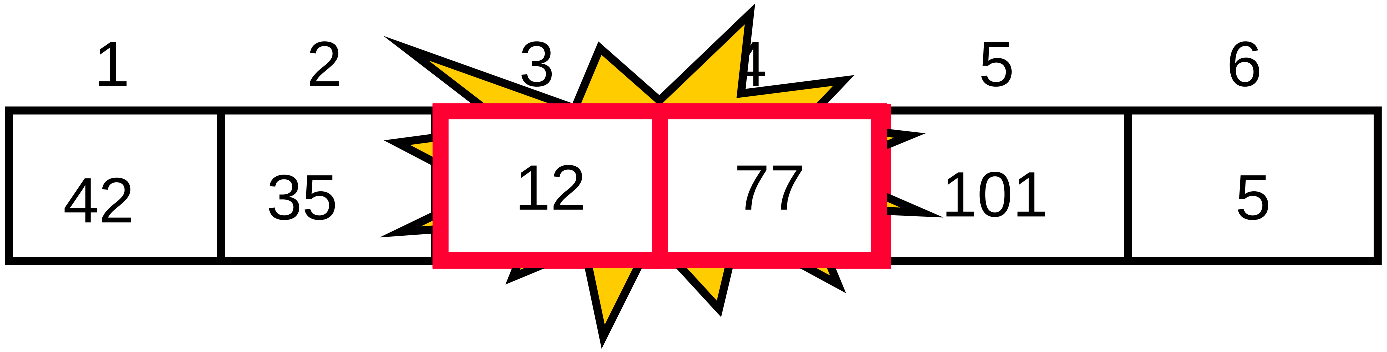
Bubble Sort- Illustration: "Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



Bubble Sort- Illustration: "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



Bubble Sort- Illustration: "Bubbling Up" the Largest Element

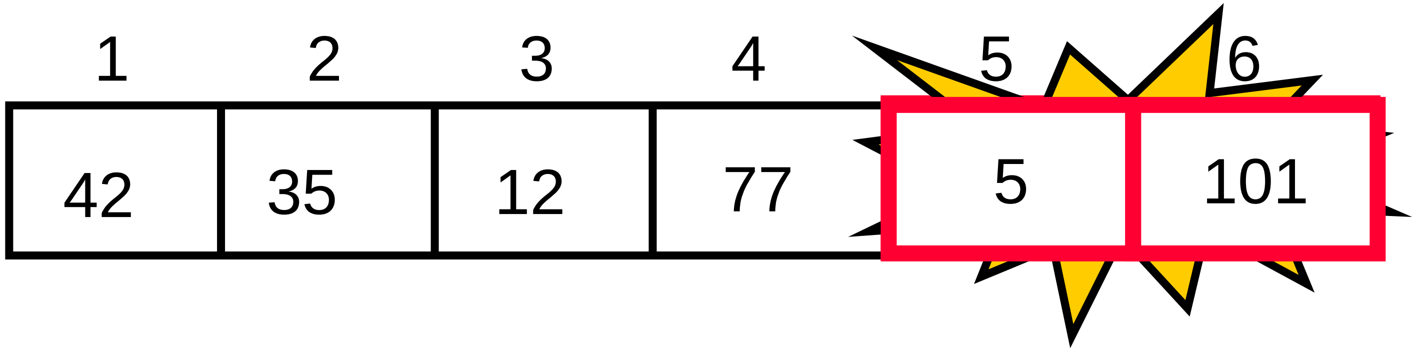
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

1	2	3	4	5	6
42	35	12	77	101	5

No need to swap

Bubble Sort- Illustration: "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



Bubble Sort- Illustration: "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

- **This is called "first pass".**

Bubble Sort- Illustration: Repeat "Bubble Up" How Many Times?

- **If we have N elements…**
- **And if each time we bubble an element, we place it in its correct location…**
- **Then we repeat the "bubble up" process N – 1 times.**
- **This guarantees we'll correctly place all N elements.**

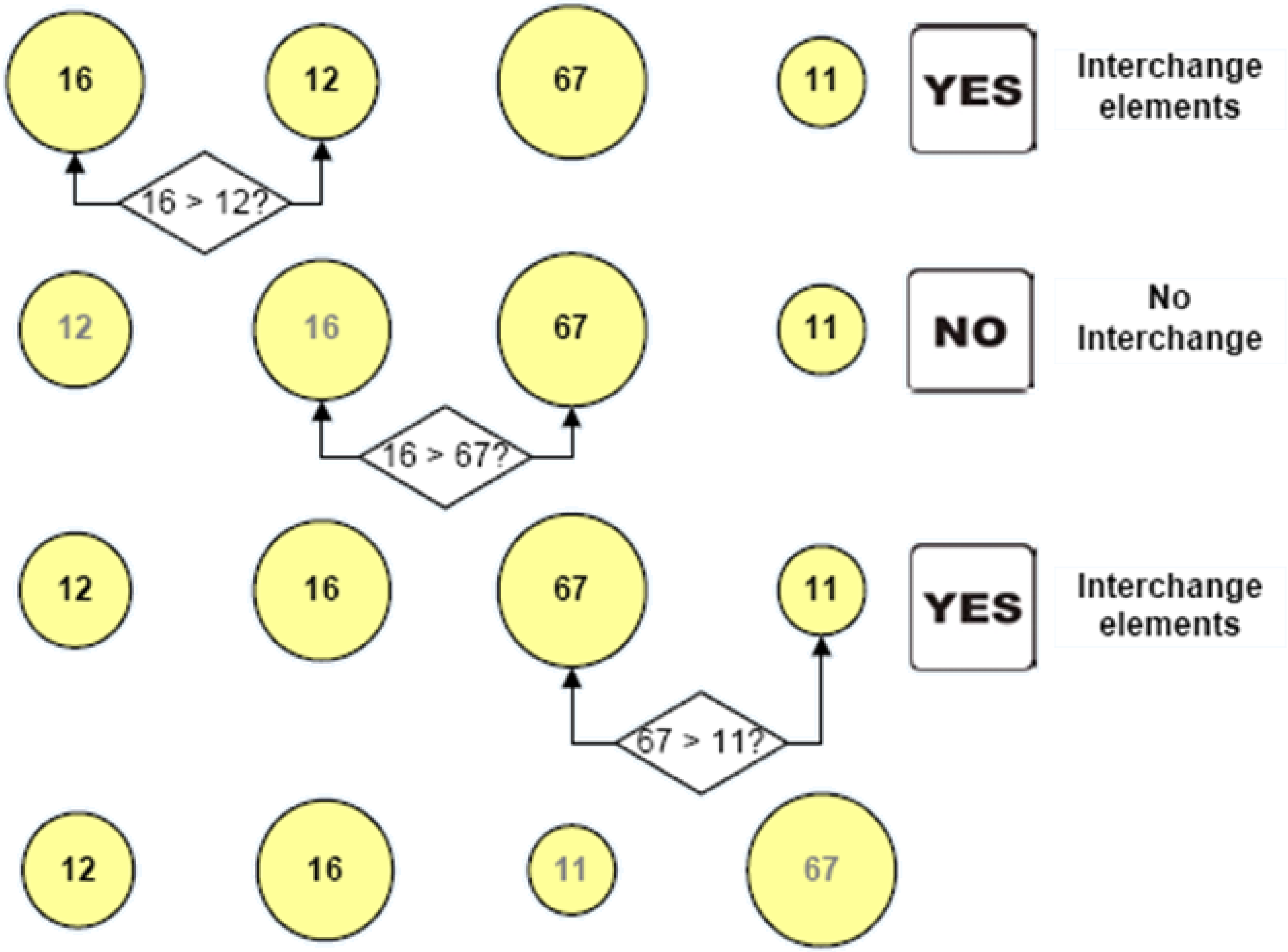
Bubble Sort- Illustration: “Bubbling” All the Elements

N – 1 PASSES

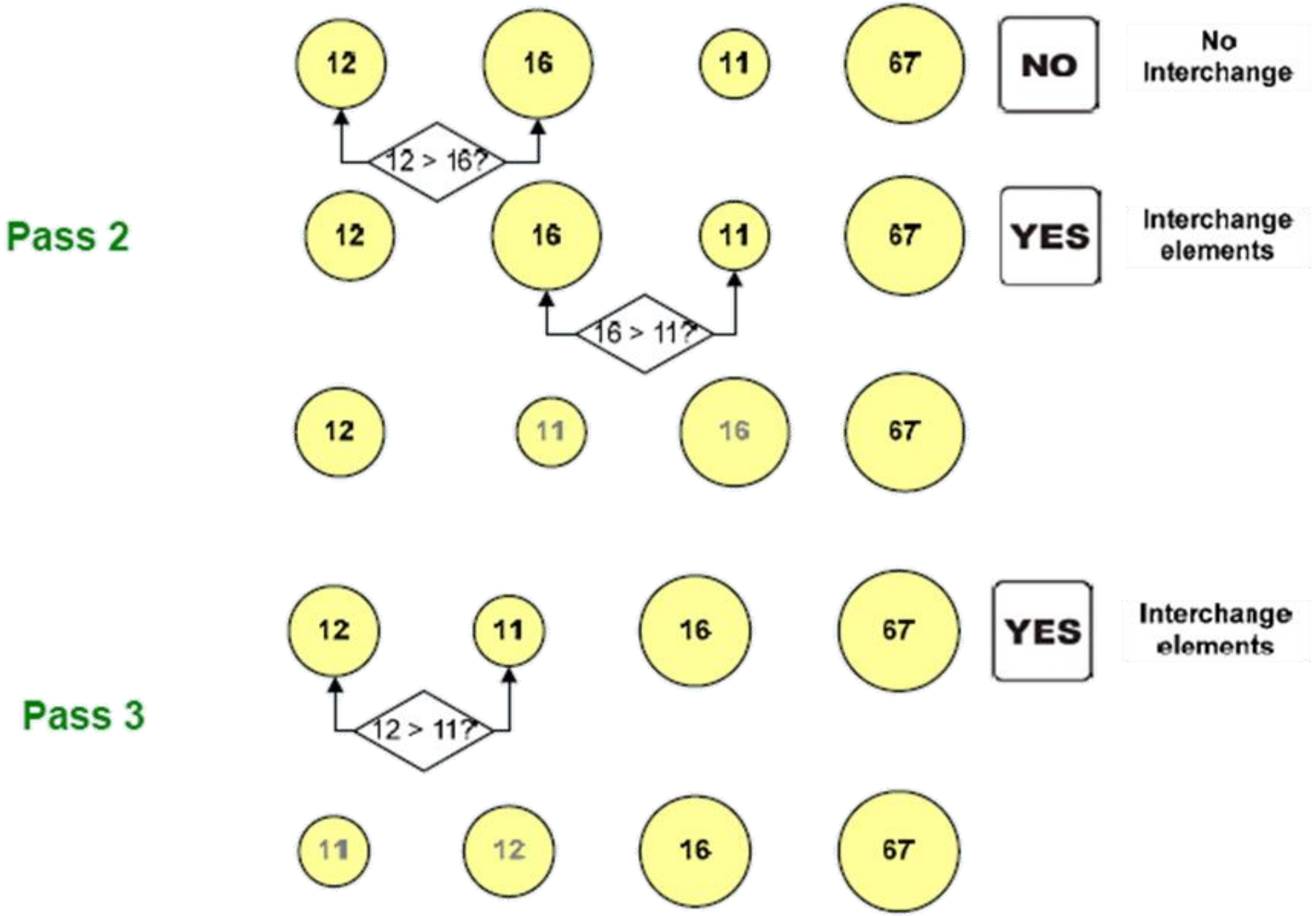
1	2	3	4	5	6	
42	35	12	77	5	101	Pass 1
35	12	42	5	77	101	Pass 2
12	35	5	42	77	101	Pass 3
12	5	35	42	77	101	Pass 4
5	12	35	42	77	101	Pass 5

Bubble Sort- Illustration

Pass 1



Bubble Sort- Illustration



Pseudo code for Bubble Sort procedure

```
for(i=0;i<n;i++)
scanf("%d", &a[i]); // entered elements

 for(i=0;i<n-1;i++) { //pass

    for(j=0;j<n-i-1;j++) {

        if(a[j]>a[j+1]){ // comparison

         // interchange
         temp=a[j];
         a[j]=a[j+1];
         a[j+1]=temp;
        }
    }
}
```

Example :
a[ ]={16, 12, 11, 67}

Array after sorting (ascending)
a[ ]={11, 12, 16, 67}

Bubble Sort- Illustration

6 5 3 1 8 7 2 4

Selection Sort

- Each pass selects the smallest data item from the unsorted set and move it to its position
- **Procedure:**

  Here 'N' indicates the number of data items to be sorted

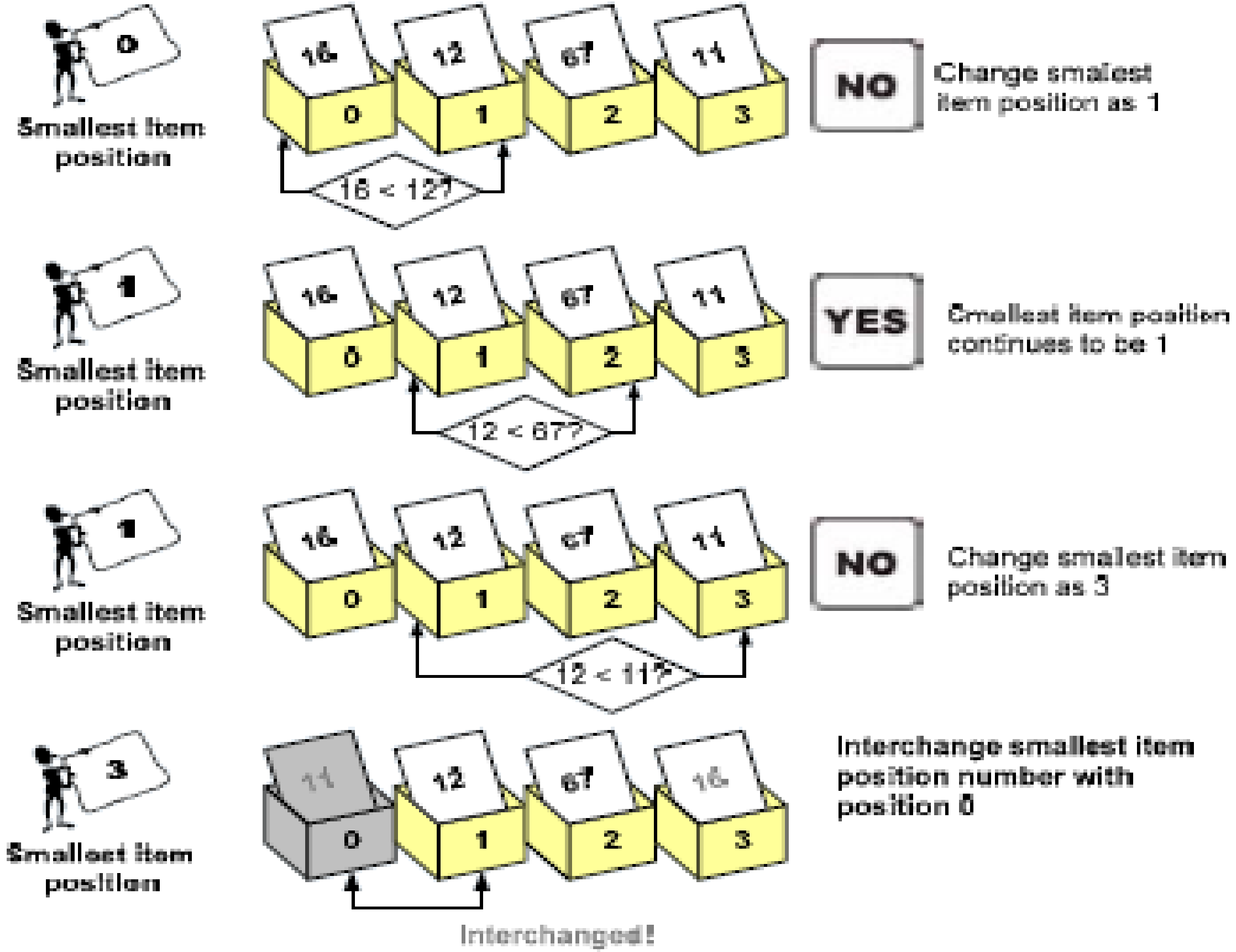
  – **Step 1**: From the data items in positions 0 to N-1, select the smallest data item and interchange with the 0th data item. Now the first data item is sorted

  – **Step 2**: From the data items in positions 1 to N-1, select the smallest data item and interchange with the 1st data item. Now the second data item is sorted

  – **Step 3**: The steps are repeated N-1 times. At the end of N-1 th time the entire data set is sorted

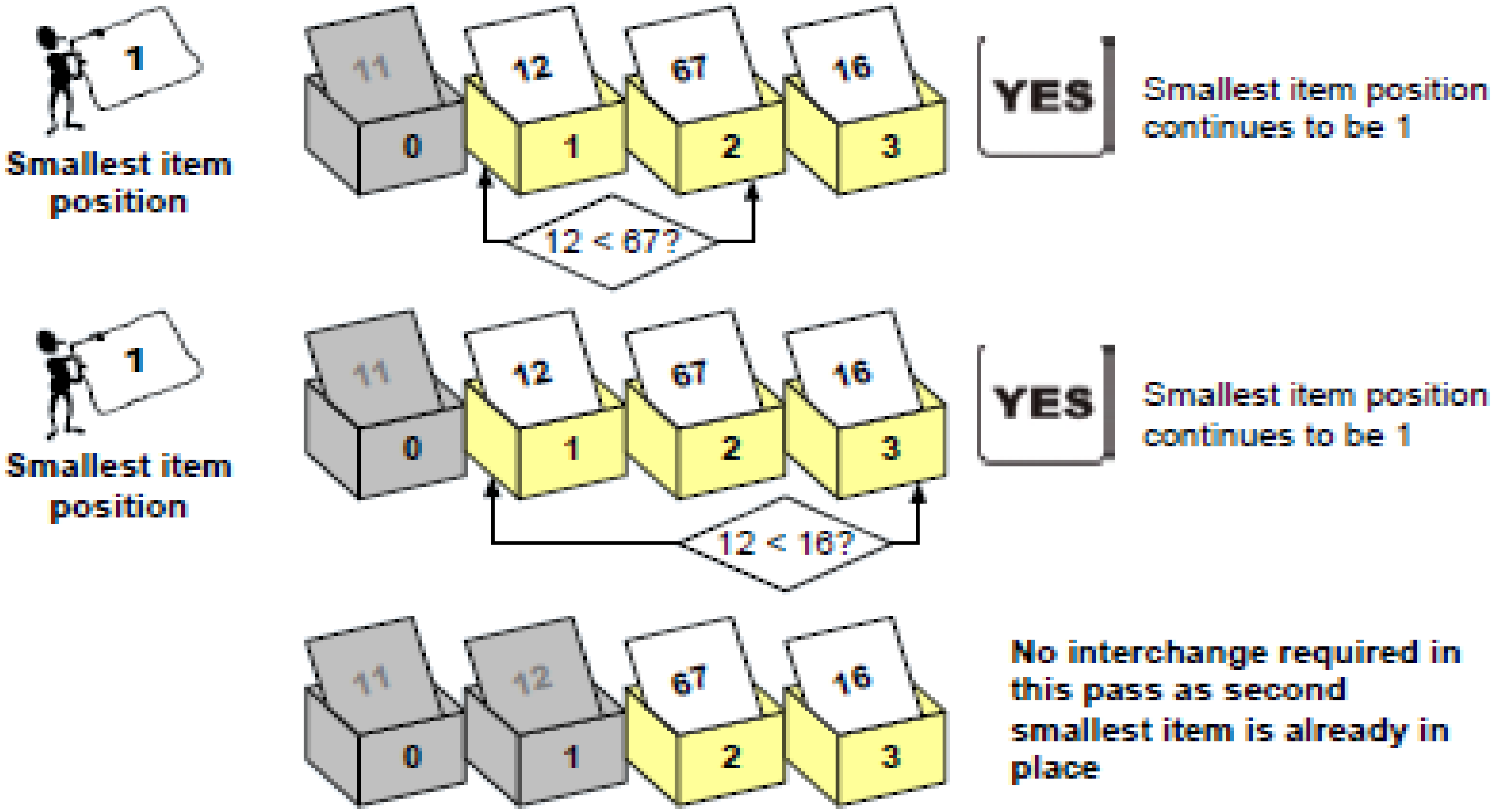
Selection Sort – example-1

Pass 1

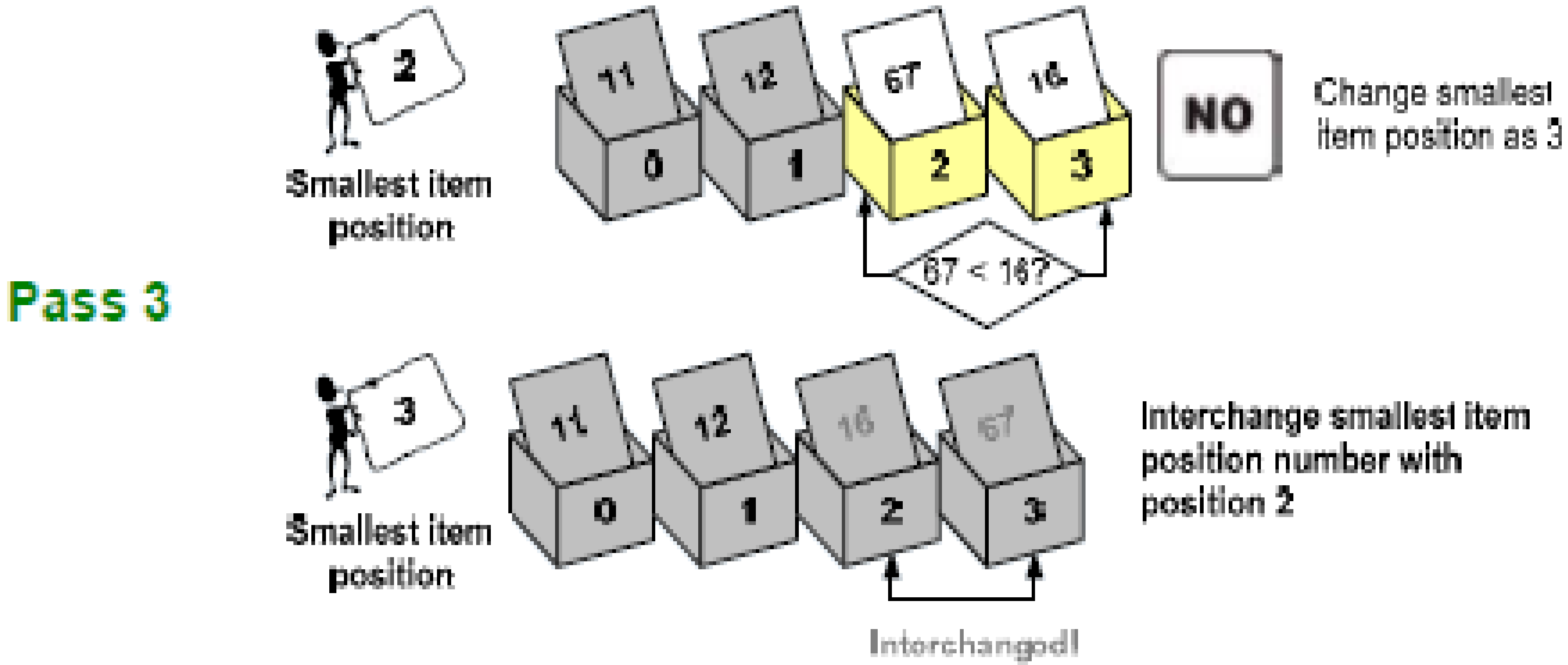


Selection Sort – example-1

Pass 2

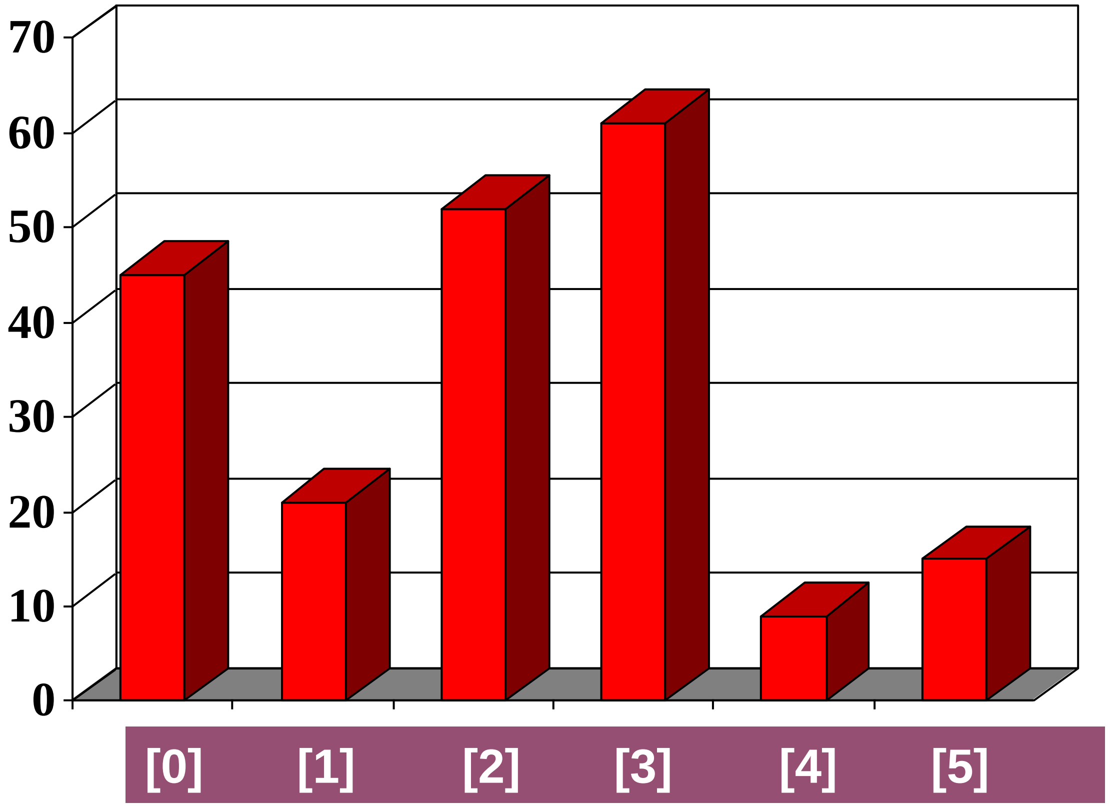


Selection Sort – example-1



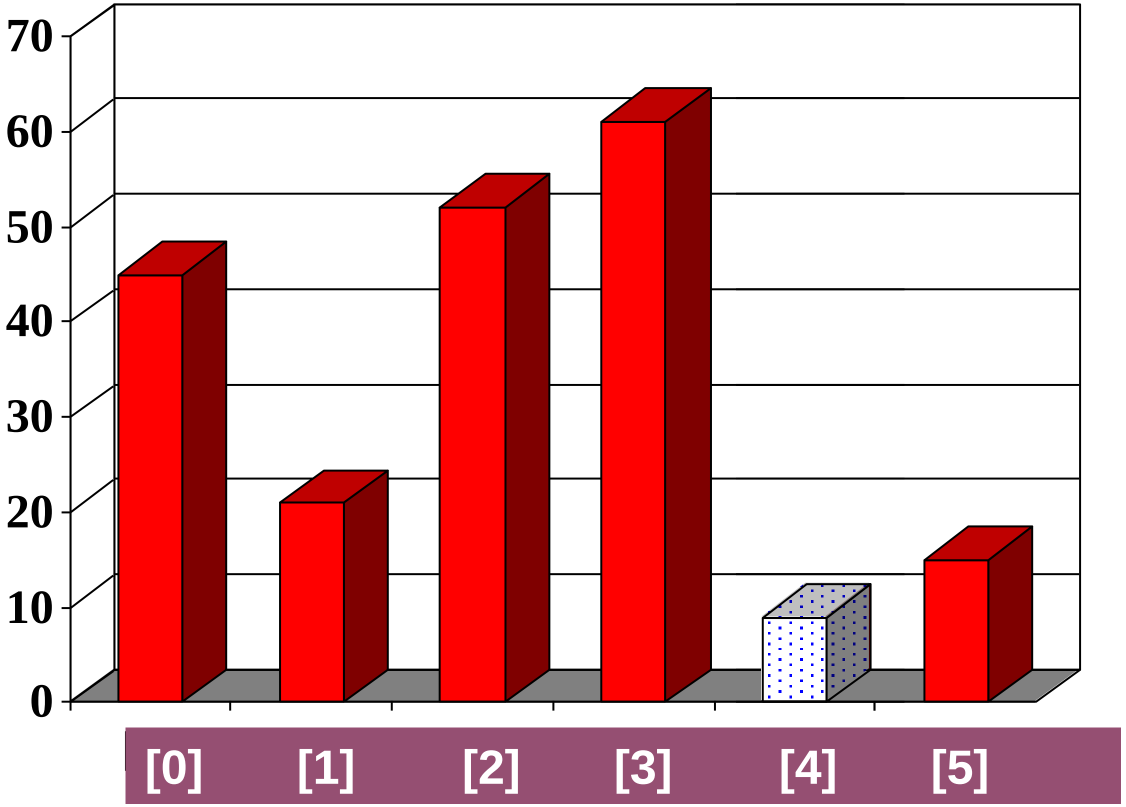
Selection Sort – illustration

- Given a list of six integers that we want to sort from smallest to largest.



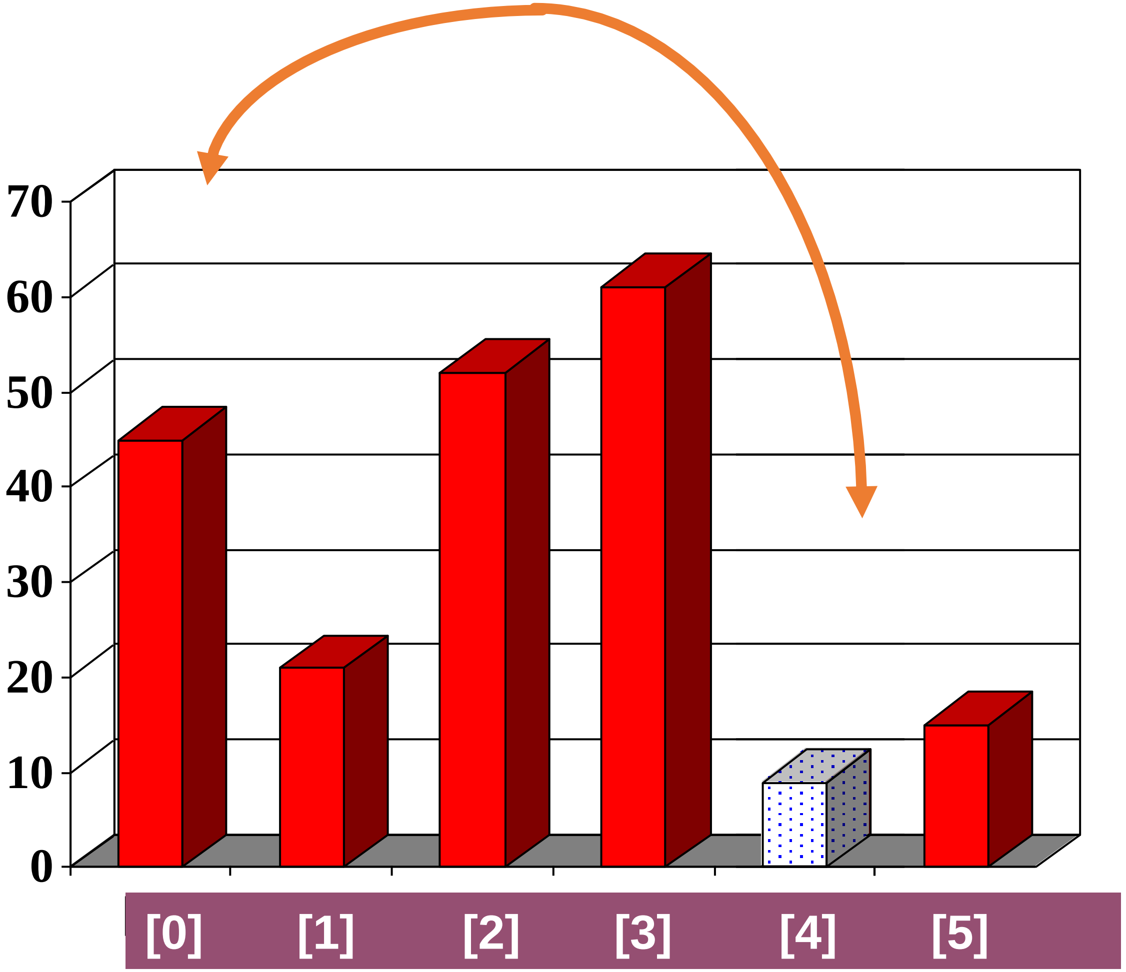
Selection Sort – illustration

- Start by finding the **smallest** entry.



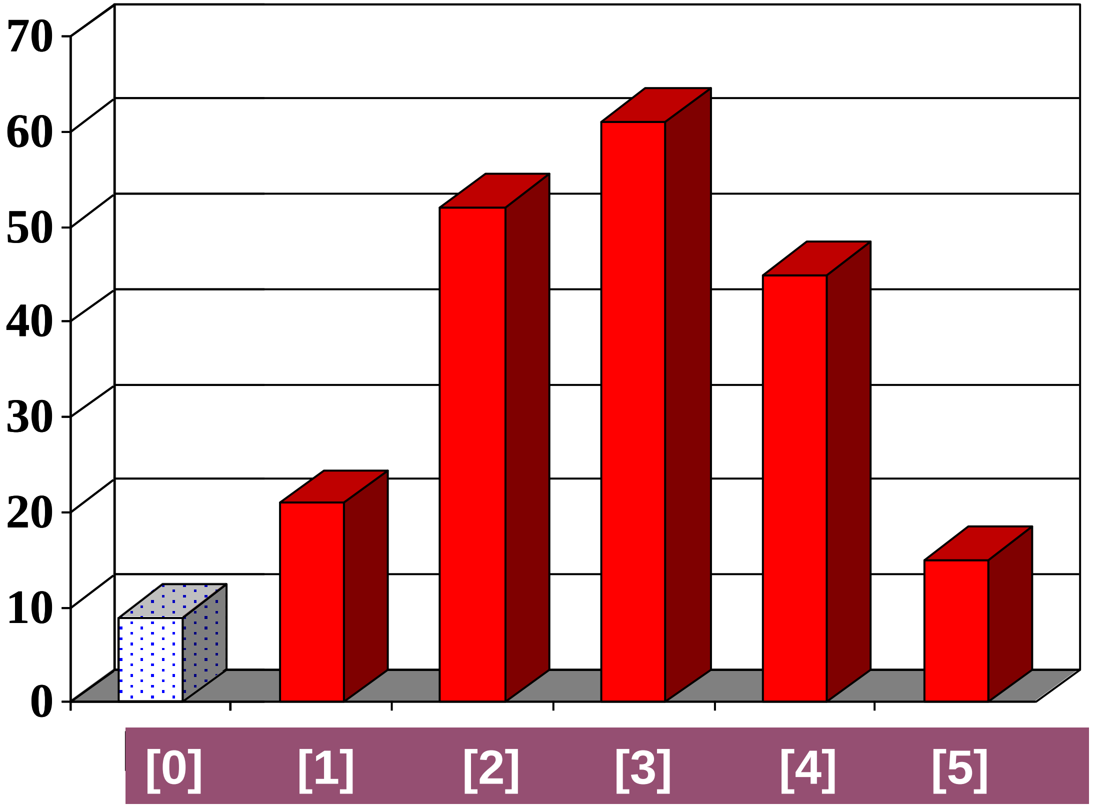
Selection Sort – illustration

- Swap the smallest entry with the first entry.



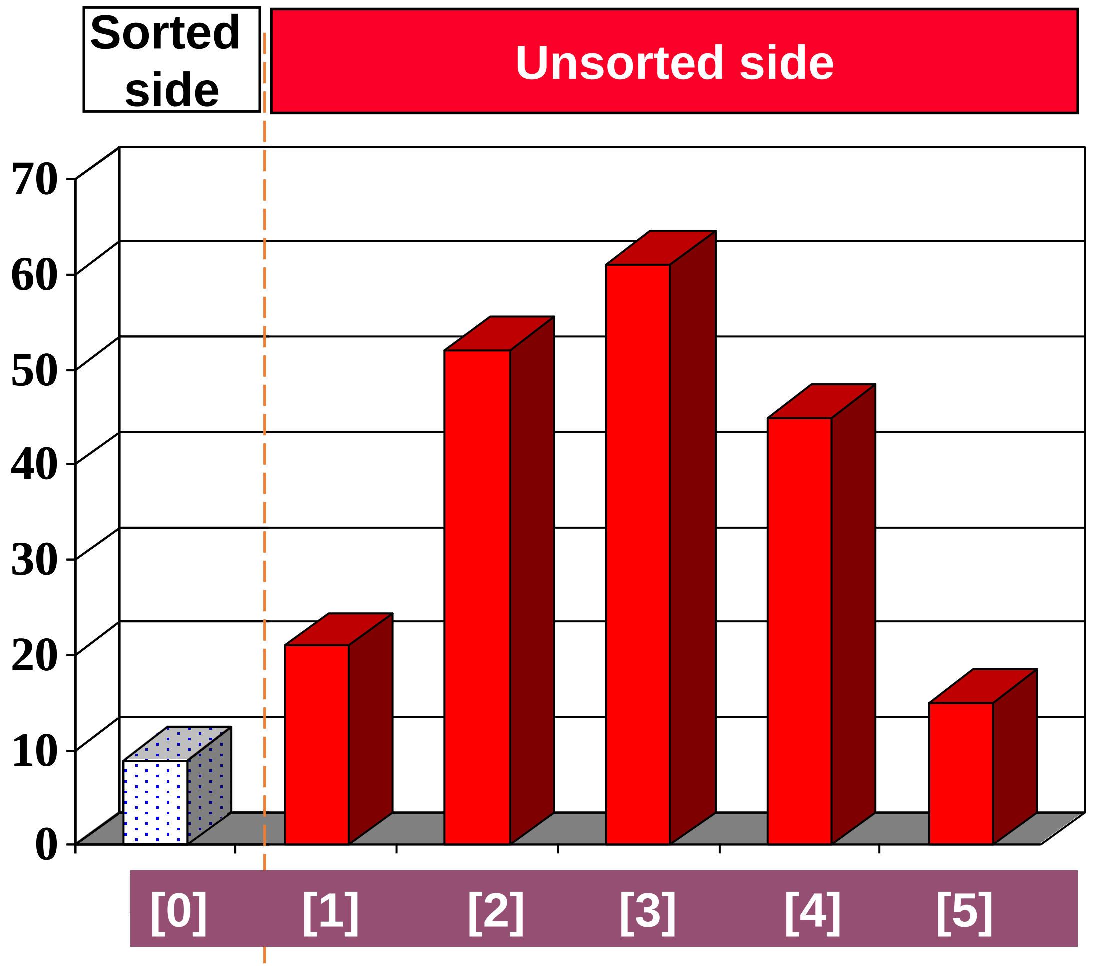
Selection Sort – illustration

- Swap the smallest entry with the first entry.



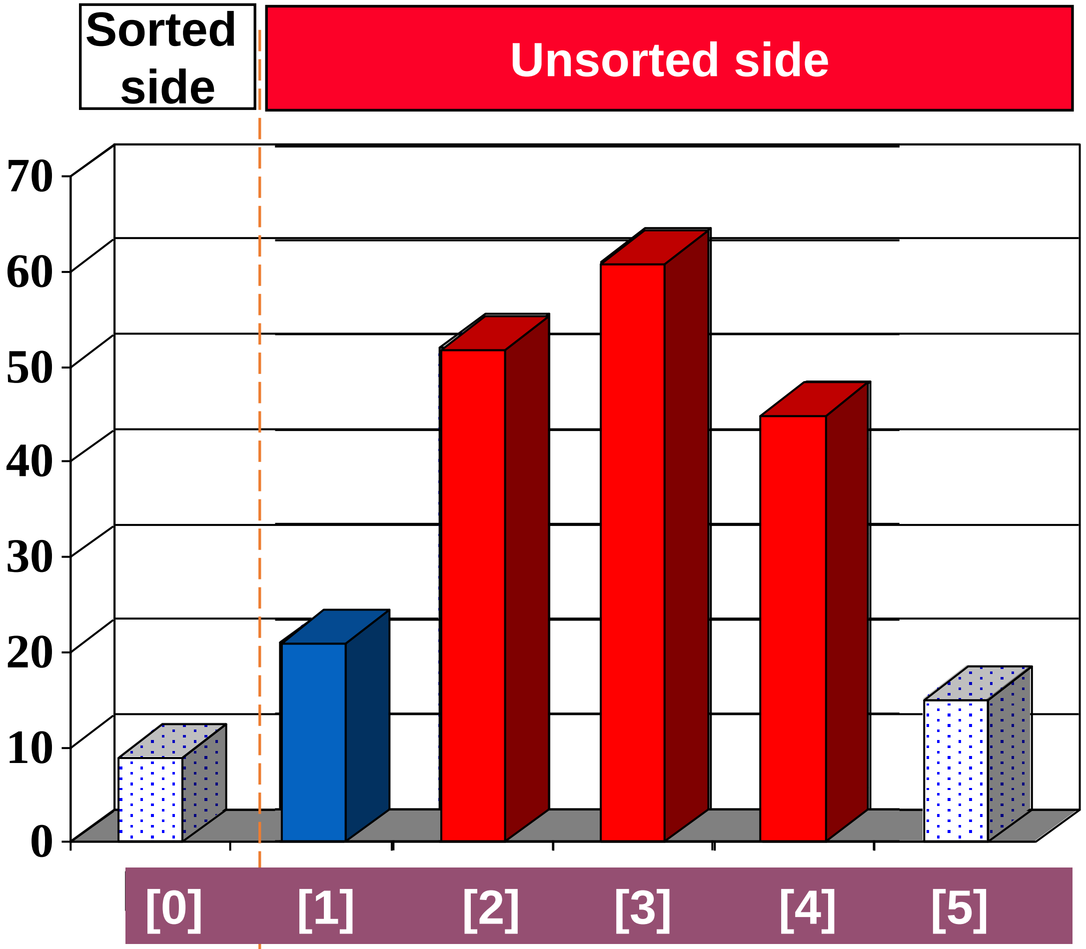
Selection Sort – illustration

- Part of the array is now sorted.



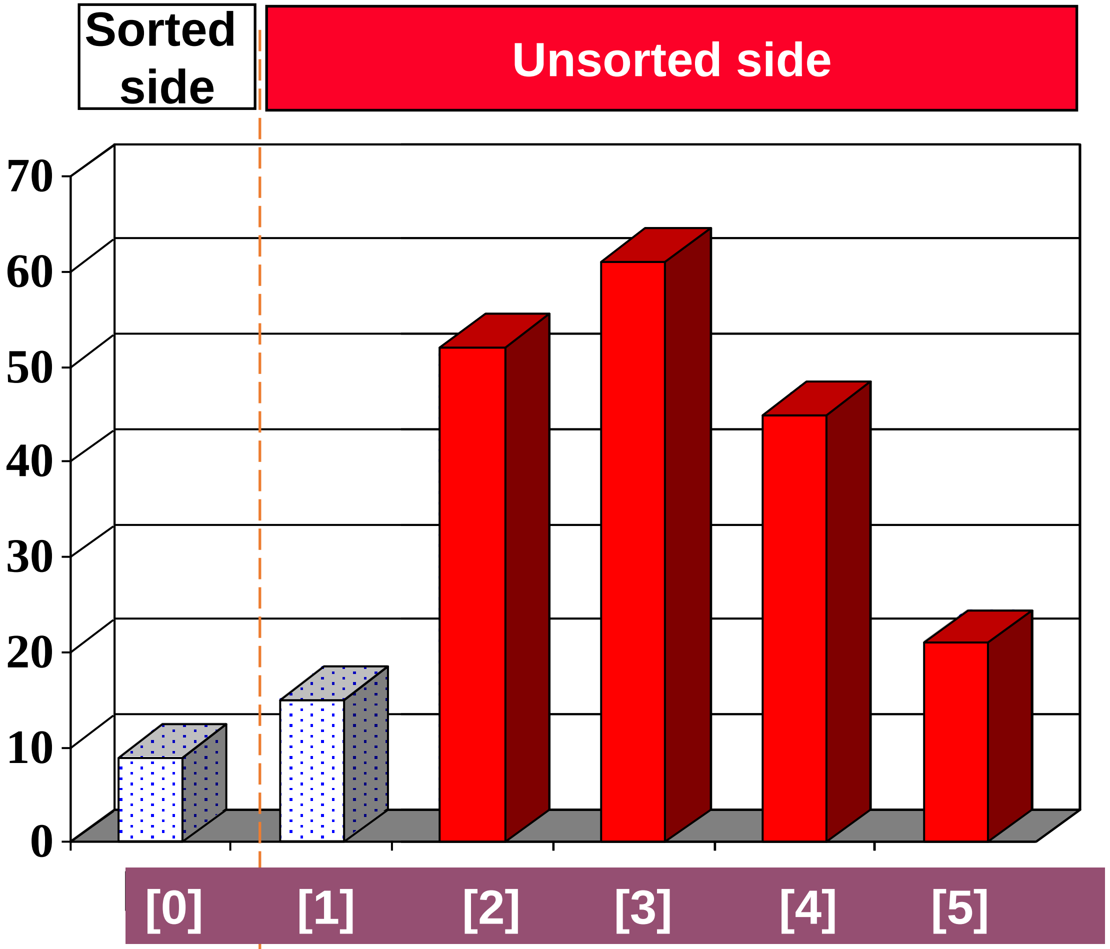
Selection Sort – illustration

- Find the smallest element in the unsorted side.



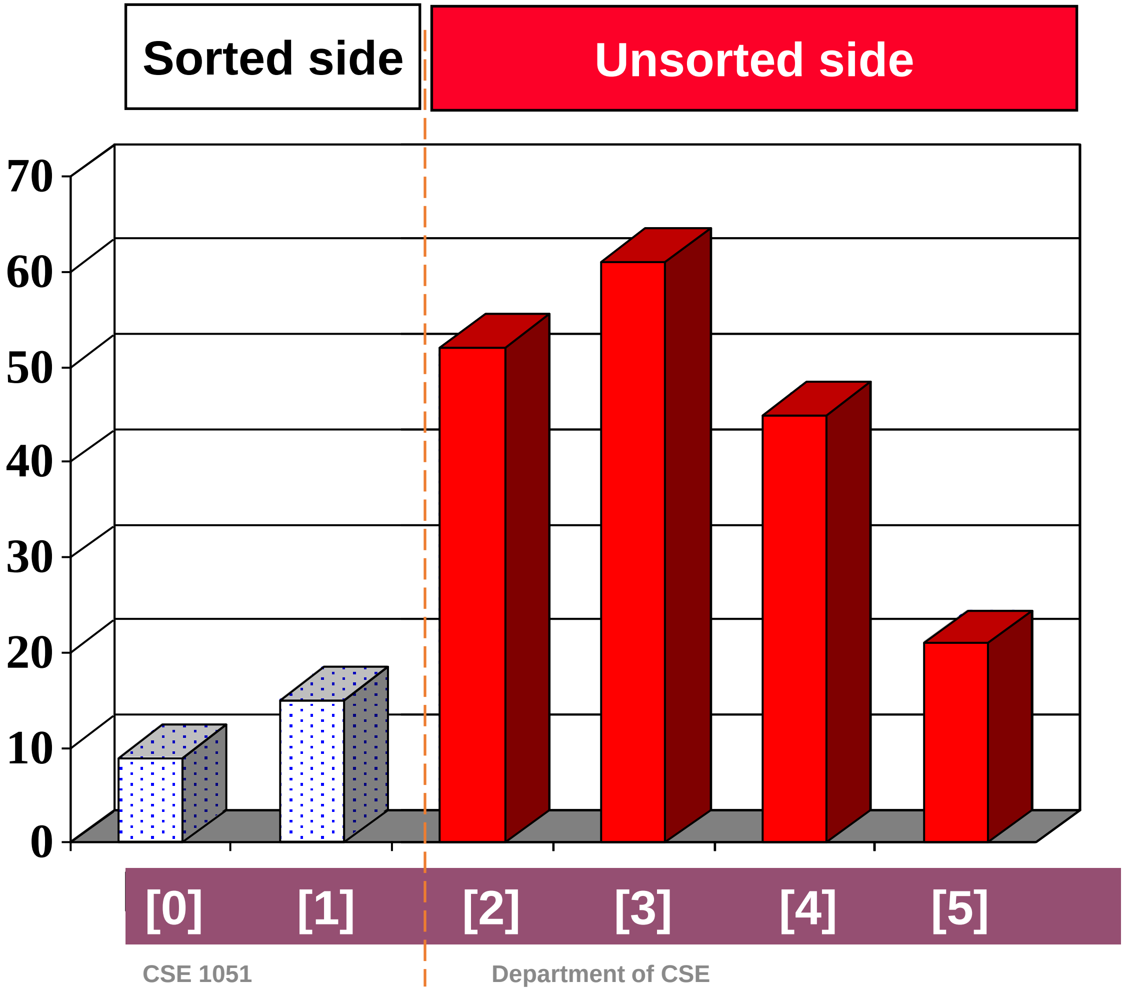
Selection Sort – illustration

- Swap with the front of the unsorted side.



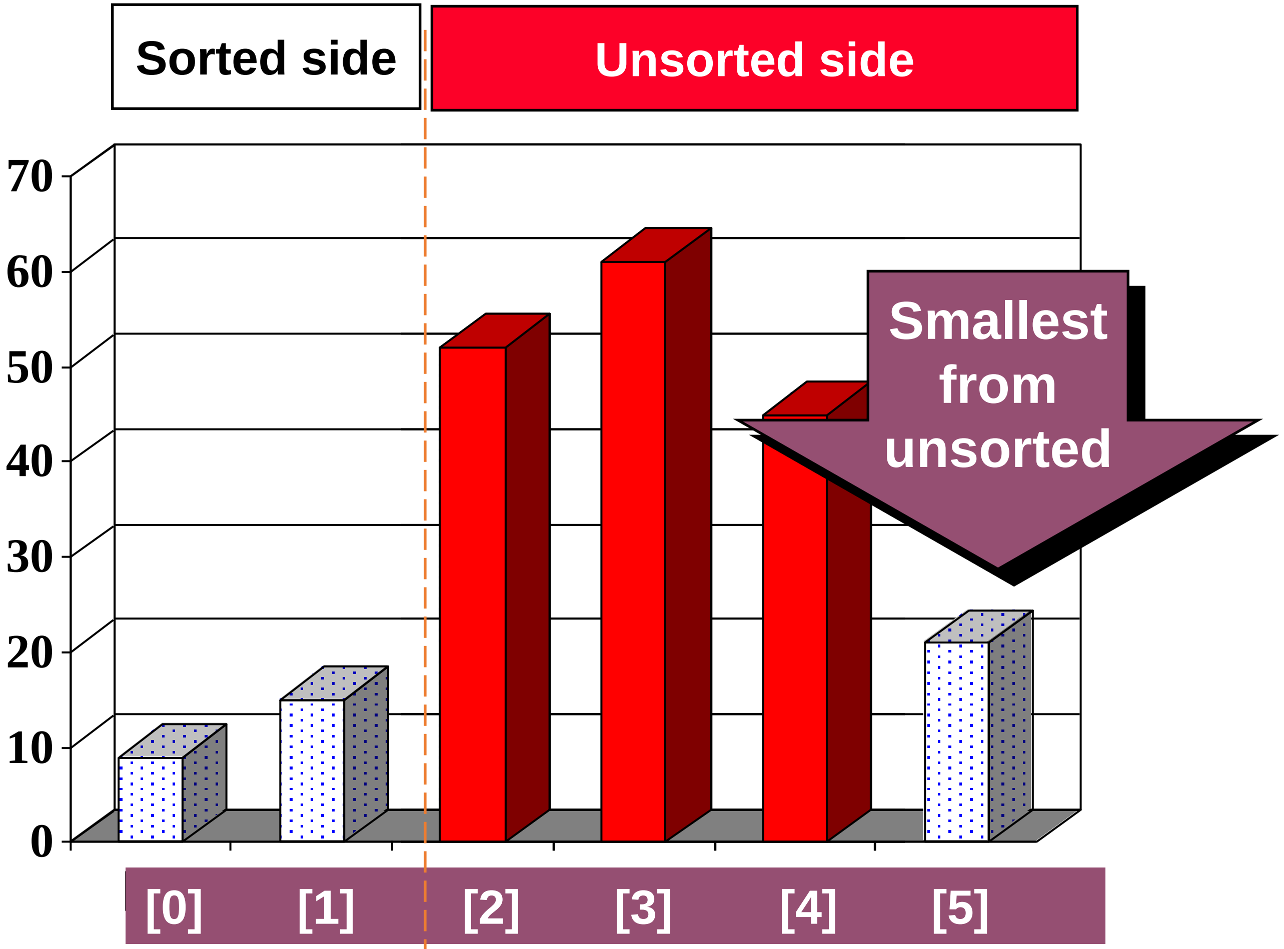
Selection Sort – illustration

- We have increased the size of the sorted side by one element.



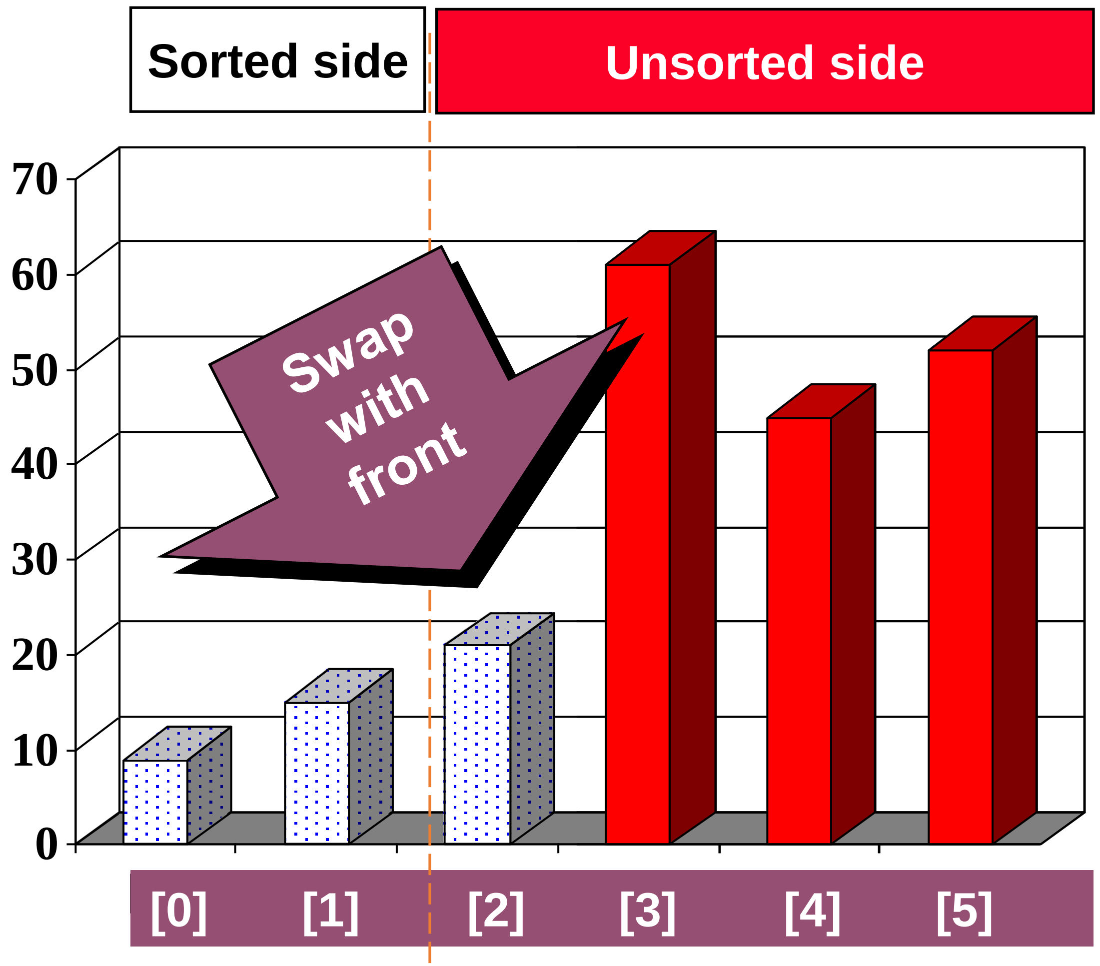
Selection Sort – illustration

- The process continues...



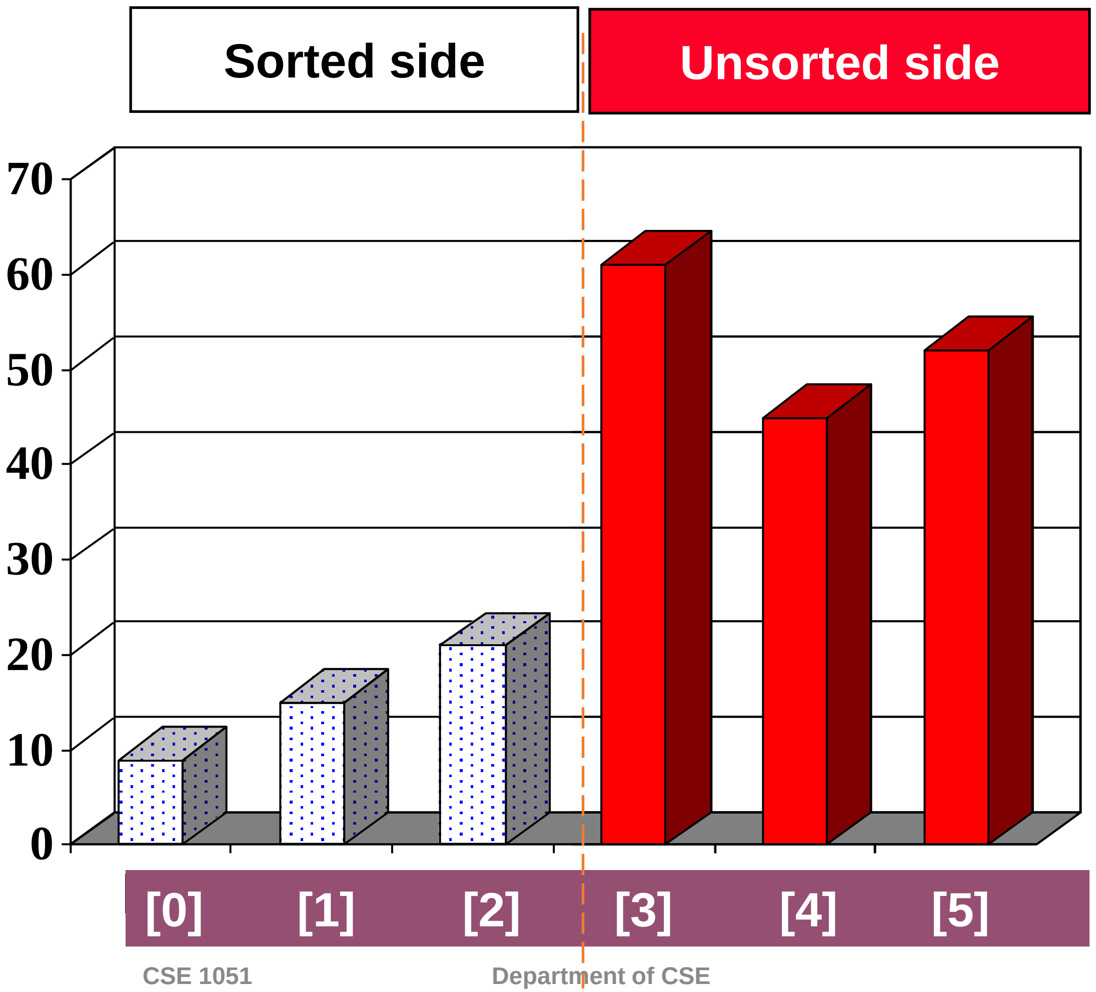
Selection Sort – illustration

- The process continues...



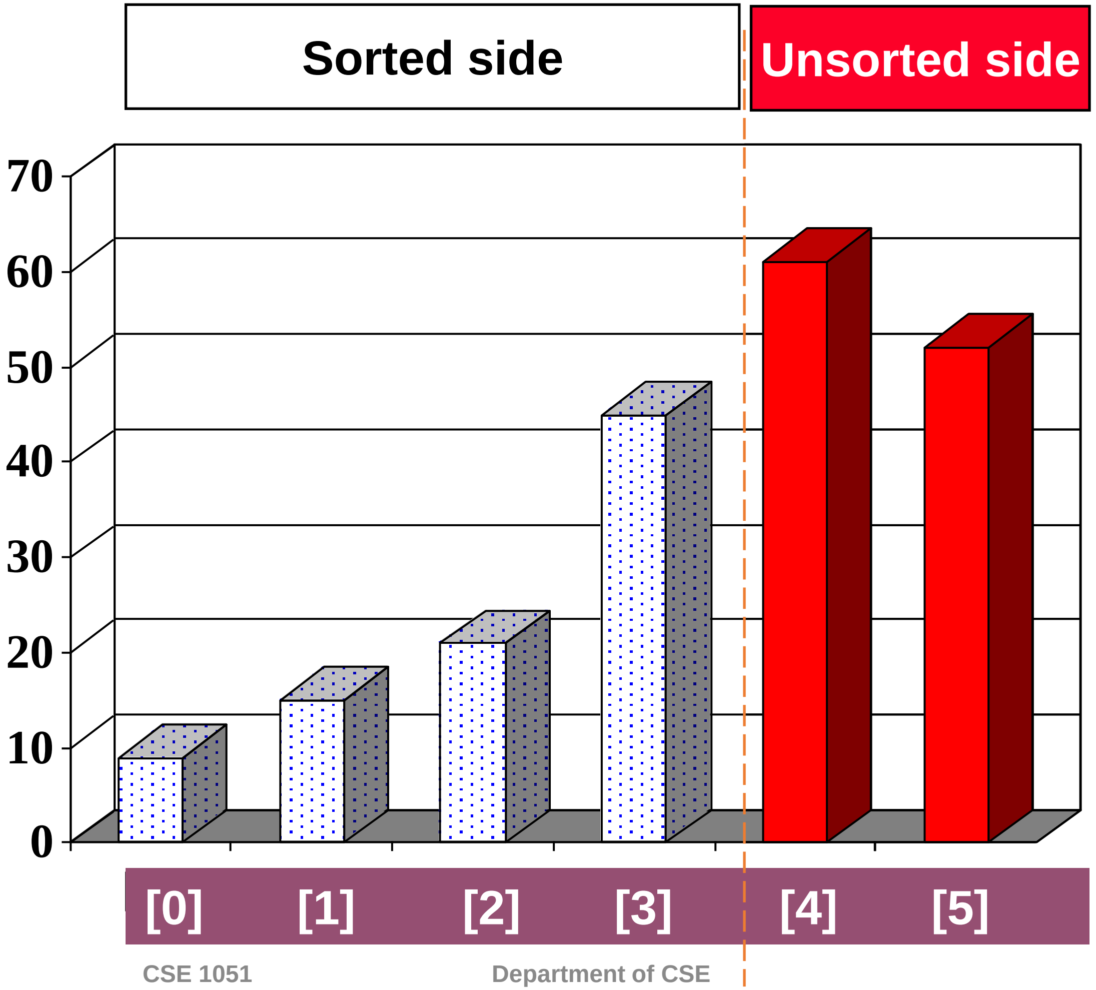
Selection Sort – illustration

- The process continues...



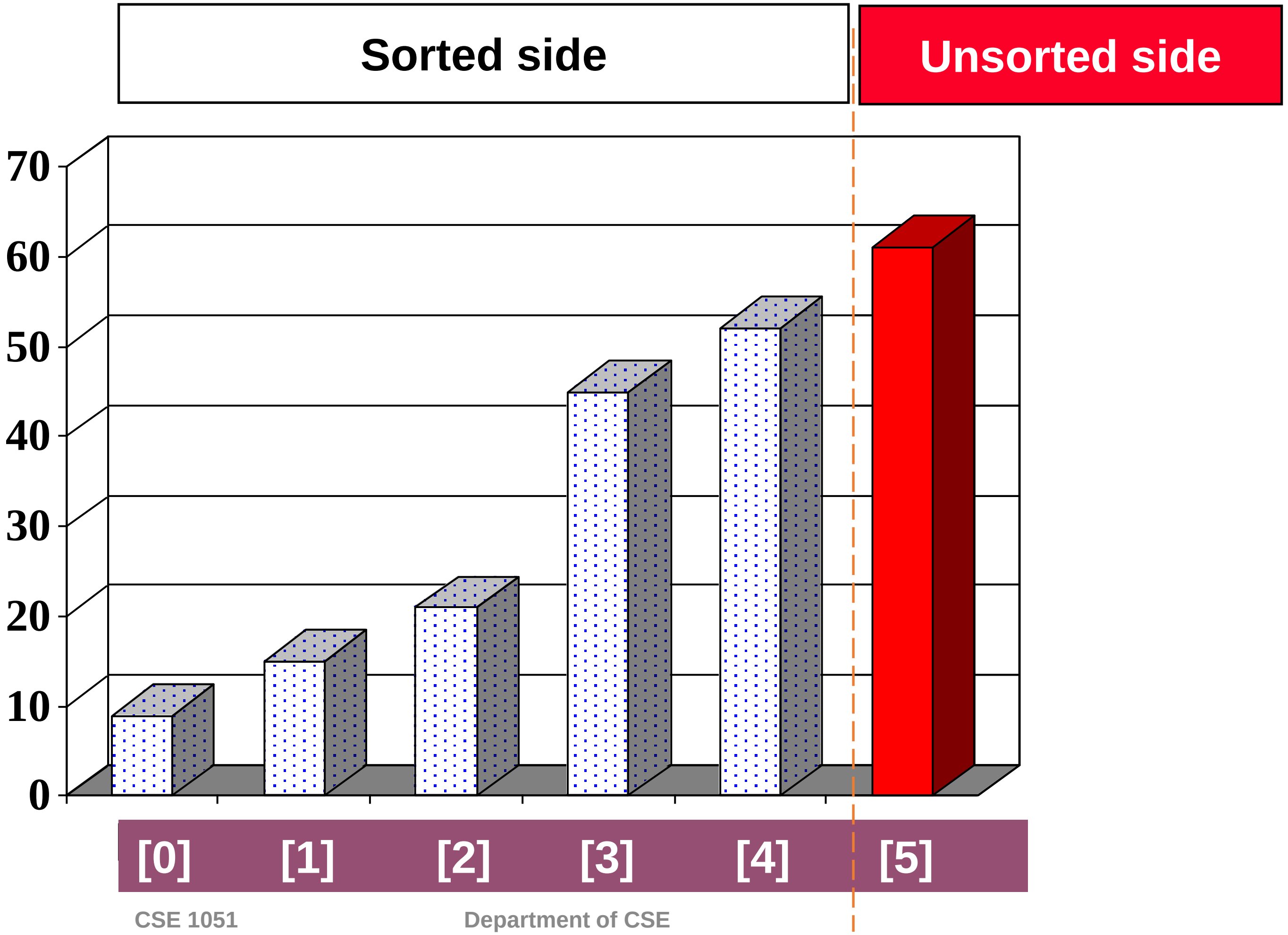
Selection Sort – illustration

- The process keeps adding one more number to the sorted side.
- The sorted side has the smallest numbers, arranged from small to large.



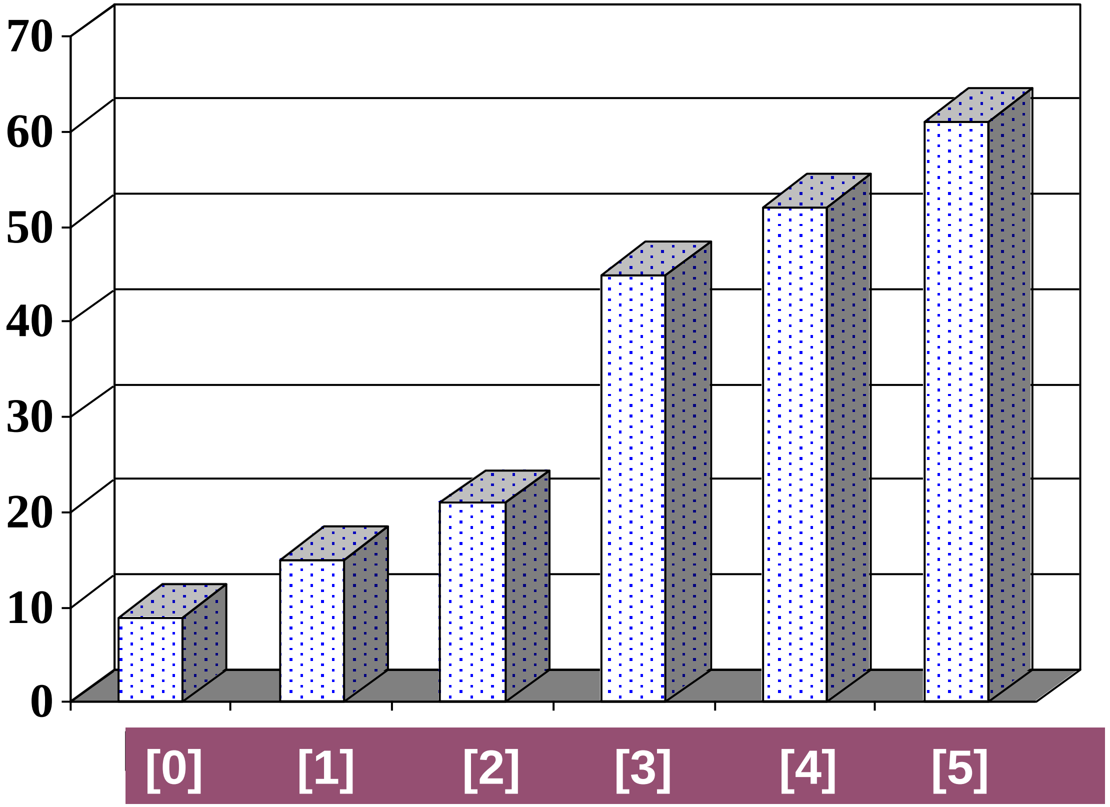
Selection Sort – illustration

- We can stop when the unsorted side has just one number, since that number must be the largest number.



Selection Sort – illustration

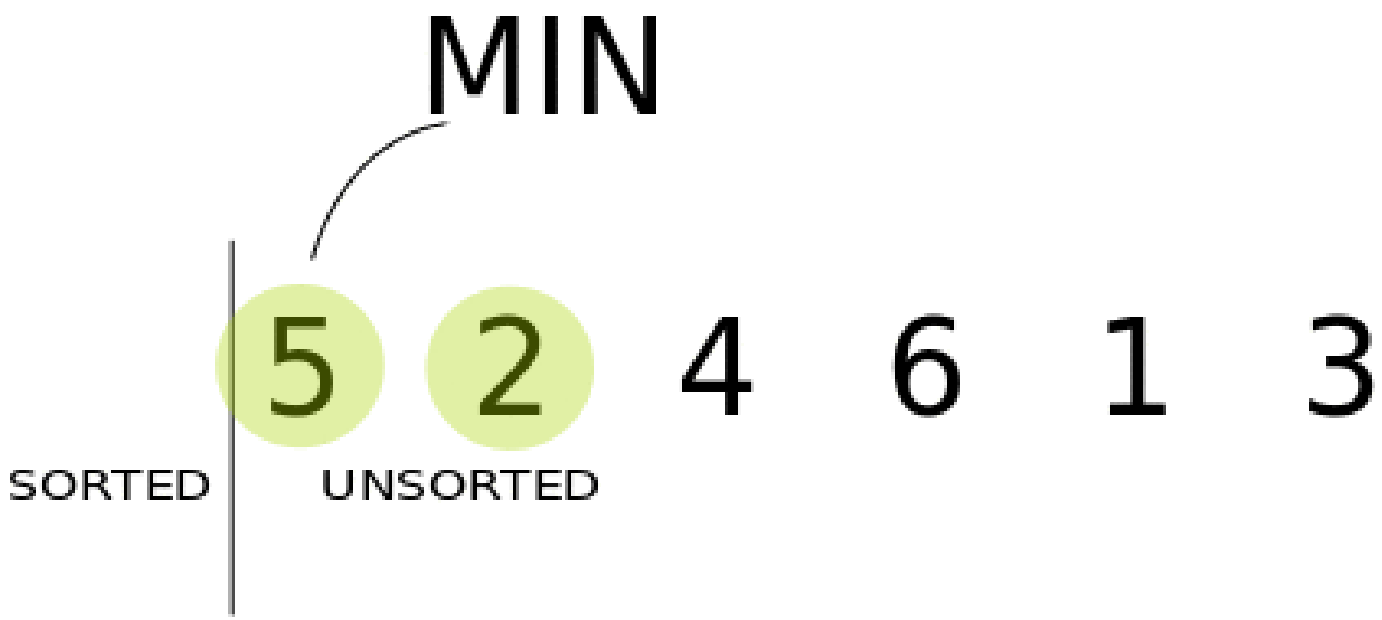
- The array is now sorted.
- We repeatedly **selected** the smallest element, and moved this element to the front of the unsorted side.



Selection Sort – *procedure*

```
for(i = 0; i < n-1; i++) {  // loop for number of pass
   pos = i; small = a[i];
      for(j=i+1; j<n; j++) {  //loop for searching the smallest
      if(small > a[j]) {  // finding the smallest
       pos = j;  // pos for interchanging
       small = a[j];  // assigning current small value
       }
      }
a[pos] = a[i];  //interchanging values
a[i] = small;
}
```

Selection Sort – illustration



Comparison between Bubble Sort and Selection Sort

Basis for Comparison	Bubble Sort	Selection Sort
Basic	Adjacent element is compared and then swapped	Smallest element is selected and swapped with the first element (in case of ascending order)
Efficiency	Inefficient	Improved efficiency when compared to Bubble sort
Method	Exchanging	selection
Speed	Slow	Fast when compared to Bubble sort

Tutorial problems on Sorting and searching

- **Write a program in C to implement binary search.**
-

Summary

- **Sorting Techniques**
  - Bubble Sort
  - Selection Sort