Some Efficient Sorting Algorithms

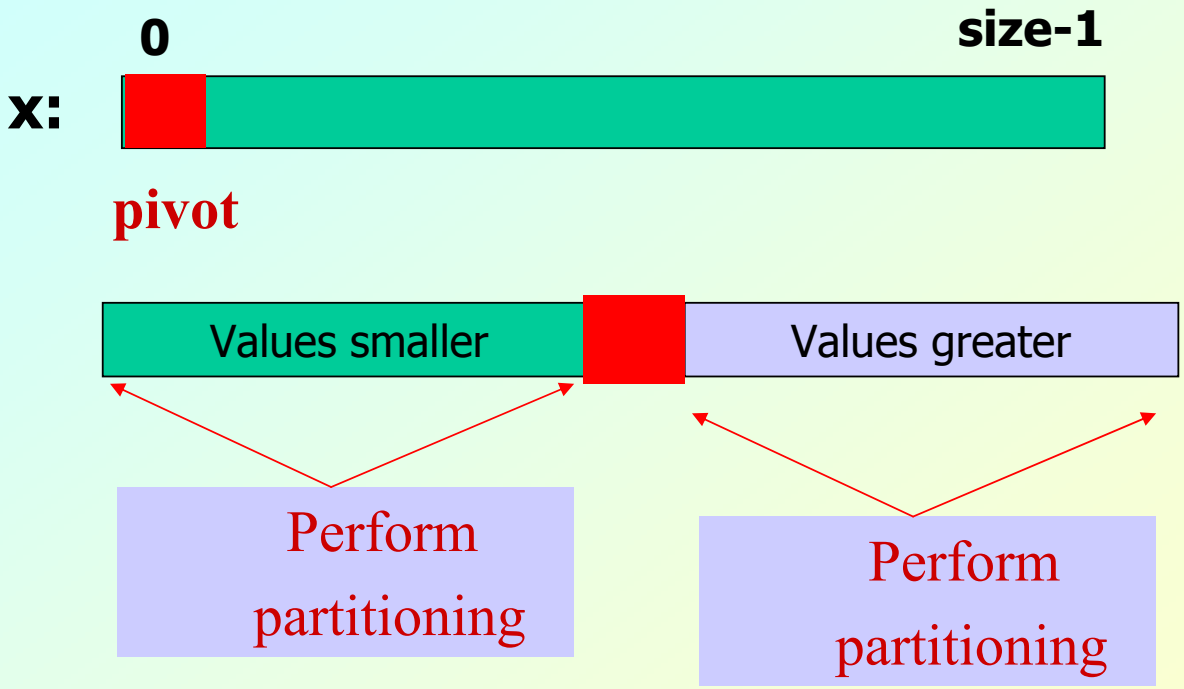
- Two of the most popular sorting algorithms are based on divide-and-conquer approach.
  - Quick sort
  - Merge sort
- Basic concept (divide-and-conquer method):

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

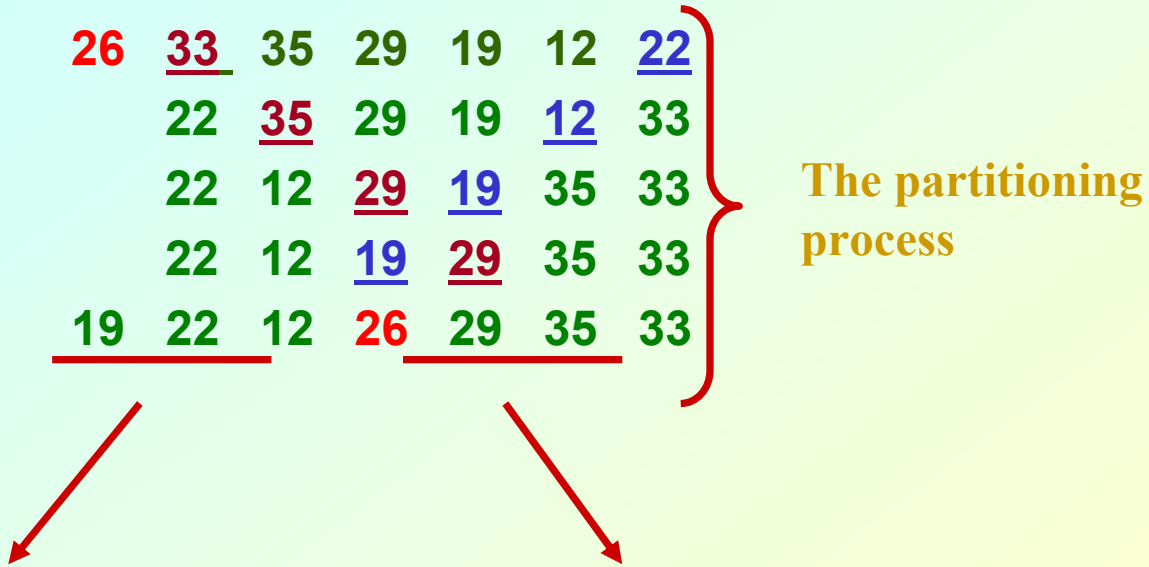
Quicksort

- 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



Example



Recursively carry out the partitioning

```
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;
}
```

```
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);
          }
}
```

```
int main (int argc, char *argv[])
{
   int *x;
   int i=0;
   int num;

   num = argc - 1;
   x = (int *) malloc(num * sizeof(int));

   for (i=0; i<num; i++)
     x[i] = atoi(argv[i+1]);

   partition(x,0,num-1);

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

Trace of Partitioning

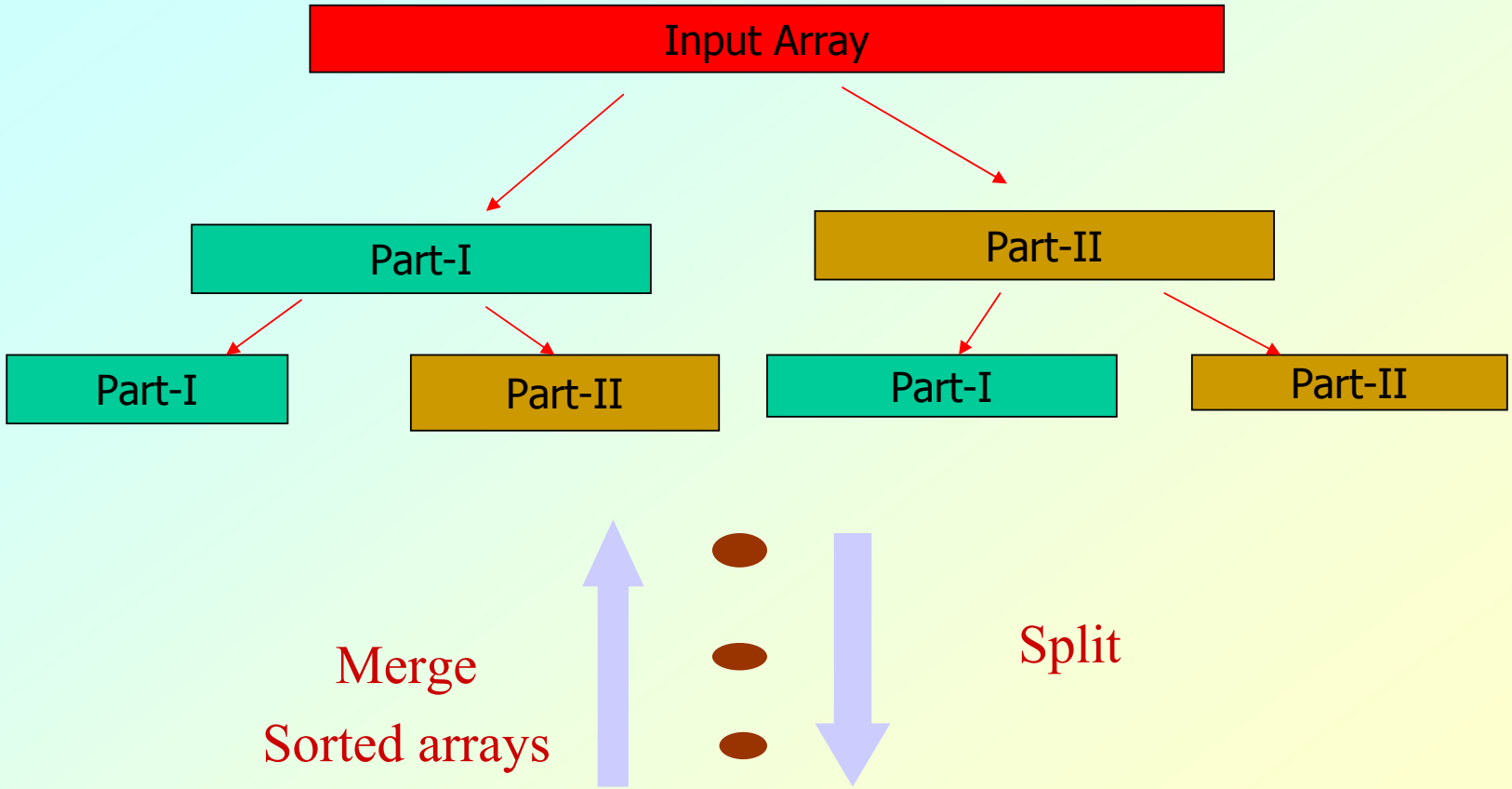
```
./a.out 45 -56 78 90 -3 -6 123 0 -3 45 69 68
```

```
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
```

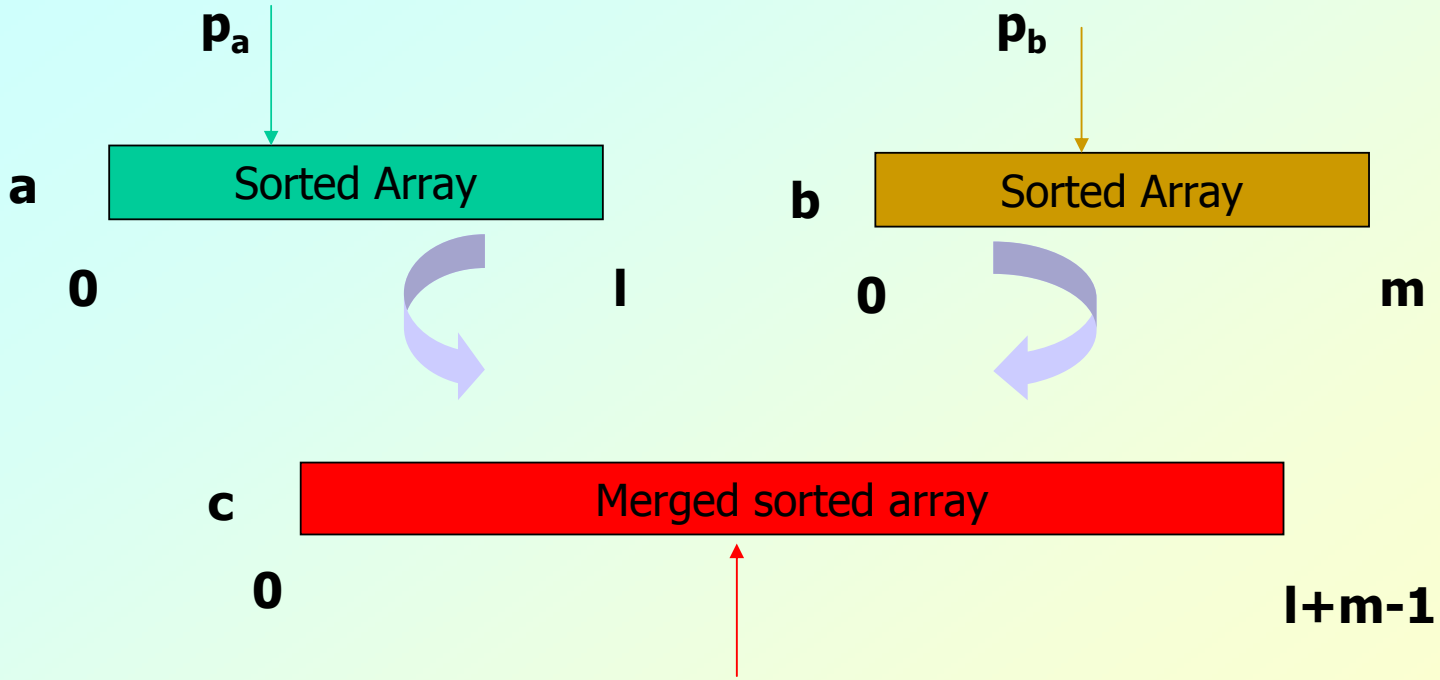
```
Sorted list:  -56 -6 -3 -3 0 45 45 68 69 78 90 123
```

Merge Sort

Merge Sort


Input Array
Part-I
Part-II
Part-I
Part-II
Part-I
Part-II
Merge
Sorted arrays
Split

Merging two sorted arrays



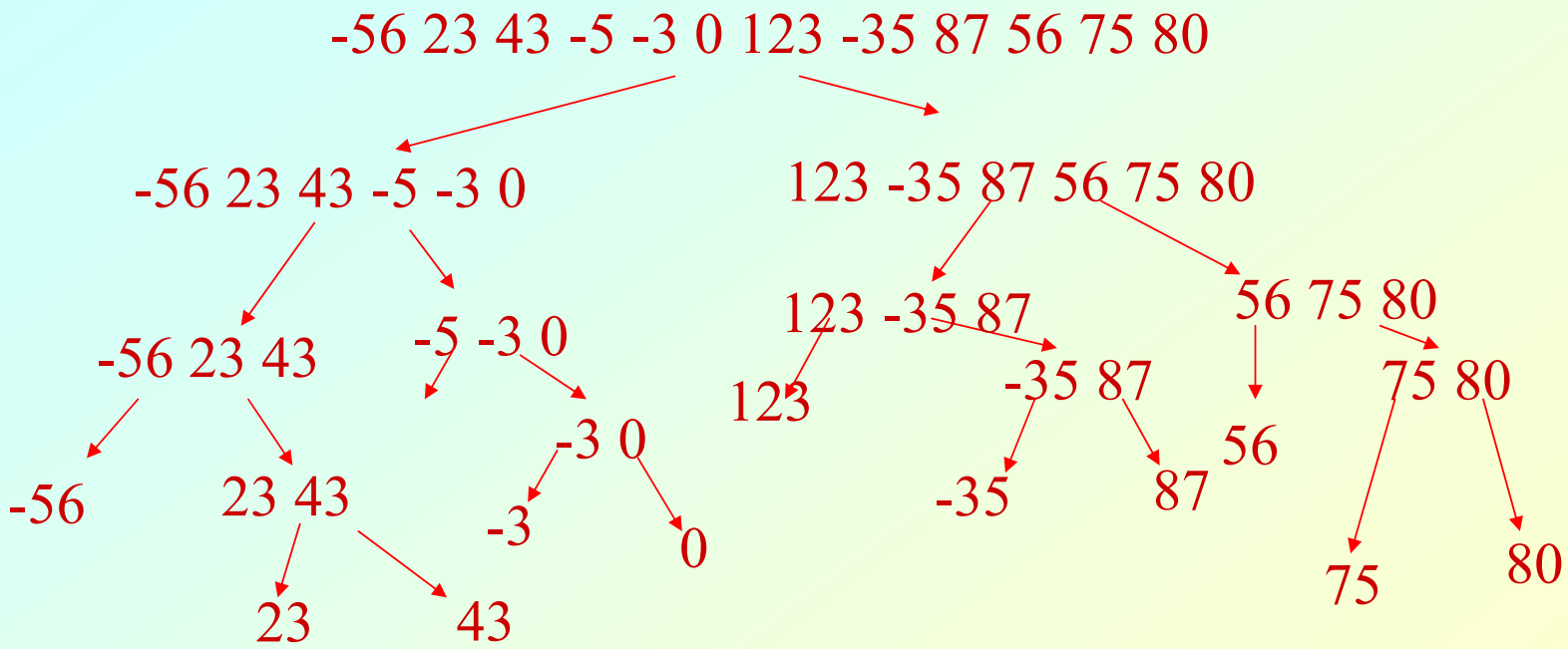
Move and copy elements pointed by p_a if its value is smaller than the element pointed by p_b in (l+m-1) operations and otherwise.

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

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

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

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

   i=j=k=0;

   do  {
     if (a[i] < b[j])  {
       c[k]=a[i]; i=i+1;
     }
     else  {
       c[k]=b[j]; j=j+1;
     }
     k++;
   }  while ((i<m) && (j<n));
   if (i == m) {
      for (l=j; l<n; l++)  { c[k]=b[l]; k++; }
   }
   else  {
      for (l=i; l<m; l++)  { c[k]=a[l]; k++; }
   }
}
```

```
main()
{
    int i, num;
    int a[ ] = {-56,23,43,-5,-3,0,123,-35,87,56,75,80};

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

    merge_sort (a, 12);

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

Splitting Trace



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