Sorting

Dr. Animesh Chaturvedi
Assistant Professor: IIIT Dharwad
Young Researcher: Heidelberg Laureate Forum
Postdoc: King's College London & The Alan Turing Institute
PhD: IIT Indore MTech: IIITDM Jabalpur

Sorting – The Task

- Given an array x[0], x[1], … , x[size-1]

reorder entries so that

x[0] <= x[1] <= . . . <= x[size-1]

- Here, List is in non-decreasing order.

- Also, sort a list of elements in non-increasing order.

Sorting – Example

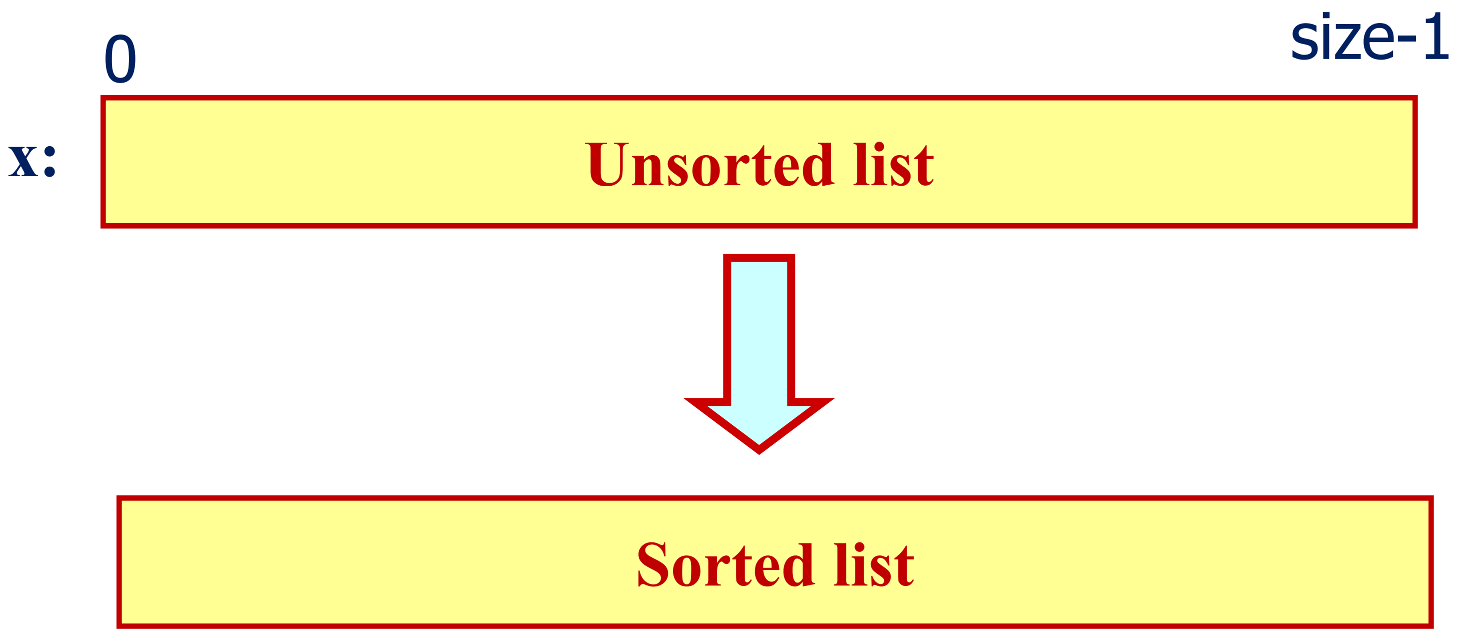
- Original list:
  - 10, 30, 20, 80, 70, 10, 60, 40, 70

- 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

Sorting Problem

- What do we want: Data to be sorted in order



Issues in Sorting

- Many issues are there in sorting techniques
  - How to rearrange a given set of data?
  - Which data structures are more suitable to store data prior to their sorting?
  - How fast the sorting can be achieved?
  - How sorting can be done in a memory constraint situation?
  - How to sort various types of data?

Sorting Algorithms

Sorting by Comparison

- A data item is compared with other items in the list of items in order to find its place in the sorted list.
- Operation
  - Insertion
  - Selection
  - Exchange
  - Enumeration

Sorting by Comparison

- Sorting by comparison – Insertion:
  - From a given list of items, one item is considered at a time. The item chosen is then inserted into an appropriate position relative to the previously sorted items. The item can be inserted into the same list or to a different list.
  - Insertion sort

- Sorting by comparison – Selection:
  - First the smallest (or largest) item is located and it is separated from the rest; then the next smallest (or next largest) is selected and so on until all item are separated.
  - Selection sort, Heap sort

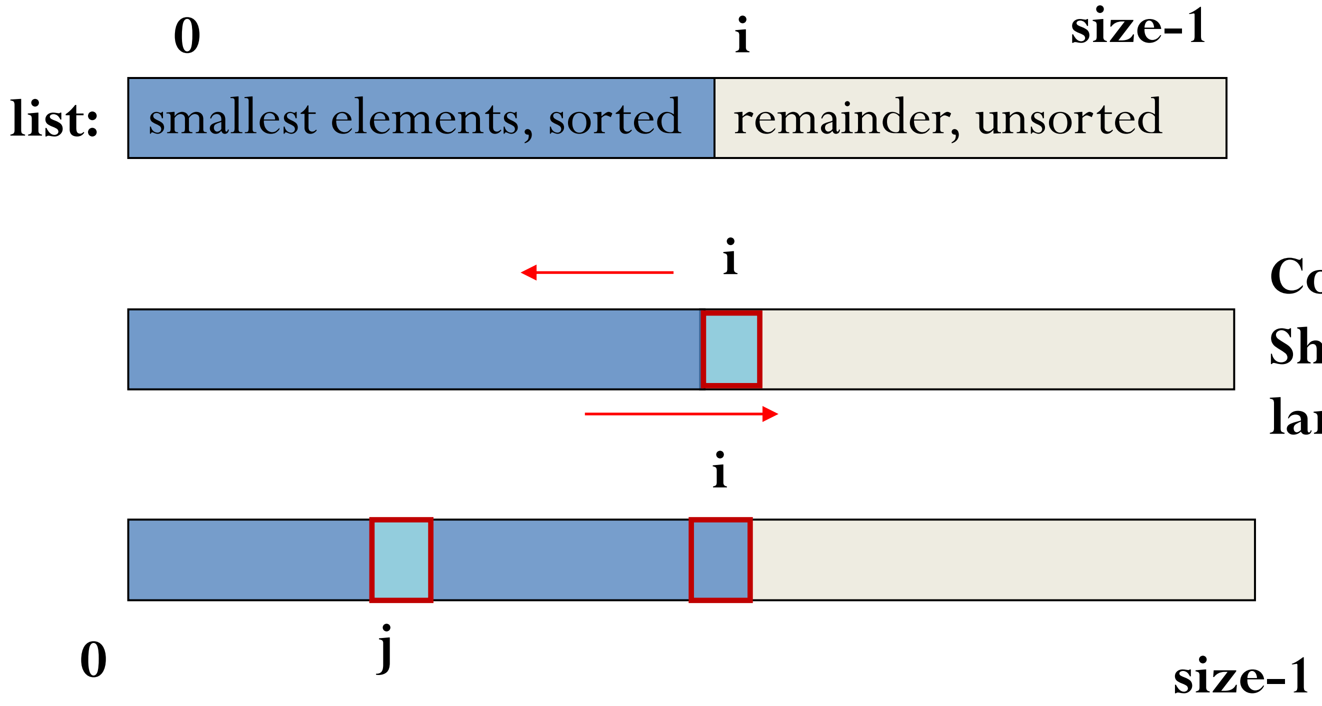
Sorting by Comparison

- Sorting by comparison – Exchange:
  - If two items are found to be out of order, they are interchanged. The process is repeated until no more exchange is required.
  - Bubble sort, Shell Sort, Quick Sort

- Sorting by comparison – Enumeration:
  - Two or more input lists are merged into an output list and while merging the items, an input list is chosen following the required sorting order.
  - Merge sort

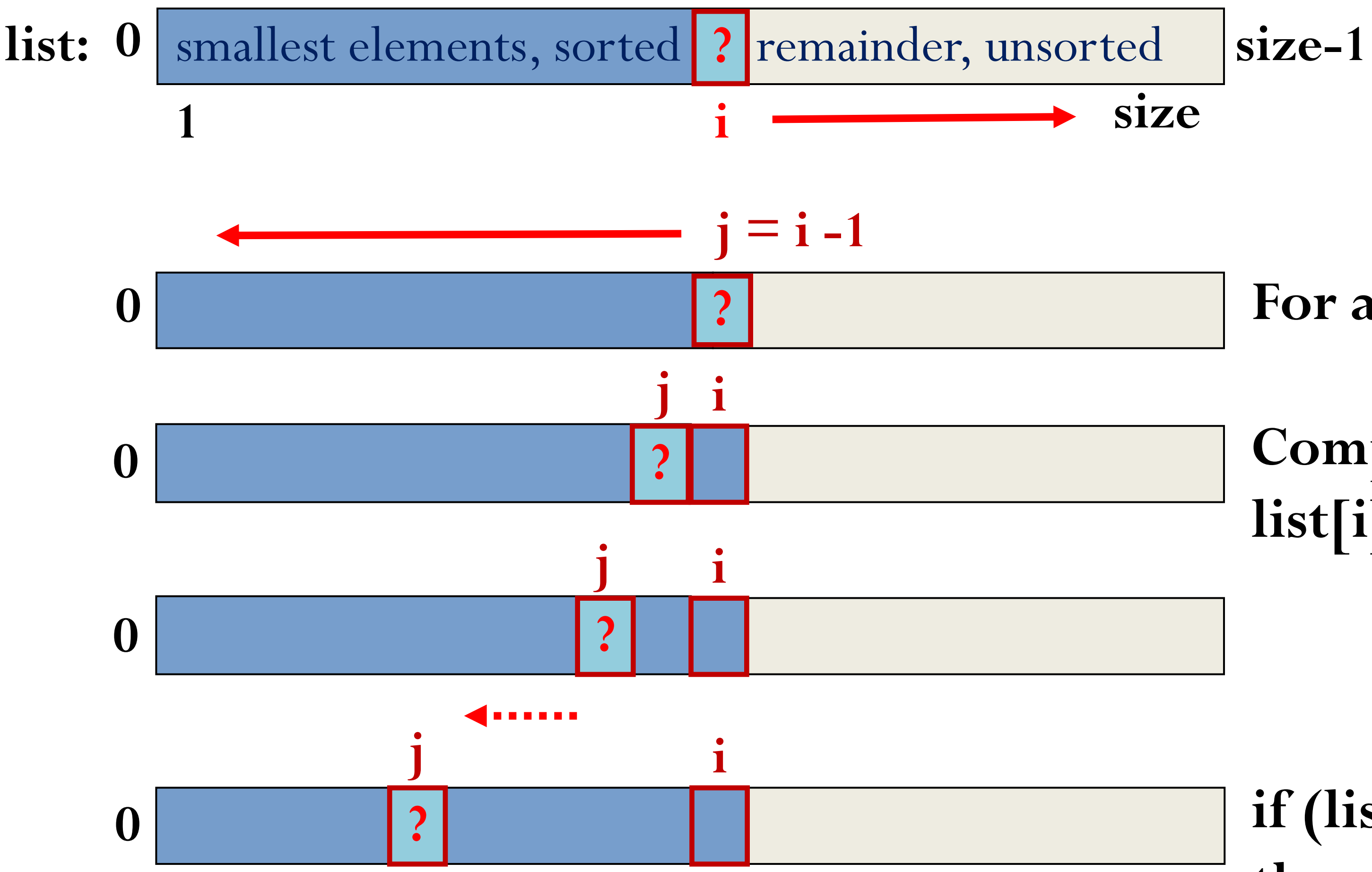
Insertion Sort

Insertion Sort



Compare and Shift till list[i] is larger.

Insertion Sort



For an i^{th} pass,

Compare and Shift till list[i] is larger than item.

if (list[j] > item) is false, then break the pass

Insertion Sort

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

        for (i=1; i<size; i++)
        {
                item = list[i] ;

/* Move elements of list[0..i-1], that are greater than item, to one
position ahead of their current position */

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

Insertion Sort

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

        for (i=1; i<size; i++)
        {       item = list[i] ;

/* Move elements of list[0..i-1], that are greater than item, to one position ahead
of their current position */

                for (j=i-1; j>=0; j--){
                        if (list[j] > item){
                                list[j+1] = list[j];}
                        else{break;}
                list[j+1] = item ;
                }
        }
}
```

Insertion Sort (while loop)

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

        for (i=1; i<size; i++)
        {       item = list[i];
                j = i - 1;

/* Move elements of list[0..i-1], that are greater than item, to one position ahead
of their current position */
                while(j>=0; && list[j] > item)
                        { list[j+1] = list[j];
                          j--;
                        }
                list[j+1] = item ;
                }
        }
}
```

Insertion Sort

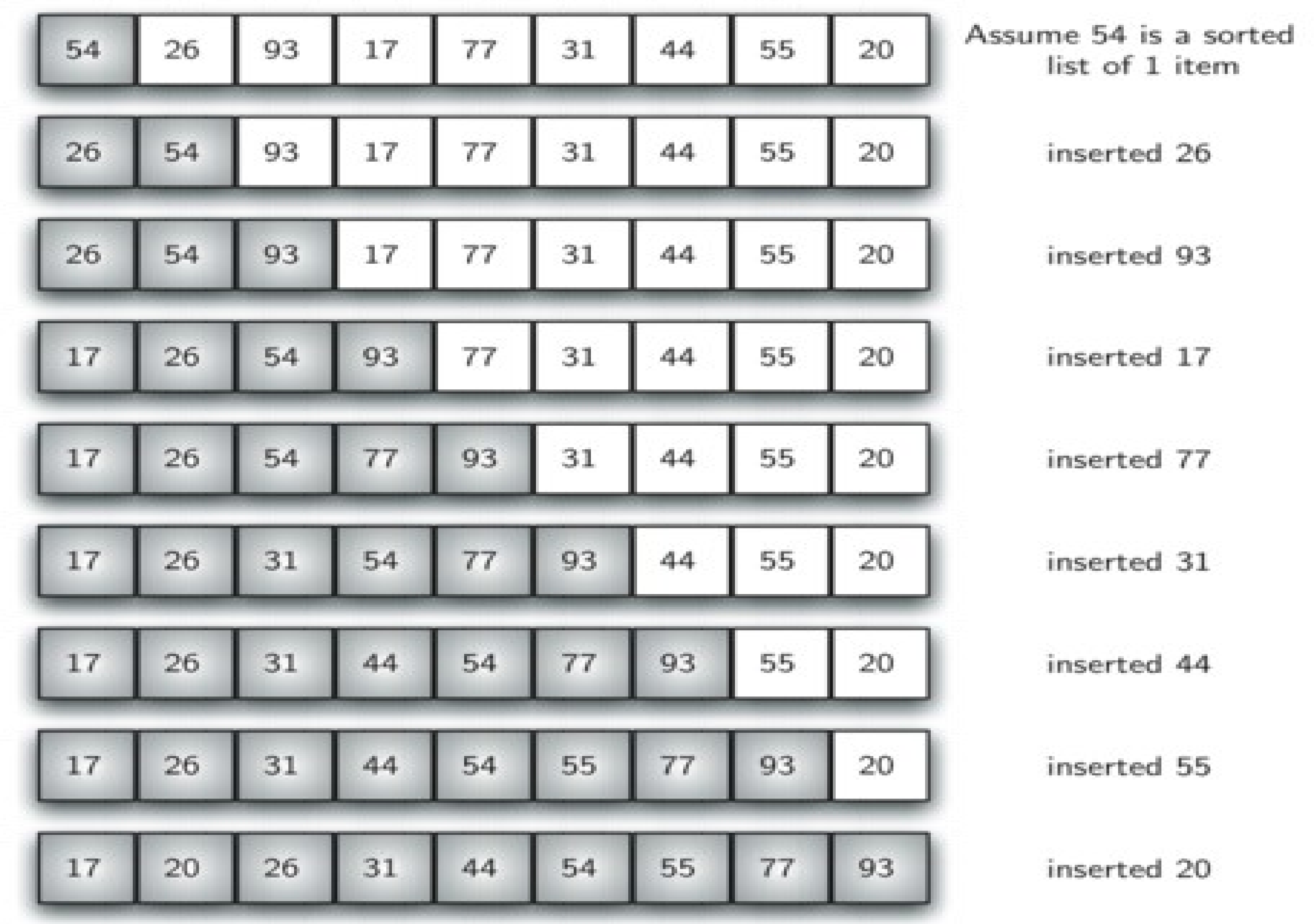
```
int main()
{
    int x[ ]={-45,89,-65,87,0,3,-23,19,56,21,76,-50};

    int i;
    for(i=0;i<12;i++)
       printf("%d ",x[i]);
    printf("\n");

    insertionSort(x,12);

    for(i=0;i<12;i++)
       printf("%d ",x[i]);
    printf("\n");
}
```

OUTPUT

-45 89 -65 87 0 3 -23 19 56 21 76 -50

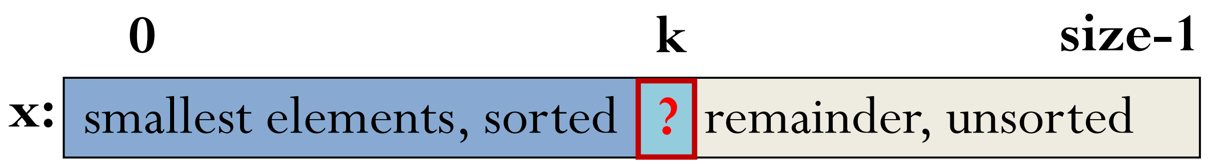
-65 -50 -45 -23 0 3 19 21 56 76 87 89

Insertion Sort



Selection Sort

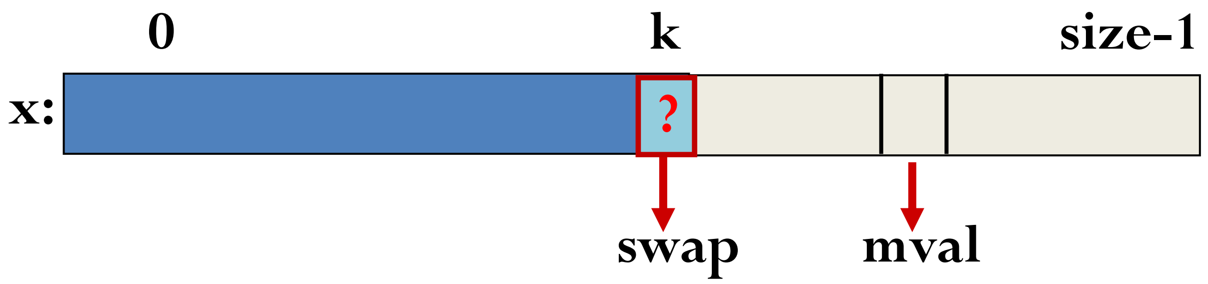
Selection Sort



Steps :

- **Find smallest element, mval, in x[k…size-1]**

- **Swap smallest element with x[k], then increase k.**



Selection Sort

```
/* The main sorting function */

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

int selectionSort (int x[], int size)
{int k, m, temp;
 for (k=0; k<size-1; k++) {
       m = findMinLoc(x, k, size);
       temp = a[k];
       a[k] = a[m];
       a[m] = temp;
       }
}
```

```
/* Yield location of smallest element in
x[k .. size-1];*/

int findMinLloc (int x[ ], int k, int size)
{
    int j, pos;
/* x[pos] is the smallest element found so
far */
      pos = k;
      for (j=k+1; j<size; j++)
           if (x[j] < x[pos])
                pos = j;
      return pos;
}
```

Selection Sort

```
int selectionSort (int x[], int size)
{   int k, min, temp;
        for (k=0; k < size-1; k++)
        {       for (j=k+1; j<size; j++){
                        if (x[j] < x[min])
                                { min = j;   }

                        temp = a[k];
                        a[k] = a[min];
                        a[min] = temp;
        }
}
```

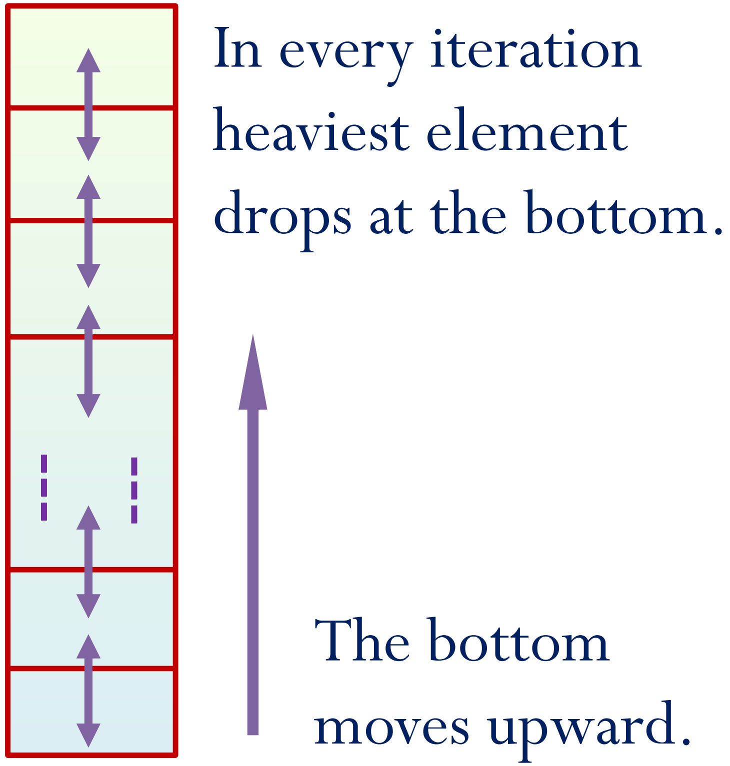
Selection Sort – (7 pass of external loop)

Pass 1 Input	x:	3	12	-5	6	142	21	-17	45
Pass 1 Output	x:	-17	12	-5	6	142	21	3	45
Pass 2 Output	x:	-17	-5	12	6	142	21	3	45
Pass 3 Output	x:	-17	-5	3	6	142	21	12	45
Pass 4 Output	x:	-17	-5	3	6	142	21	12	45
Pass 5 Output	x:	-17	-5	3	6	12	21	142	45
Pass 6 Output	x:	-17	-5	3	6	12	21	142	45
Pass 7 Output	x:	-17	-5	3	6	12	21	45	142

Bubble Sort

Bubble Sort

- The sorting process proceeds in several passes.
- In every pass, compare neighboring pairs, and swap them if out of order.
- In every pass, the largest of the elements under considering will bubble to the top (i.e., the right).



Bubble Sort

- How the passes proceed?
  - In pass 1, we consider index 0 to n-1.
  - In pass 2, we consider index 0 to n-2.
  - In pass 3, we consider index 0 to n-3.
  - ……
  - ……
  - In pass n-1, we consider index 0 to 1.

Bubble Sort – Pass (internal loop)

- **Pass: 1**

x:	3	12	-5	6	72	21	-7	45
x:	3	12	-5	6	72	21	-7	45
x:	3	-5	12	6	72	21	-7	45
x:	3	-5	6	12	72	21	-7	45
x:	3	-5	6	12	72	21	-7	45
x:	3	-5	6	12	21	72	-7	45
x:	3	-5	6	12	21	-7	72	45
x:	3	-5	6	12	21	-7	45	72

Bubble Sort – Pass (internal loop)

- **Pass: 2**

x:	3	-5	6	12	21	-7	45	72

x:	-5	3	6	12	21	-7	45	72

x:	-5	3	6	12	21	-7	45	72

x:	-5	3	6	12	21	-7	45	72

x:	-5	3	6	12	21	-7	45	72

x:	-5	3	6	12	-7	21	45	72

x:	-5	3	6	12	-7	21	45	72

Bubble Sort

```
void swap(int *x, int *y)
{
        int tmp = *x;
        *x = *y;
        *y = tmp;
}

void bubble_sort(int x[], int n)
{
        int i,j;
        for (i=n-1; i>0; i--)
            for (j=0; j<i; j++)
               if (x[j] > x[j+1])
                   swap(&x[j],&x[j+1]);
}
```

```
int main()
{
   int x[ ]={-45,89,-65,87,0,3,-23,19,56,21,76,-50};
   int i;
   for(i=0;i<12;i++)
      printf("%d ",x[i]);
   printf("\n");

   bubble_sort(x,12);

   for(i=0;i<12;i++)
      printf("%d ",x[i]);
   printf("\n");
}
```

OUTPUT

-45 89 -65 87 0 3 -23 19 56 21 76 -50

-65 -50 -45 -23 0 3 19 21 56 76 87 89

Bubble Sort

- How do you make best case with (n-1) comparisons only?
  - By maintaining a variable flag,
    - to check if there has been any swaps in a given pass.
  - If not, the array is already sorted.

```
void bubble_sort(int x[], int n)
{
  int i,j;
  int flag = 0;
   for (i=n-1; i>0; i--)
   {
     for (j=0; j<i; j++)
     if (x[j] > x[j+1])
     {
       swap(&x[j],&x[j+1]);
       flag = 1;
     }
     if (flag == 0) return;
   }
}
```

Quick Sort

Efficient Sorting algorithms

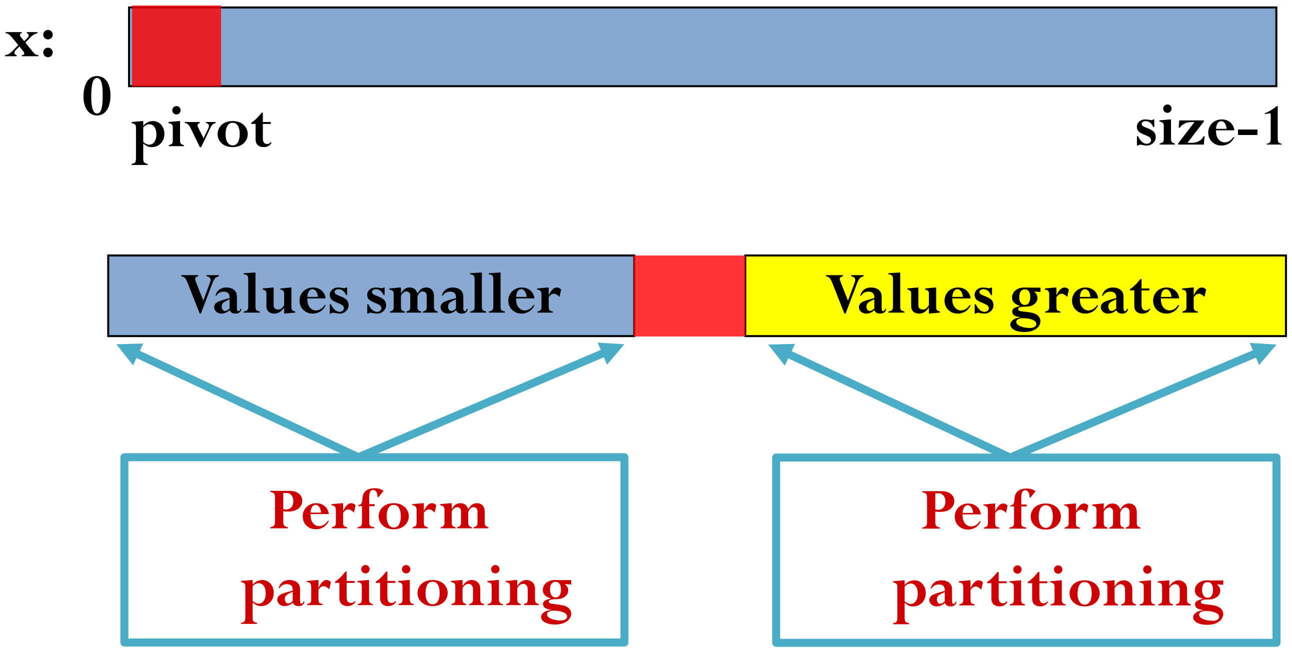
- Two of the most popular sorting algorithms are based on divide-and-conquer approach.
  - Quick sort
  - Merge sort

Basic concept of 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);
    }
}
```

Quick Sort – How it Works?

- **At every step, select a *pivot element* in the list (usually 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*.



Quick Sort

```
int partition( int a[], int l, int r)
{  int pivot, i, j, t;
   pivot = a[l];
   i = l;
   j = r+1;
   while(1) {
          do { ++i;
             } while(a[i]<=pivot && i<=r);
          do { --j;
             } while( a[j] > pivot );
          if( i >= j ) break;
          t = a[i];
          a[i] = a[j];
          a[j] = t;
   }
   t = a[l];
   a[l] = a[j];
   a[j] = t;
   return j;
}
```

```
#include <stdio.h>
void quickSort( int[], int, int);
int partition( int[], int, int);
void main()
{
   int i,a[] = { 7, 12, 1, -2, 0, 15, 4, 11, 9};
   printf("\n\nUnsorted array is: ");
   for(i = 0; i < 9; ++i)
      printf(" %d ", a[i]);
   quickSort( a, 0, 8);
   printf("\n\nSorted array is: ");
   for(i = 0; i < 9; ++i)
      printf(" %d ", a[i]);
}
void quickSort( int a[], int l, int r)
{
   int j;
   if( l < r ) {  // divide and conquer
   j = partition( a, l, r);
   quickSort( a, l, j-1);
   quickSort( a, j+1, r);
   }
}
```

Quick Sort - Example

Input 45 -56 78 90 -3 -6 123 0 -3 45 69 68

45 -56 78 90 -3 -6 123 0 -3 45 69 68

-6 -56 -3 0 -3 45 123 90 78 45 69 68

-56 -6 -3 0 -3 68 90 78 45 69 123

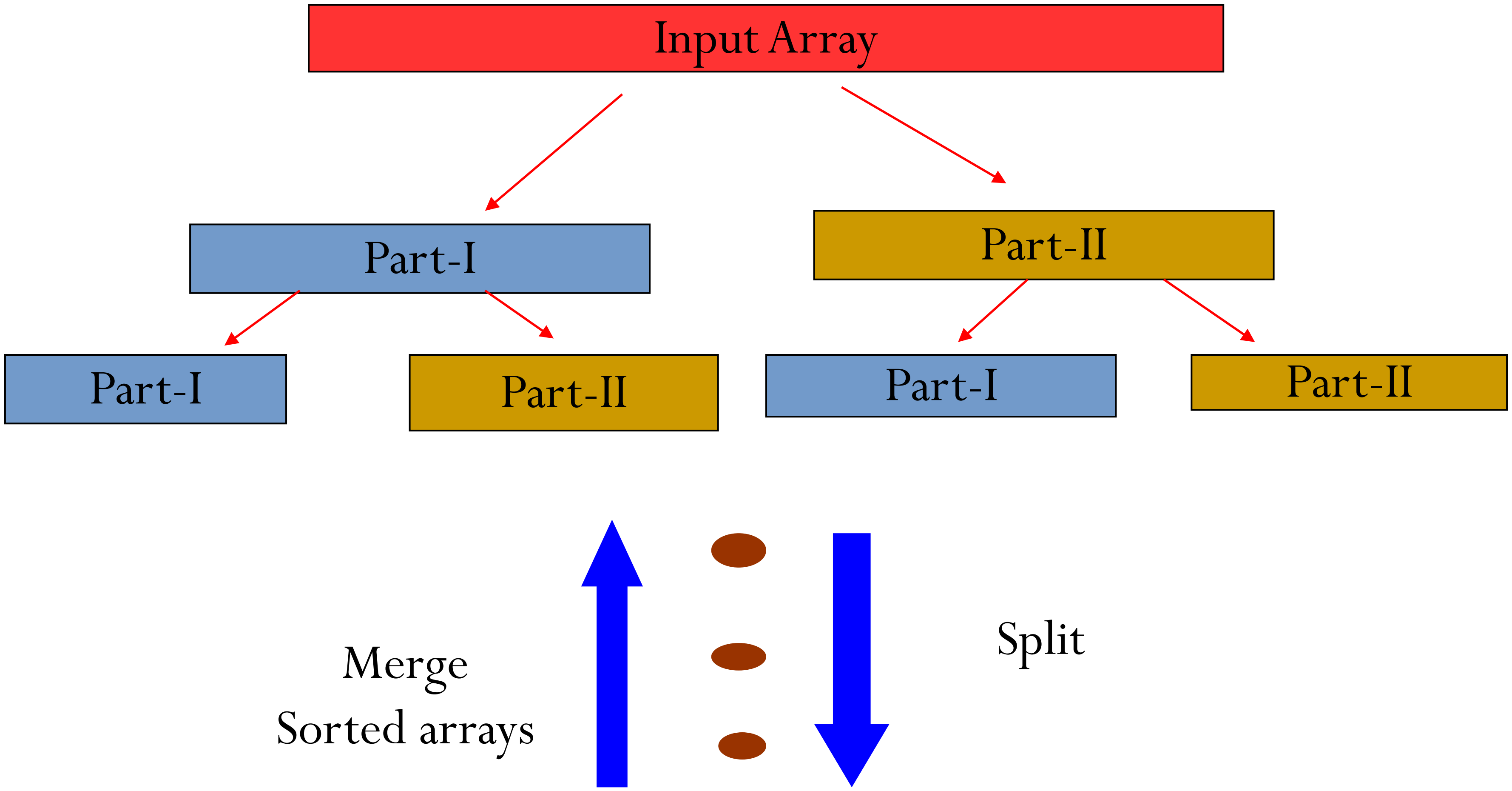
-3 0 -3 45 68 78 90 69

-3 0 69 78 90

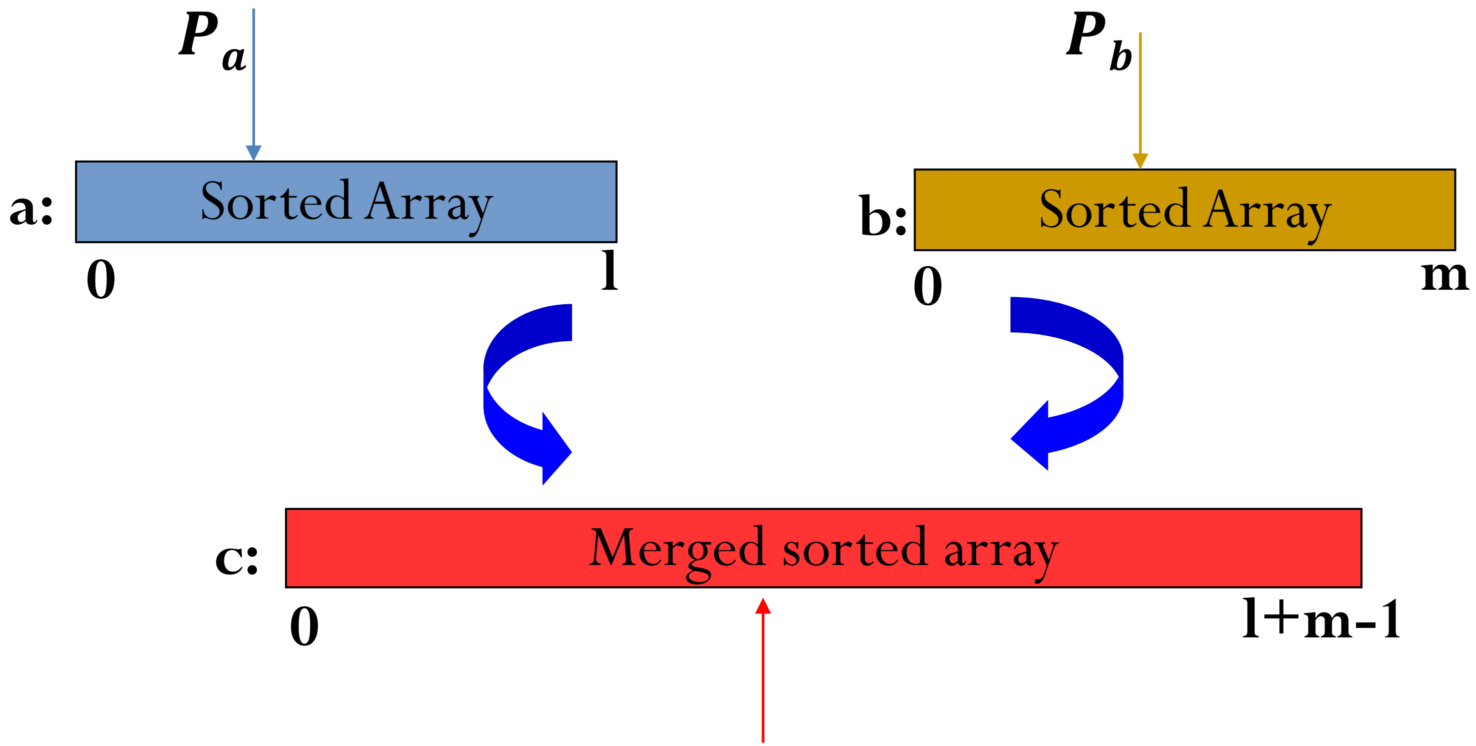
Output -56 -6 -3 -3 0 45 45 68 69 78 90 123

Merge Sort

Merge Sort – How it Works?

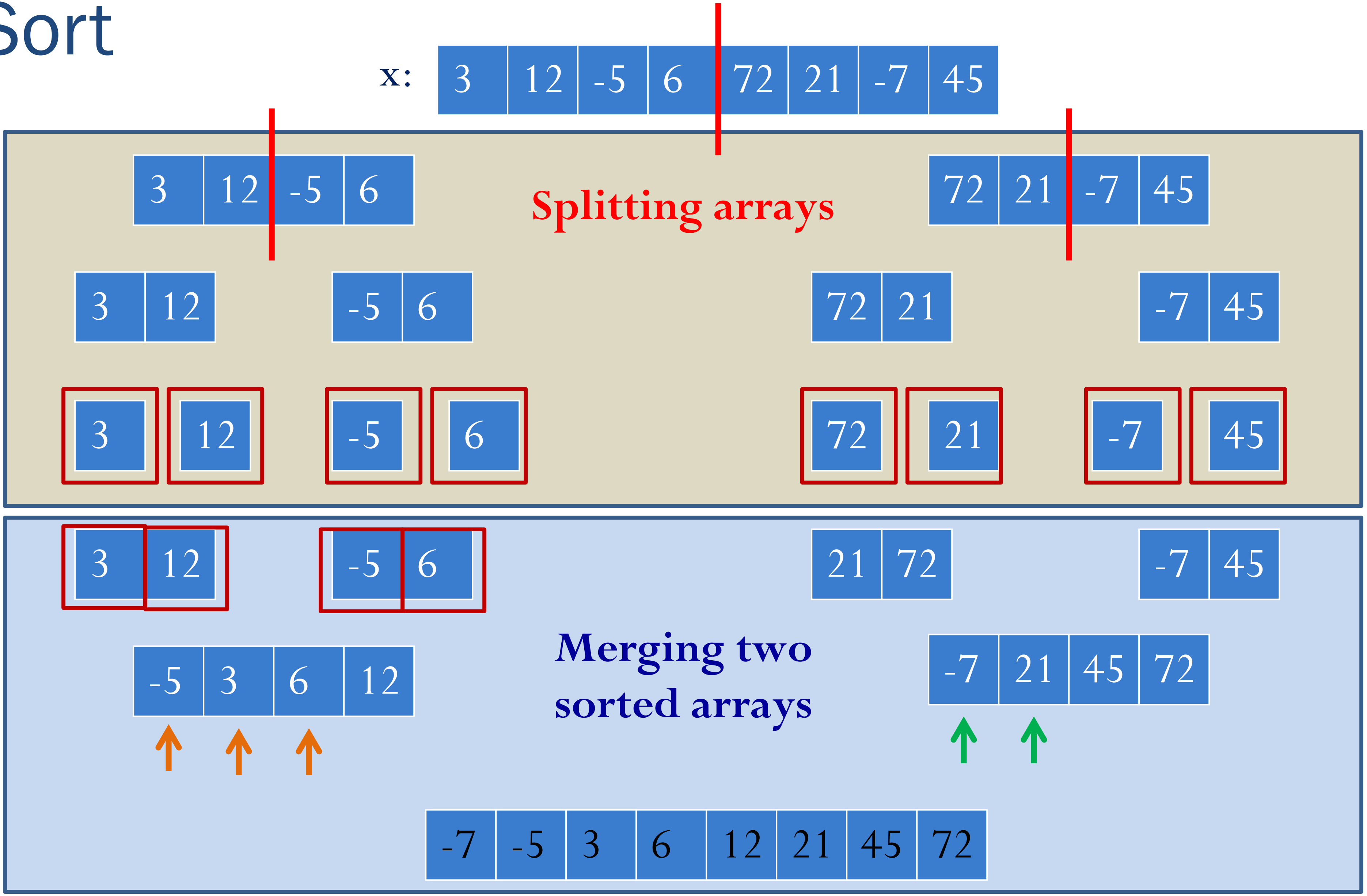


Merging two Sorted arrays



Move and copy elements pointed by P_a if its value is smaller than the element pointed by P_b in $(l + m - 1)$ operations and otherwise.

Merge Sort



x:
3 12 -5 6 72 21 -7 45
3 12 -5 6
Splitting arrays
72 21 -7 45
3 12
-5 6
72 21
-7 45
3
12
-5
6
72
21
-7
45
3 12
-5 6
21 72
-7 45
-5 3 6 12
Merging two sorted arrays
-7 21 45 72
-7 -5 3 6 12 21 45 72

Merge Sort Program

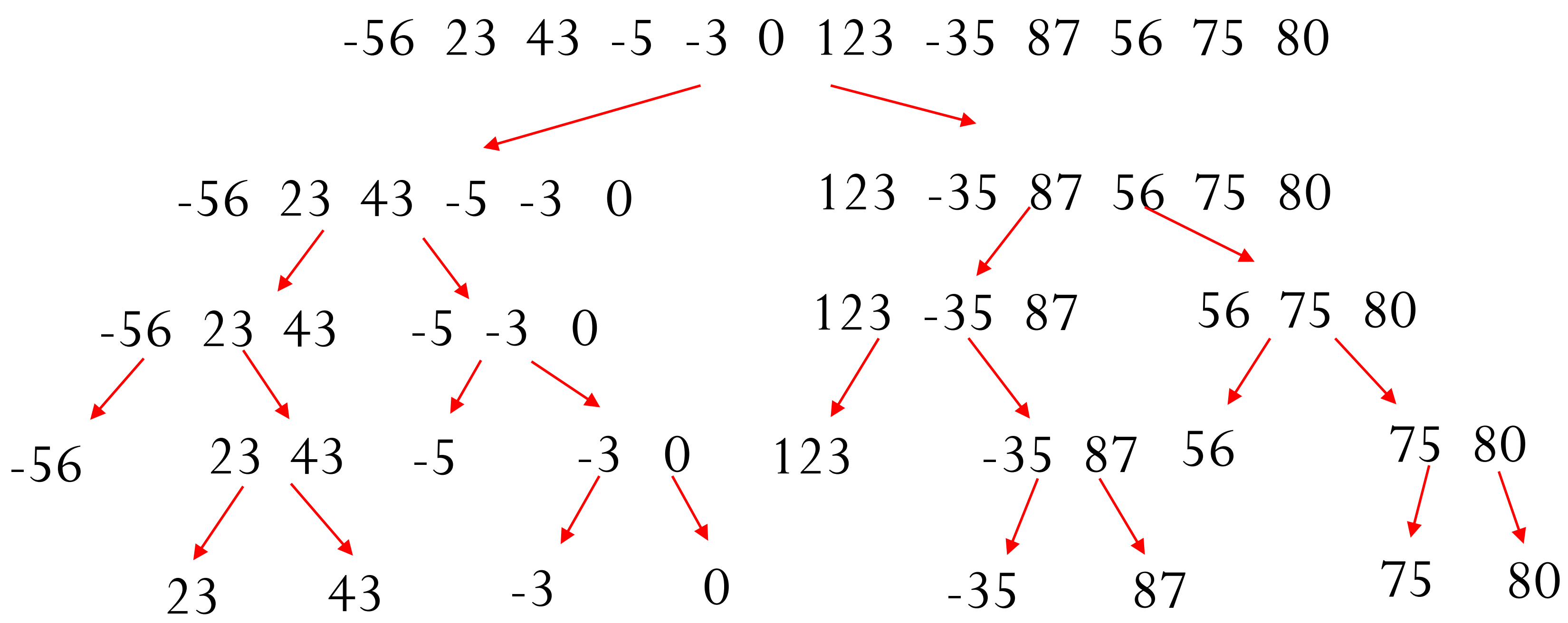
```c
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
    int a[30],n,i;
    printf("Enter no of elements:");
    scanf("%d",&n);
    printf("Enter array elements:");

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

    mergesort(a,0,n-1);

    printf("\nSorted array is :");
    for(i=0;i<n;i++)
      printf("%d ",a[i]);

    return 0;
}
```

```c
void mergesort(int a[],int i,int j)
{
     int mid;

     if(i<j)  {
     mid=(i+j)/2;
    /* left recursion */
     mergesort(a,i,mid);
    /* right recursion */
     mergesort(a,mid+1,j);
  /* merging of two sorted sub-arrays */
     merge(a,i,mid,mid+1,j);
     }
}
```

Merge Sort Program

```
void merge(int a[],int i1,int i2,int j1,int j2)
{    int temp[50]; //array used for merging
     int i=i1,j=j1,k=0;

     while(i<=i2 && j<=j2) //while elements in both lists
     { if(a[i]<a[j])
            temp[k++]=a[i++];
       else
            temp[k++]=a[j++];
      }

     while(i<=i2) //copy remaining elements of the first list
        temp[k++]=a[i++];

     while(j<=j2) //copy remaining elements of the second list
        temp[k++]=a[j++];

     for(i=i1,j=0;i<=j2;i++,j++)
     a[i]=temp[j]; //Transfer elements from temp[] back to a[]
}
```

Merge Sort – Splitting Trace



Output: -56 -35 -5 -3 0 23 43 56 75 80 87 123

Quick Sort vs. Merge Sort

- **Quick sort**
  - **hard division, easy combination**
  - partition in the divide step of the divide-and-conquer framework
  - hence combine step does nothing

- **Merge sort**
  - **easy division, hard combination**
  - merge in the combine step
  - the divide step in this framework does one simple calculation only

Quick Sort vs. Merge Sort

- Both the algorithms divide the problem into two sub problems.
  - **Merge sort:**
    - two sub problems are of almost equal size always.
  - **Quick sort:**
    - an equal sub division is not guaranteed.
- This difference between the two sorting methods appears as the deciding factor of their run time performances.

References

Debasis Samanta, Computer Science & Engineering, Indian Institute of Technology Kharagpur, Spring-2017, Programming and Data Structures. https://cse.iitkgp.ac.in/~dsamanta/courses/pds/index.html

תודה רבה
Hebrew

Ευχαριστώ
Greek

Спасибо
Russian

Danke
German

धन्यवादः
Sanskrit

Merci
French

ধন্যবাদ
Bangla

நன்றி
Tamil

شكراً
Arabic

ಧನ್ಯವಾದಗಳು
Kannada

Thank You
English

നന്ദി
Malayalam

Grazie
Italian

多謝
Traditional Chinese

ధన్యవాదాలు
Telugu

આભાર
Gujarati

ਧੰਨਵਾਦ
Punjabi

धन्यवाद
Hindi & Marathi

Gracias
Spanish

多谢
Simplified Chinese

https://sites.google.com/site/animeshchaturvedi07

Obrigado
Portuguese

ありがとうございました
Japanese

ขอบคุณ
Thai

감사합니다
Korean