Searching and Sorting

Commonly used methods

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Search

Searching

Check if a given element (called *key*) occurs in the array.

- Example: array of student records; rollno can be the key.

Two methods to be discussed:

- If the array elements are unsorted.
  - Linear search
- If the array elements are sorted.
  - Binary search

Basic Concept of Linear Search

Basic idea:

- **Start at the beginning of the array.**
- **Inspect elements one by one to see if it matches the key.**

Time complexity:

- **A measure of how long an algorithm takes to run.**
- **If there are *n* elements in the array:**
  - **Best case:**

    match found in first element (1 search operation)
  - **Worst case:**

    no match found, or match found in the last element (n search operations)
  - **Average case:** *(n + 1) / 2 search operations*

Linear Search

/* The function returns the array index where the match is found. It returns -1 if there is no match.   */

```
int  linear_search (int a[], int size, int key)
{
        int pos = 0;
        while ((pos < size) && (a[pos] != key))  pos++;
        if (pos < size)
                return pos;         /* Return the position of match */
        return -1;                  /* No match found */
}
```

Binary Search

Basic Concept

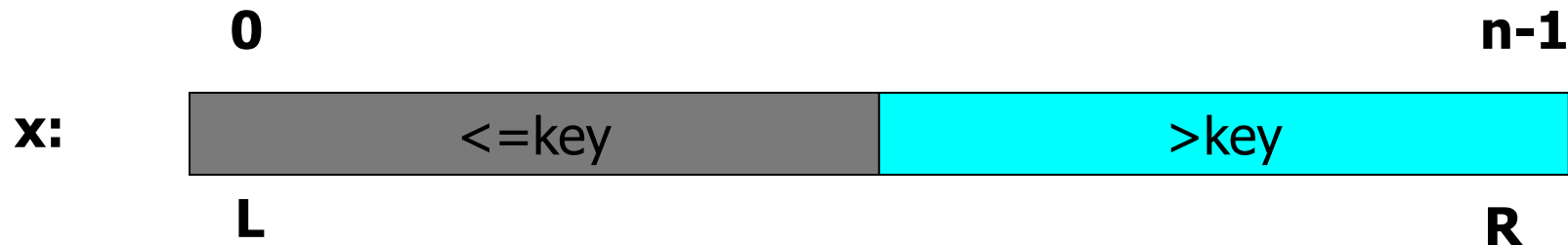Binary search is applicable if the array is *sorted*.

BASIC IDEA

- **Look for the target in the middle.**
- **If you don't find it, you can ignore half of the array, and repeat the process with the other half.**

In every step, we reduce the number of elements to search in by half.

The Basic Strategy

What do we want?

- **Find split between values larger and smaller than key:**

- **Situation while searching:**
  - **Initially L and R contains the indices of first and last elements.**

- **Look at the element at index [(L+R)/2].**
  - **Move L or R to the middle depending on the outcome of test.**

Binary Search

/* If key appears in x[0..size-1], return its location, pos such that x[pos]==key.
If not found, return -1 */

```
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;

        _____ ;
        while ( _____ )
        {

                        _____;
        }

        _____ ;
}
```

The basic search iteration

```
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;

        _____;
        while ( _____ )
        {
                mid = (L + R) / 2;
                if (x[mid] <= key)  L = mid;
                else R = mid;
        }
                _____ ;
}
```

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Loop termination criterion

```
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;

        _____;
        while ( L+1 != R )
        {
                mid = (L + R) / 2;
                if (x[mid] <= key) L = mid;
                else R = mid;
        }

        _____ ;
}
```

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Initialization and Return Value

```c
int bin_search (int x[ ], int size, int key)
{
        int L, R, mid;
        L = –1;   R = size;
        while ( L+1 != R )
        {
                mid = (L + R) / 2;
                if (x[mid] <= key) L = mid;
                else R = mid;
        }
        if (L >= 0 && x[L] == key) return L;
        else return –1;
}
```

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Binary Search Examples

Sorted array

-17 -5 3 6 12 21 45 63 50

Trace :

binsearch (x, 9, 3);

binsearch (x, 9, 145);

binsearch (x, 9, 45);

L= –1;   R=9;   x[4]=12;
L= –1;   R=4;   x[1]= –5;
L= 1;   R=4;   x[2]=3;
L=2;   R=4;   x[3]=6;
L=2;   R=3;   return L;

We may modify the algorithm by checking equality with x[mid].

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Is it worth the trouble ?

Suppose that the array x has 1000 elements.

Ordinary search

– **If *key* is a member of x, it would require 500 comparisons on the average.**

Binary search

- **after 1st compare, left with 500 elements.**
- **after 2nd compare, left with 250 elements.**
- **After at most 10 steps, you are done.**

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Time Complexity

If there are n elements in the array.

- **Number of iterations required:**

  $$\log_2 n$$

For n = 64 (say).

- **Initially, list size = 64.**
- **After first compare, list size = 32.**
- **After second compare, list size = 16.**
- **After third compare, list size = 8.**
- **…….**
- **After sixth compare, list size = 1.**

$2^k = n$**, where k is the number of steps.**

$\log_2 64 = 6$

$\log_2 1024 = 10$

Exercise

Write a recursive version of the Binary Search function.

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Sorting

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

The Basic Problem

Given an array: x[0], x[1], ... , x[size-1] reorder entries so that

<p style="color:red">x[0] <= x[1] <= . . . <= x[size-1]</p>

- **List is in non-decreasing order.**

We can also sort a list of elements in non-increasing order.

Example

Original list:

    10, 30, 20, 80, 70, 10, 60, 40, 70

Sorted in non-decreasing order:

    10, 10, 20, 30, 40, 60, 70, 70, 80

Sorted in non-increasing order:

    80, 70, 70, 60, 40, 30, 20, 10, 10

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

SELECTION SORT

General situation :

0   **k**   **size-1**

x: | smallest elements, sorted | remainder, unsorted |

Step :

- **Find smallest element, mval, in x[k..size-1]**
- **Swap smallest element with x[k], then increase k.**

0   **k**   **mval**   **size-1**

swap
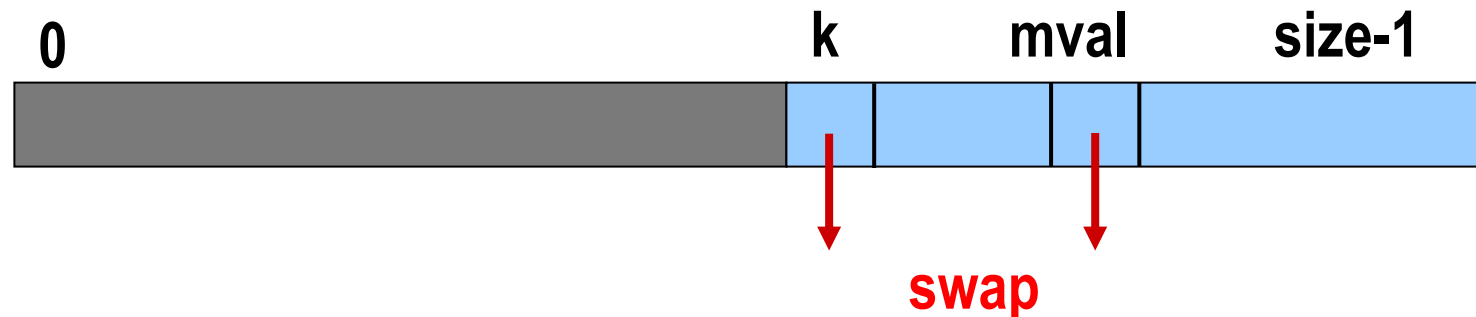
Subproblem

/* Find index of smallest element in x[k..size-1] */

```c
int min_loc (int x[ ], int k, int size)
{
        int j, pos;

        pos = k;
        for (j=k+1; j<size; j++)
                    if ( x[ j ] < x[ pos ] ) pos = j;
        return pos;
}
```

Selection Sort Function

/* Sort x[0..size-1] in non-decreasing order */

```c
int sel_sort (int x[ ], int size)
{
        int k, m;

        for (k=0; k<size-1; k++)
        {
                m = min_loc (x, k, size);
                temp = a[ k ]; a[ k ] = a[ m ]; a[ m ] = temp;  /* Exchange */
        }
}
```

Example

Analysis

How many steps are needed to sort *n* items ?

- **Total number of steps *proportional* to n^2.**

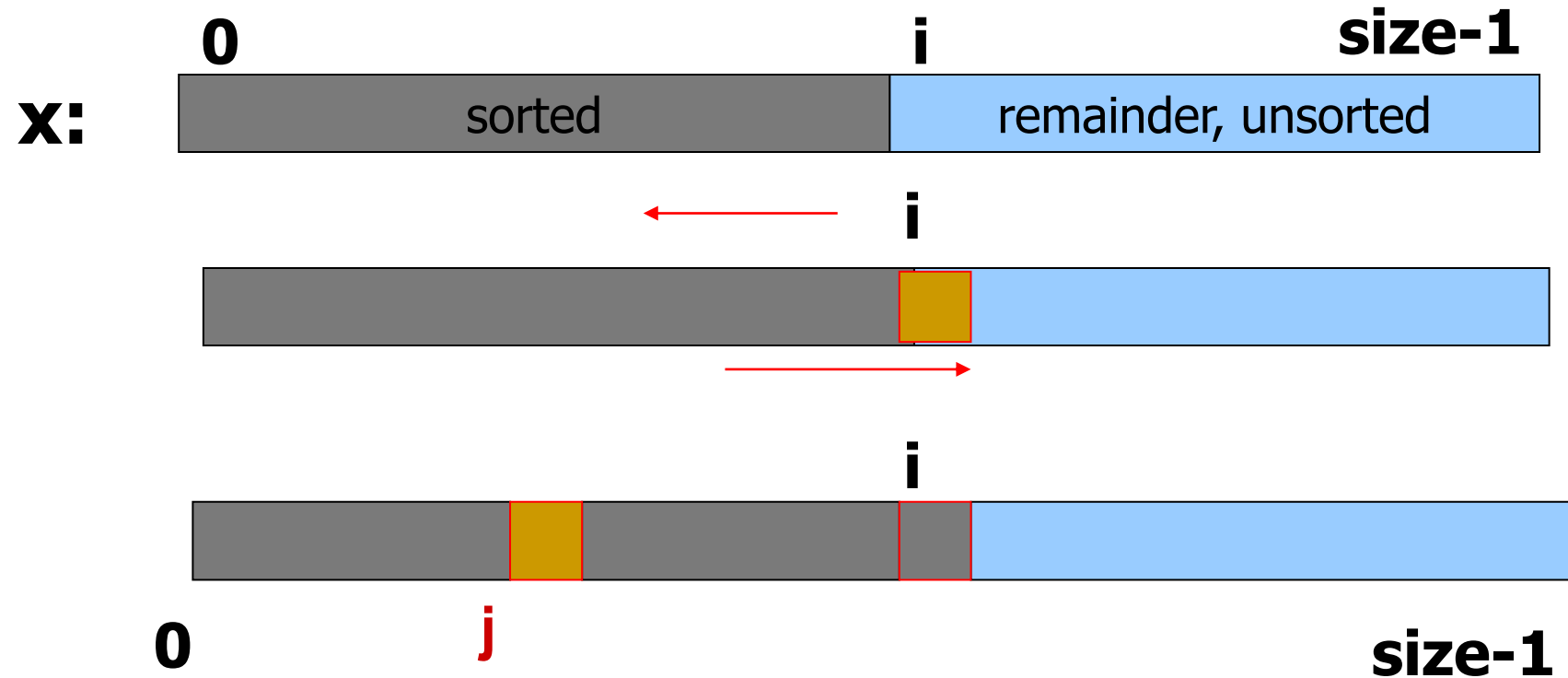- **What is the number of comparisons?**

$$(n-1)+(n-2)+\ldots\ldots+1= n(n-1)/2$$

Of the order of n^2

- **Worst Case? Best Case? Average Case?**

INSERTION SORT

General situation :

Insertion Sort

```
void insert_sort ( int list[ ], int size )
{
        int i,j,item;

        for (i=1; i<size; i++)
        {
                item = list[i] ;
                for (j=i-1; (j>=0)&& (list[j] > i); j--)  list[j+1] = list[j];
                list[j+1] = item ;
        }
}
```

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Time Complexity

Number of comparisons and shifting:

- **Worst case?**

  1 + 2 + 3 + …… + (n-1) = n(n-1)/2

- **Best case?**

  1 + 1 + …… to (n-1) terms = (n-1)

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Some Efficient Sorting Methods

Two of the most popular sorting algorithms are based on **divide-and-conquer** approach.

- **Quick sort**
- **Merge sort**

Basic idea (divide-and-conquer method):

```
sort (list)
{
        if the list has length greater than 1
        {
                Partition the list into lowlist and highlist;
                sort (lowlist);
                sort (highlist);
                combine (lowlist, highlist);
        }
}
```

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

QUICKSORT

At every step, we select a *pivot element* in the list (usually the first element).

- We put the pivot element in the *final position* of the sorted list.
- All the elements less than or equal to the pivot element are to the left.
- All the elements greater than the pivot element are to the right.

Partitioning
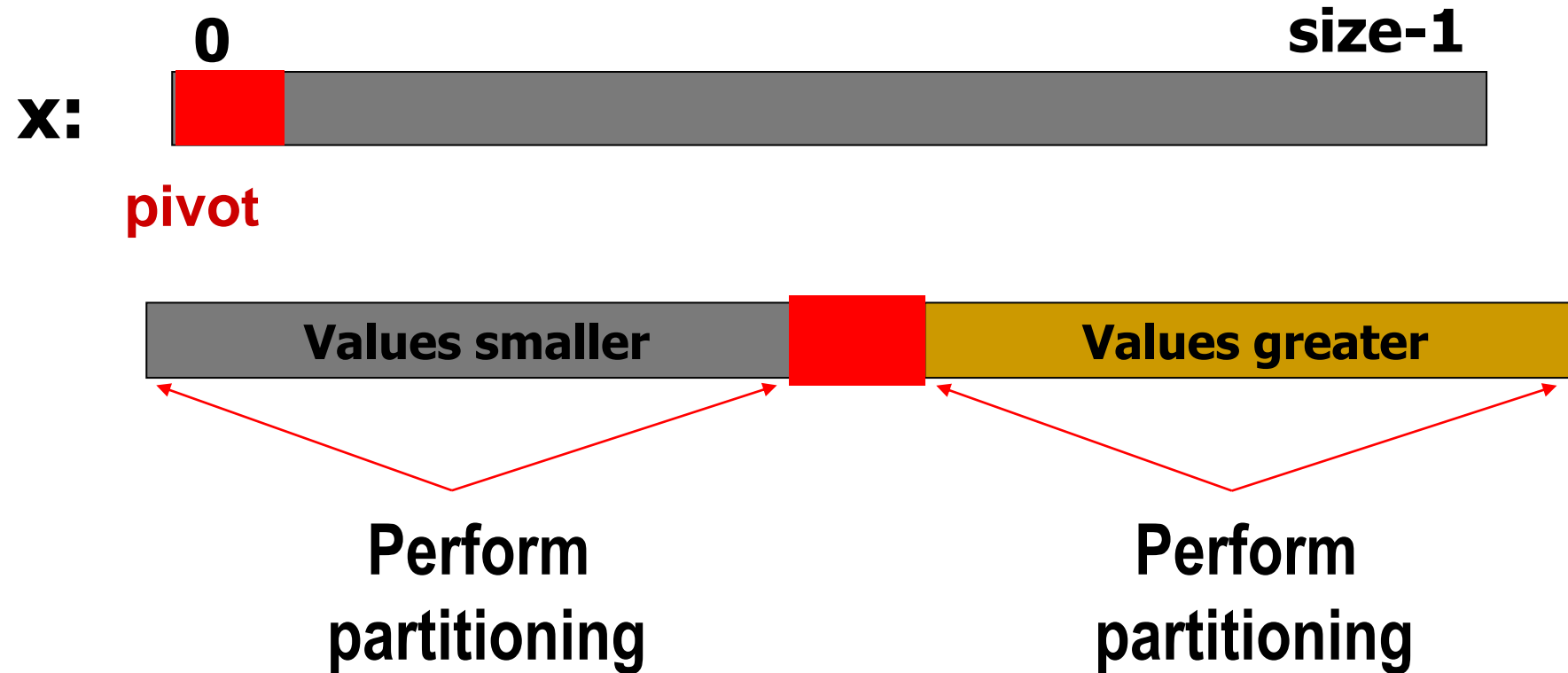
Preparation

```c
void print (int x[], int low, int high)
{
        int i;
        for(i=low; i<=high; i++) printf(" %d", x[i]);
        printf("\n");
}


void swap (int *a, int *b)
{
        int tmp=*a;   *a=*b;    *b=tmp;
}
```

Quicksort

```
void partition ( int x[ ], int low, int high )
{
    int i = low+1,  j = high;
    int pivot = x[low];

    if (low >= high) return;
    while (i<j)  {
      while ((x[i]<pivot) && (i<high))  i++;
      while ((x[j]>=pivot) && (j>low))  j--;
      if (i<j)  swap (&x[i], &x[j]);
    }

    if (j==high)  {
            swap (&x[j], &x[low]);
            partition (x, low, high-1);
    }
    else if (i==low+1)
            partition (x, low+1, high);
    else {
            swap (&x[j], &x[low]);
            partition (x, low, j-1);
            partition (x, j+1, high);
    }
}
```
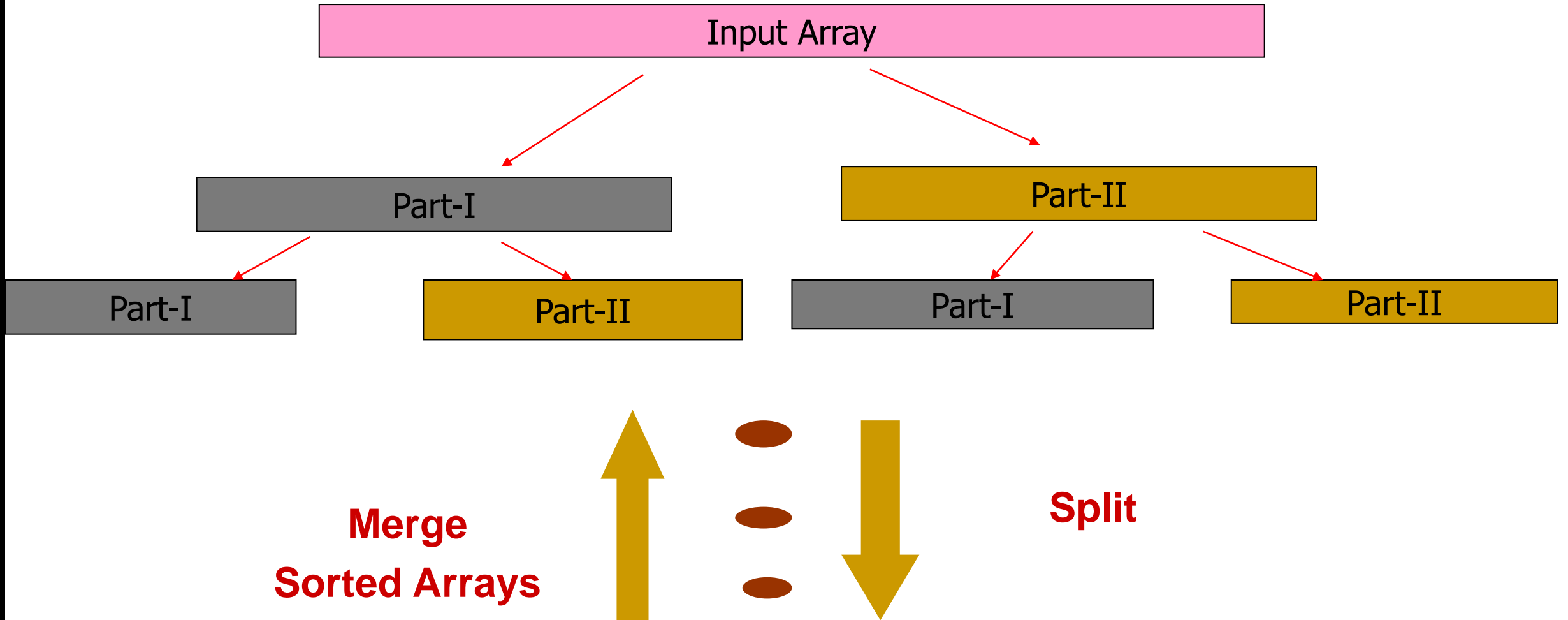
Time Complexity

Worst case:

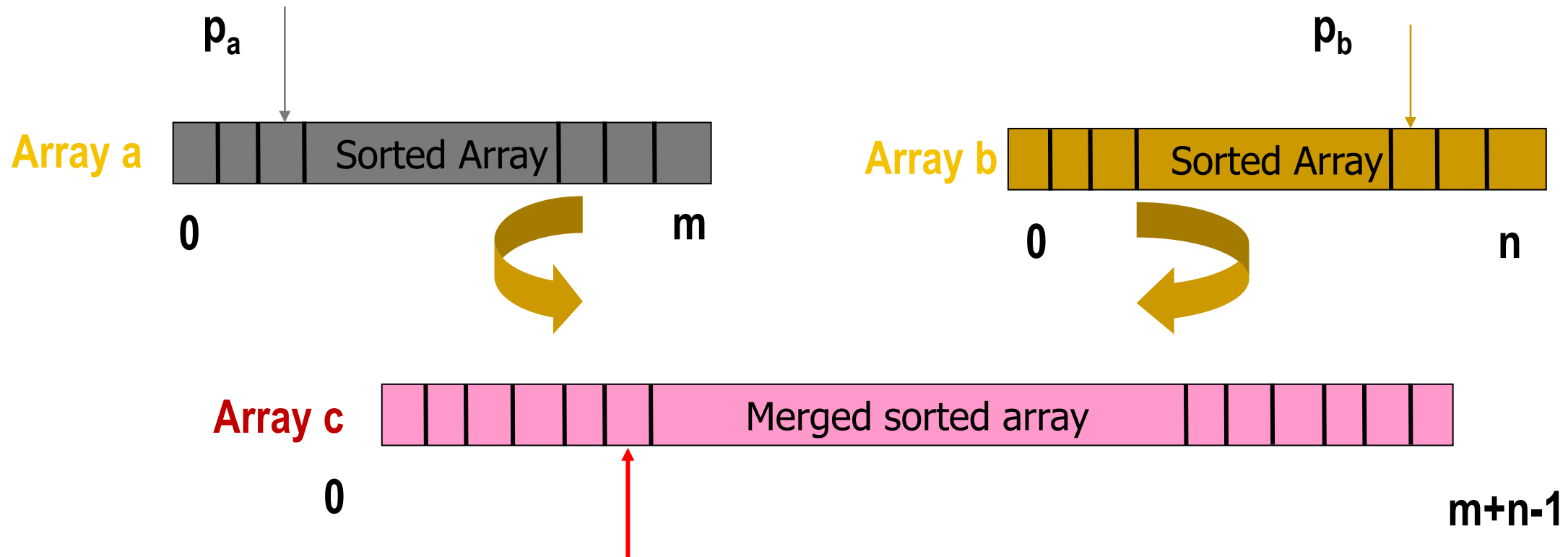n^2 ==> list is already sorted

Average case:

$n \log_2 n$

Statistically, quick sort has been found to be one of the fastest algorithms.

Merge Sort

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Merging two sorted arrays

Move and copy elements pointed by p_a if its value is smaller than the element pointed by p_b in (m+n-1) operations; otherwise, copy elements pointed by p_b.

Example

Initial array A contains 14 elements:

- 66, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30

Pass 1 :: Merge each pair of elements

- (33, 66) (22, 40) (55, 88) (11, 60) (20, 80) (44, 50) (30, 70)

Pass 2 :: Merge each pair of pairs

- (22, 33, 40, 66) (11, 55, 60, 88) (20, 44, 50, 80) (30, 77)

Pass 3 :: Merge each pair of sorted quadruplets

- (11, 22, 33, 40, 55, 60, 66, 88) (20, 30, 44, 50, 77, 80)

Pass 4 :: Merge the two sorted subarrays to get the final list

- (11, 20, 22, 30, 33, 40, 44, 50, 55, 60, 66, 77, 80, 88)

```c
void merge_sort ( int *a, int n )
{
    int i, j, k, m;
    int *b, *c;

    if (n>1)  {
        k = n/2;   m = n–k;
        b = (int *) calloc(k,sizeof(int));
        c = (int *) calloc(m,sizeof(int));
        for (i=0; i<k; i++)  b[i] = a[i];
        for (j=k; j<n; j++)  c[j– i] = a[j];

        merge_sort (b, k);
        merge_sort (c, m);
        merge (b, c, a, k, m);
        free(b); free(c);
    }
}
```

```c
void merge (int *a, int *b, int *c, int m, int n)
{
    int i=0, j=0, k=0, p;

    do  {
            if (a[i] < b[j]) { c[k]=a[i]; i++; }
            else { c[k]=b[j]; j++; }
            k++;
     } while ((i<m) && (j<n));

    if (i == m) {
            for (p=j; p<n; p++)  { c[k]=b[p]; k++; }
    }
    else  {
            for (p=i; p<m; p++)  { c[k]=a[p]; k++; }
    }
}
```